J Ambient Intell Human Comput (2016) 7:239–256
DOI 10.1007/s12652-015-0322-7

ORIGINAL RESEARCH

Searching optimal menu layouts by linear genetic programming

Luigi Troiano[1] · **Cosimo Birtolo**[2] · **Roberto Armenise**[2]

Received: 15 June 2015 / Accepted: 1 September 2015 / Published online: 15 September 2015

Abstract Designing effective menu systems is a key ingredient to usable graphical user interfaces. This task generally relies only on human ability in building hierarchical structures. However, trading off different and partially opposite guidelines, standards and practices is time consuming and can exceed human skills in problem solving. Recent advances are showing that this task can be addressed by generative approaches which exploit evolutionary algorithms as means for evolving different and unexpected solutions. The search of optimal solutions is made not trivial due to different alternatives which lead to local optima and constraints which can invalidate large sectors of the search space and make valid solutions sparse. This problem can be addressed by choosing an appropriate algorithm. In this paper we face the problem of searching optimal solutions by Linear Genetic Programming in particular, and we compare the solution to more conventional approaches based on simple genetic algorithms and genetic programming. Experimental results are discussed and compared to human-made solutions.

Keywords Genetic algorithm · Genetic programming · Menu layout · Search based software engineering

✉ Luigi Troiano
troiano@unisannio.it

Cosimo Birtolo
birtoloc@posteitaliane.it

Roberto Armenise
armenis5@posteitaliane.it

[1] Department of Engineering, University of Sannio, Viale Traiano, 82100 Benevento, Italy

[2] Poste Italiane S.p.A., SI/CCUP/GC, p.zza Matteotti 3, 80133 Napoli, Italy

1 Introduction

Navigation is one of the key factors in designing web applications, and menus play a key role in enhancing usability. Indeed menu structure can positively or negatively impact on accessibility to application functions. Ability in structuring information and designing user interfaces is generally considered in the realm of human creativity. More recently, generative approaches based on evolutionary computing are proving to challenge this belief. In this paper we attempt to generate hierarchical menu layouts by evolutionary algorithms, and facing solutions obtained so far to those provided by humans. Experimental results with genetic algorithms and genetic programming are discussed and compared.

Despite the technological shift which led to web and mobile applications, menu systems still represent a key ingredient in accessing application functions. Menu systems evolved in structure, features and purpose, but application usability still largely rely on their structure. Indeed, models of web applications do not refer to a common and shared metaphor, and a consistent navigation menu is one of the few design elements which provide users with some sense of orientation and guide them through the site. Users should be able to rely on it, and this makes the design of menu layout critical.

According to Kong et al. (2011), several frameworks and models have been recently proposed to support the design and development of interfaces and the process of designing an interface has been becoming a key task. Within the element belonging to user interface design process, the menu system can be considered as a component of fundamental importance for making UI attractive and usable, and special care is paid to their design and implementation.

A generative approach based on evolutionary algorithms has being investigated in order to assist the design of user interfaces in different aspects, e.g., see Hardman et al. (2009), Humayoun et al. (2014), du Plessis and Barnard (2008), Masson et al. (2011), Quiroz et al. (2007), Russo et al. (2008), Singh and Bhattacharya (2010) and Troiano et al. (2008). The reason of growing interest to this approach relies on different factors. In the first place, the design of UI requires to deal with several tradeoffs between conflicting requirements. In designing a menu layout of good quality, engineers have to consider many aspects including how effectively functions are retrieved and activated, what standard guidelines suggest, and what are the preferences of users. These aspects are translated into several design requirements, that often are conflicting. For instance, although having flat hierarchical structures improves accessibility, 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 tradeoff between these different requirements must be found in order to maximize the menu system quality. Evolutionary algorithms enable the search of optimum among a large number of alternatives, suggesting solutions that would never been considered by designers. In the second place, evolutionary algorithms allows to explore solutions in domains where there is lack of knowledge or too hard to model how to criteria in order to obtain an optimal solution, and this is the case of UI design. Finally, the ability of exploring and exploiting local optima in evolutionary algorithms makes possible the search over very irregular landscapes shaped over multidimensional search spaces.

In Birtolo et al. (2010) and Troiano and Birtolo (2014), we proposed a preliminary solution based on genetic algorithms. This paper goes further and provides a more extensive comparison between different evolutionary techniques. In particular, we investigated the application of Linear Genetic Programming, see Brameier and Banzhaf (2001) and Brameier and Banzhaf (2010), as a natural means to build optimal menu layouts by a generative approach. We performed an experimental comparison on both quantitative and qualitative analysis. The remainder of this paper is organized as follows: Sect. 2 overviews the problem of designing and optimizing a menu system; Sect. 3 provides some basics of Intelligent User Interface Design, with a focus on menu system generation; Sect. 4 describes the proposed evolutionary solutions investigated in our work; Sect. 5 reports quantitative experimental results; while Sect. 6 experiments menu design from scratch and as evolution; Sect. 7 outlines conclusions and future directions.

2 Designing and optimizing a menu system

This section provides a common understanding of the domain, in order to clarify terms and to disambiguate definitions. In particular, a *menu layout* represents the hierarchical structure providing access to the different application functions. So that, a menu layout is a tree made of menus, each of them is made of a list of *items* referring to *submenus* or to *actions*. While the first are menus at lower level, the latter activate functions, so they represent the leaves of the tree (i.e., terminals).

There are three main aspects to take into account when we design a menu leayout: (i) *accessibility*, as the ease of reaching desired actions, (ii) *guidelines*, as a set of best practices in organizing the menu layout, and (iii) *preferences*, as a wish list made explicit or implicit by the end user.

Among them, accessibility represent the key quality to optimize, so we assume it as optimization goal, while we leave guidelines and preferences as hard/soft constraints to drive the searching. When not made explicit, we will use the term constraint only for mandatory requirements, thus assuming them as hard. Instead, we will keep the term (optimization) preferences for soft constraints. In summary, the problem refers to find a menu layout that maximizes accessibility and compliance to guidelines and user preferences.

A main challenge in designing an effective menu system regards the way menu items are organized in the hierarchy, since from this aspect it depends accessibility to application functions. Menu selection involves many cognitive issues, since the user is first demanded to visually inspect the menu, by reading and understanding each item in order to find a path that will lead to the desired action. Thus, how menu items are organized affects the way task is accomplished.

Preliminarily, research investigated which functional features a menu system should have in order to improve accessibility. As an example, Walker and Smelcer Walker and Smelcer (1990) investigated the relationship between the structure made of walking menus and cascading menus against the time required by an user to reach the target action. We assume this issue is solved by the current standard implementations, so our focus is mostly on the relationship between the menu hierarchical structure (layout) and accessibility.

Several models have been proposed as predictors of the selection time as function of the menu organization, in order to find the structure able to minimize the selection time. If items are sorted, e.g. alphabetically, search time can be predicted by Hick's Law (see Hick 1952), 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, described in Fitts (1954) and Cockburn et al. (2007), 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 in Accot and Zhai (1997). 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 (2006) 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, in Bernard (2002) a further model for predicting the selection time has been presented. The Hypertext Accessibility Index measure (H_{HAI}) is defined as

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

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 $d = 1$

It can be easily verified that $H_{HAI} \in [1, +\infty)$: the lower H_{HAI} is, the more 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. (1992) 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 (Ahlström 2005). Ahlström et al. (2006) 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.

The drawback of adaptation and customization techniques, such as split menus, is the disruption of the original structure of menus and it is not the optimal solution for expert users, who have memorized the menu structure. The original design of split menus assumed that selection frequencies are a priori known and remain constant. In real environments, however, selection patterns may vary among users and change as user interaction and experience evolve over time. Adaptive menus in Microsoft Office made use of evolving selection patterns, but their success has been questioned (McGrenere et al. 2002; Findlater and McGrenere 2004). Tsandilas and Schraefel (2007) introduce bubbling menus, a design for cascading drop-down menus to facilitate the access of certain items in a menu, such as frequently selected items. The advantage of this technique is the fact that their application does not affect the original structure of menus.

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 Apple Computer Inc. (2006) and Sun's Java Look and Feel Guidelines Inc.

(2001). Guidelines are either too specific or too vague, so they do not always apply to the problem at hand Oliver et al. (2002). 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.

Finally, user preferences represent a reference to follow. The layout should reflect the way user conceptualizes and memorizes the functions of a program, and explicitly uses the organization and structure of the layout to benefit the interface (Norman 1991).

According to the current literature, building of menu hierarchy and optimization is a challenging task, whose applications go further the desktop and web applications. For instance Amant et al. (2007) discuss techniques for evaluating and improving cell phone usability, in particular the usability of the hierarchical menus, while Hollink et al. (2007) address the optimization of menus with a purely navigational function and define the optimal menu as the one that minimizes the average time users need to reach their target pages. People are not always good at building hierarchies. One hierarchy building task most users engage in is that of organizing their menu items. 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 support in screening alternatives and in providing starting point that can be refined more efficiently.

To sum up, we can state that 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.

3 Generative design for HCI

Designing User Interfaces (UI) is generally considered a creative and human intensive task, preventing from adopting computer aiding tools in exploring alternative solutions. The process leading to the ideation of user interfaces can be long running, time and cost consuming, entailing many decisions and iterative in nature.

Designers usually use guidelines to organize the layout and the features of user interface. Existing guidelines, such as Apple's Human Interface Guidelines and Sun's Java Look and Feel Guidelines are either too specific or too vague, so they do not always apply to the problem at hand. Thus UI designers tend to be guided both by objective measures gleaned from UI style guidelines and design principles, and by subjective measures such as the "look" and "feel" of an interface.

Therefore, designing an interface entails a number of decision problems with respects to the structure, attributes and logic. For instance, what is the widgets layout, how to split the user interaction among different frames, choosing the color palette, are common issues to be addressed during the interface design. As they can be basically reduced to choosing the most appropriate solution among different alternative, each presented as a combination of simpler alternatives, these issues can be regarded as optimization problems aimed at maximizing some utility function. This perspective makes possible to build a bridge between interface design and search algorithms, in order to adopt a generative approach in designing and building user interfaces. This approach takes several positive aspects, among them:

- A larger number of alternatives can be explored, often resulting in surprising solutions, thus supporting proactively human creativity and decision making;
- Different quality attributes and guidelines can be considered at a time (by means of a suitable utility function), thus facilitating the trade-off among conflicting criteria;
- Designers are made free to focus on more adding-value tasks, leaving algorithms to finely optimize their choices;
- Interfaces can be automatically adapted to a larger set of devices, and a more specific set of user preferences.

4 Searching a menu layout with Evolutionary Algorithms

The algorithm is inspired to the Simple GA given by Goldberg Goldberg (1989). The structure is outlined in Fig. 1.

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.

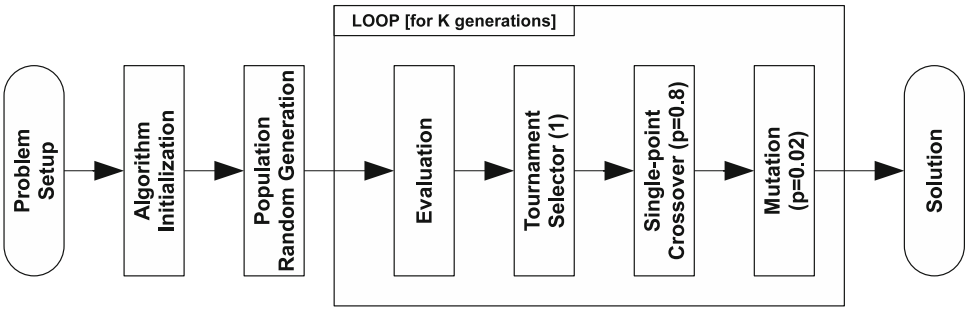


Fig. 1 Algorithm structure

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.

The problem of finding an optimal menu layout consists in placing all action items by maximizing accessibility 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

$$c_i(x) = 1 - \frac{v_i(x)}{mv_i(x)} \tag{2}$$

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

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

$$fitness(x) = \sigma \cdot H(x) + (1-\sigma) \cdot C(x) \tag{3}$$

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

$$H(x) = e^{k(1-H_{HAI}(x))} \tag{4}$$

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

$$C(x) = \frac{\sum_{i=1}^{m} \bar{p}_i c_i(x)}{\sum_{i=1}^{m} \bar{p}_i} \tag{5}$$

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.

4.1 Path evolution by genetic algorithm

Among the different ways of representing a tree structure by a chromosome, we chose 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 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:

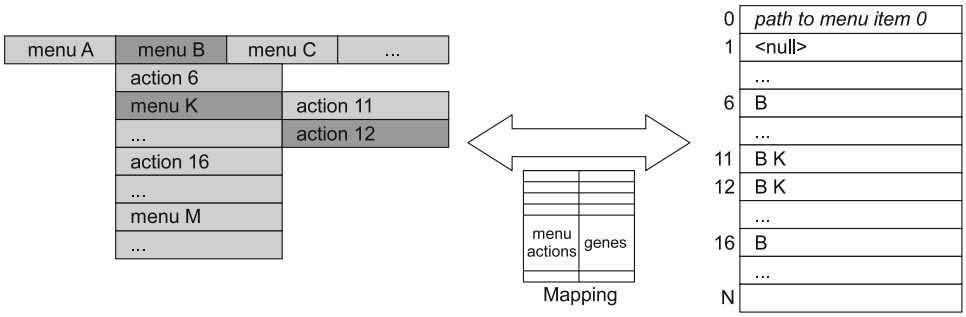


Fig. 2 Chromosome with mapping to the menu layout

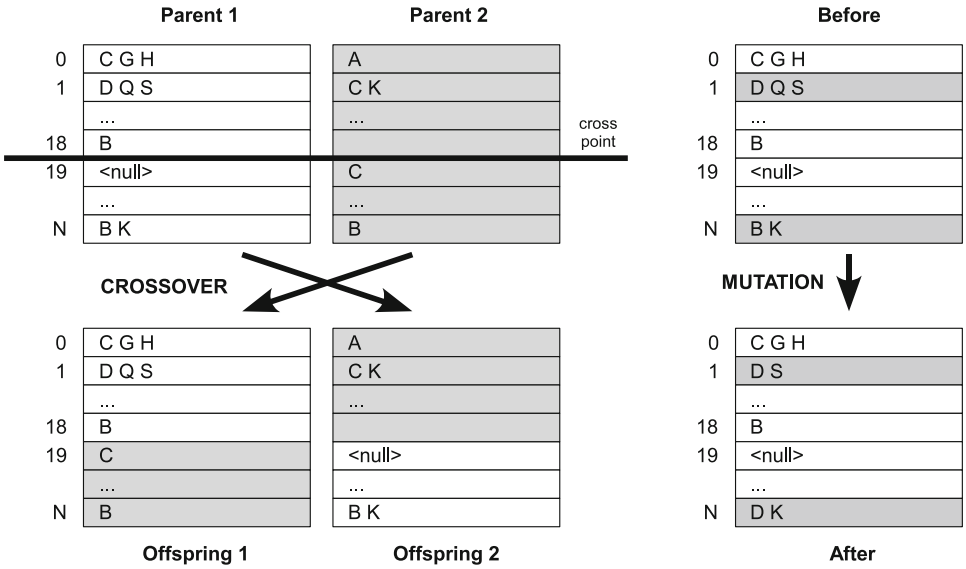


Fig. 3 Crossover and mutation

- 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 Goldberg (1989). Crossover and mutation are described in Fig. 3.

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

4.2 Structure evolution by genetic programming

The algorithm implemented a breeding sequence typical of evolutionary algorithms, that can be outlined as follows:

Algorithm 1 Algorithm structure

```
Generate and evaluate a random population
repeat
    Select individuals for mating
    Cross selected individuals
    Mutate selected individuals
    Apply elitism
    Evaluate generated individuals
until generation limit is reached
```

Initial population is built using a procedure able to meet as much as possible the given constraints and preferences, thus assuring valid and highly fitted individuals since beginning. For selection, we adopted a tournament operator in order to reduce the effect of fitness scaling, since each

individual is evaluated according to a fitness function defined on logical basis, as described below.

Crossover consists in visiting the nodes in the common region and deciding at each locus whether the corresponding offspring node should be picked from the first or the second parent. Mutation randomly replaces the instruction identifier, a variable, or the constant (if existent) by equivalents from valid ranges. Elitism replaced random individuals with best individuals in order to improve performances, as this strategy does not require to sort the population before being applied. The algorithm was setup with standard parameters[1] as follows: Crossover 0.8, Mutation 0.02, Elitism 3.

In GP menu hierarchies can be represented directly by tree based chromosomes. Therefore each individuals represent one possible menu hierarchy. The fitness function of an individual x is aimed at modeling the trade-off between accessibility and preference compliance. Thus it is defined as convex combination

$$fitness(x) = \sigma \cdot H(x) + (1 - \sigma) \cdot C(x) \tag{6}$$

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

$$H(x) = e^{k(1-H_{HAI}(x))} \tag{7}$$

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

$$C(x) = \frac{\sum_{i=1}^{m} \bar{p}_i c_i(x)}{\sum_{i=1}^{m} \bar{p}_i} \tag{8}$$

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.

4.3 Relation evolution by Linear Genetic Programming

In this case we represent the relation between menu items by a function

$$r : M \cup Ac \rightarrow M \times W \tag{9}$$

where M is the set of menu items, A the set of actions and W the vector of weights. Thus, $r(i)$ provides the pair (m, w), where m is the menu item to which i belongs, and w is a weighting vector so that given $i', i'' \in M \cup Ac$ such that $r(i') = (m, w')$ and $r(i'') = (m, w'')$, i' will be listed in m before i'' if $w' < w''$. Where $w' = w''$, we assume the natural ordering, e.g., lexicographic, imposed to $M \cup Ac$.

Therefore, the building of menu is led by a sequence of such operations. Linear Genetic Programming (LGP) has been introduced in Brameier and Banzhaf (2001) and Brameier and Banzhaf (2010) as means of evolving programs represented by sequences of instructions. Different problems have been faced by LGP. Among them, we have successful applications to financial trading in Wilson et al. (2011) and to maximization problem in Fagan et al. (2011) . Solutions aimed to improve the performances of LGP have been proposed in Downey et al. (2010) and Hu and Banzhaf (2009).

In our application, programs are sequences made of the same operation applied to different operands. We can have the following cases to consider: (i) an operation could be repeated on the same node i several times in the program; (ii) the sequence of operations could lead to cycles, so that constraint of obtaining a tree layout would be violated. For the first point, we note that in a building process led by randomly generated programs in which each operation overwrites the previous result, this is equivalent to pick up one operation at random and remove the others. Therefore, the genetic encoding can make an implicit use of operation referred to operands expressed at gene level. In more details, each gene is referred to a specific menu item i and its value is the pair (m, w) obtained by the application of function r to i.

With respect to the second point, we can avoid cycles by restricting the set of nodes and assigning an item i only to nodes m already assigned by the program. This ensures that the menu layout is built step by step from the root. In addition, constraints can be considered along the building process by further restricting the set of nodes m.

Expected advantages from this encoding are:

- The menu building program offers a linear representation of the tree layout.
- The tree layout and constraints are respected along the building process.

The process outlined so far is able to build valid solutions from scratch. In breeding new solutions from existing ones we need to pay attention to crossover and mutation, as they might produce inconsistencies with respect to the layout and constraints. Such inconsistencies can be limited by encoding the sequence of steps by triplets (i, m, w) such that $r(i) = (m, w)$. In this case, crossover between assignments will have place only if m is already assigned and compatible with constraints when assumed in the receiving sequence, otherwise the crossover will not have place. Similarly, mutation will have place by selecting i m and w appropriately according to the tree layout and constraints.

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

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

	A	B	C	D	E	F	G	H	I	J	K	L	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
A	—									3		3	2				3					4																
B		—						3		2														2				3				4					
C			—		2						3									3						4											
D				—		2				4		2						2					3										4				
E					—				3																			3	3	3							
F						—	2				4																										
G							—																												3	3	3
H								—						2	2	2																				
I									—										2																	
J										—											2															
K											—														2											
L												—															3									

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

5 Experimental results

In order to validate the proposed approach, we designed experimentation addressing two main aspects. The former is to prove the feasibility of the proposed approach in term of convergence and support in menu system design-process. The latter is to understand if it is possible to provide in a shorter time or in a easier way solutions comparable or better than those made by humans.

5.1 Genetic algorithm: convergence analysis

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 Fig. 4.

An example of layout compatible with this set of preferences is given by Fig. 5.

The algorithm was setup with standard parameters according to Table 1.[2]

Figures 6 and 7 show the layout of the best individual respectively after 10 and 1000 generations.

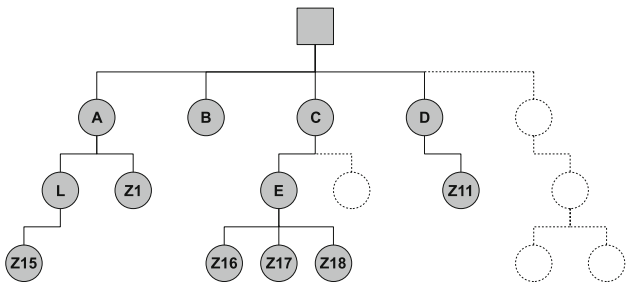


Fig. 5 An example of layout compatible with the preference set

Table 1 Algorithm's parameters

Tournaments	1
Crossover probability	0.8
Mutation probability	0.02
Elitism	2

Figure 6 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

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

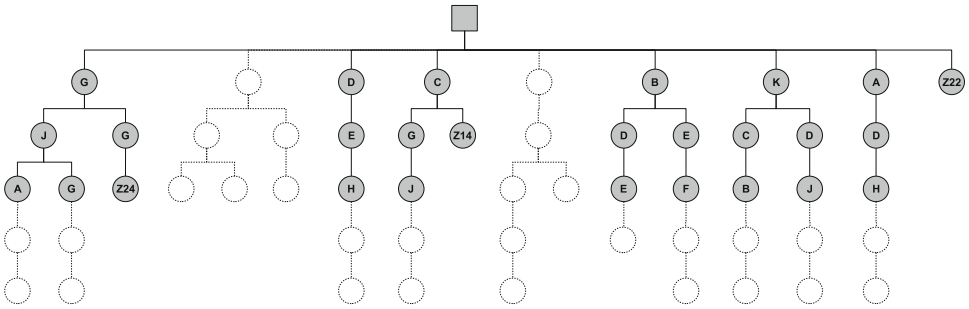


Fig. 6 Layout of the best individual after 10 generations (fitness 0.7949)

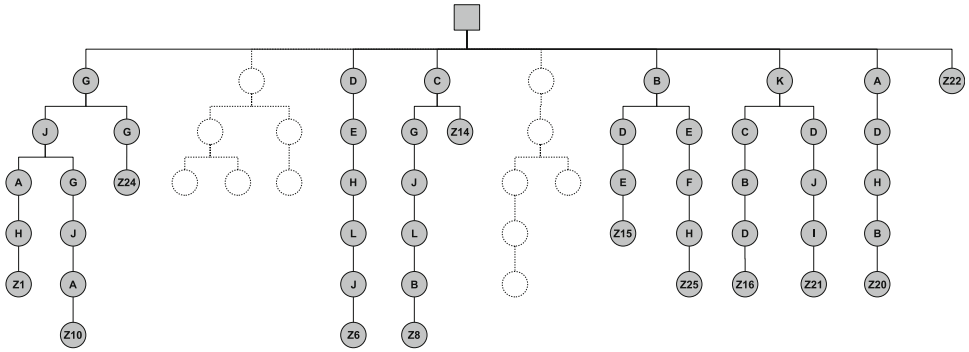


Fig. 7 Layout of the best individual after 1000 generations (fitness 0.9717)

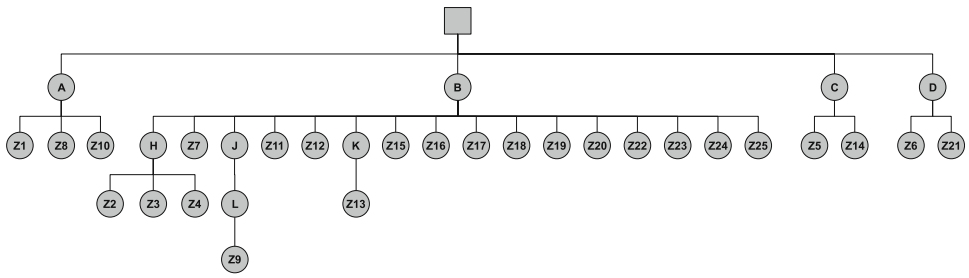


Fig. 8 Layout with constraints (fitness = 0.566)

in Fig. 7 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 Fig. 8 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 Fig. 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 Fig. 9.

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 *Z*1 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 *Z*10 is in *A*, action *Z*21 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 Fig. 9 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 Fig. 10, we report the median of best fitness with a different cardinality of the

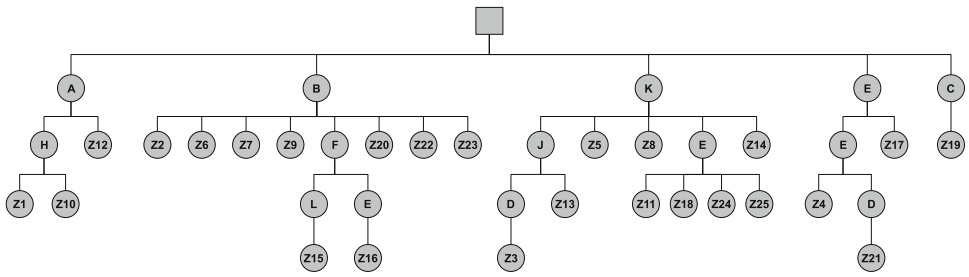


Fig. 9 Layout with compatible constraint and preference sets (fitness = 0.9952)

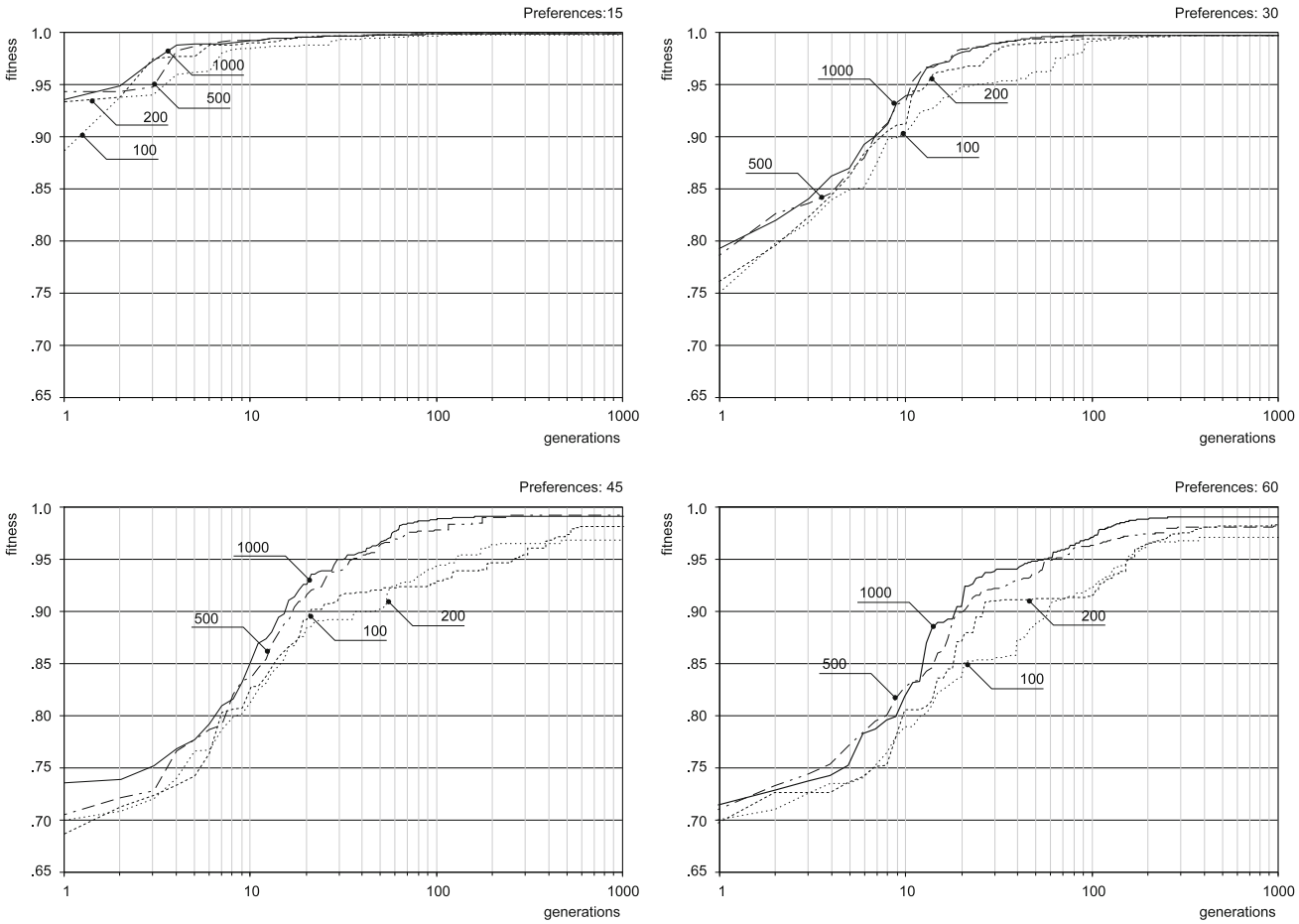


Fig. 10 GA: algorithm convergence

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 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 Fig. 11 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

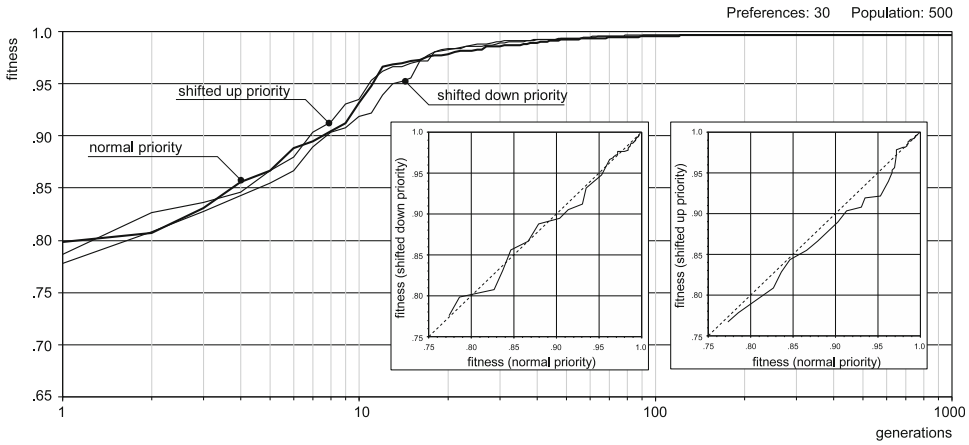


Fig. 11 Algorithm convergence with priority shift

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.

5.2 GP experimentation

As an example of application let us compose a menu layout made of 25 action items (terminals) and 12 submenus (non terminals). Experimentation was aimed at searching an optimal solution able to satisfy a set of 60 preferences with a different priority and to score a good value of accessibility index defined in Eq. (1).

We run the algorithm 5 times with different population size (100, 200, 500 and 1000 individuals) in order to limit the effect of randomness in studying and comparing convergence. From Fig. 12 we can outline some preliminary conclusions: (i) GP starts with better fitted individuals; (ii) GP convergence towards optimal solutions is faster in GP; (iii) population size is less relevant in GP than in GA. Fig. 13 presents a possible solution.

5.3 Linear genetic programming

When problem increase its complexity neither GA and GP performs as reported in Figs. 16 and 17 respectively.

Linear Genetic Programming is introduced to better provide a solution in a real problem, where the number of constraints is considerably.

In our experimentation we take into account the menu system available at http://www.unisannio.it and we consider a subset of 27 items and 9 menu items. In order to satisfy user needs and to design a real menu system, we explicit 90 constraints (i.e., 69 Item under menu Constraints, 6 Level constraints, 9 Repetition Constraints, 4 Number of Children Constraints and 2 Menu Ordering constraints) as reported in Figs. 14 and 15.

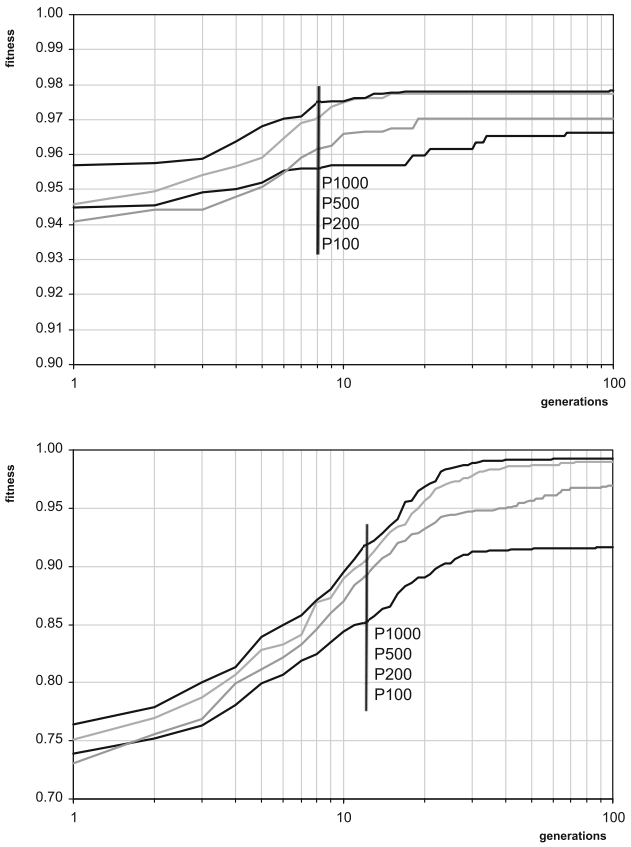


Fig. 12 GP (*top*) and GA (*bottom*) fitness evolution

We repeated 10 runs for different problem configurations. The average behavior of the algorithm is depicted in Figs. 16, 17 and 18.

GA is not able to provide a valid solution with all the 90 constraints and we consider only a subset of 81 constraints from the original ones, obtaining some valid solutions. In details, in Fig. 16 is plotted the mean of best individual fitness at different population size of 100, 500, 1000, 2000 individuals when GA is considered.

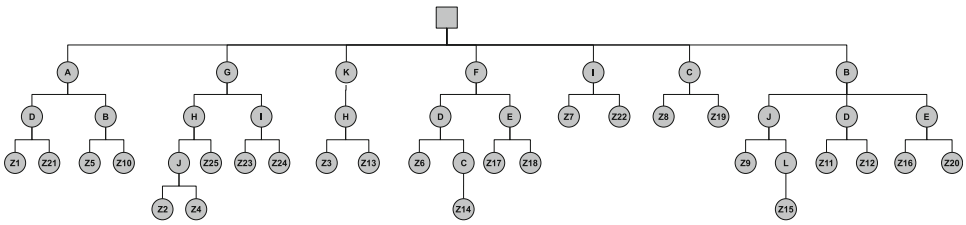


Fig. 13 A possible solution given the set of preferences

Fig. 14 Building Unisannio menu system: item under Menu Constraints

Item e Menu	Post Laurea	Didattica	Ricerca	Ateneo	Bacheca	Utilità	Servizi	News	Internazionale
Postlaurea									
Dottorati	1	1	1						
Assegni ricerca	1		1						
Esami di stato	1								
Master	1	1					3	3	
Didattica									
Orientamento		1			1	1			
Offerta formativa		1			1	1			
Segreteria Studenti		1		3			3		
Centro Linguistico		1		1					
Ricerca									
Ufficio ricerca			1						
Ufficio TTO			1						
Relazione ricerca			1						
Ateneo									
Organi di governo				1					
Statuto		2	2	1					
Biblioteche				1		2	2		
Bacheca									
Avvisi Ateneo				2	1			2	
Avvisi Studenti			3	2	1			3	
Concorsi e Bandi				1	1	2	1		
Utilità							2		
Immatricolazioni A.A. 2010/2011		2		3		2			
Rubrica Telefonica					2	1	2		
Servizi online					2				
Servizi Docenti							1		
Servizi Studenti							1		
News					2				
Rassegna Stampa								1	
Convegni e Seminari	3				2			1	
Comunicati Ateneo				3				1	
Eventi Ateneo				3				1	
Internazionale							3		
Erasmus					2		1	2	1
Accordi Internazionali					2			2	1

Instead, in Fig. 17 is plotted the mean of best individual fitness when GP is considered as a solver. GP is not always able to provide a solution because it attempts to build a valid solution within a given number of trials (i.e., 100 trials) per generation and a valid one is presented only in the 30 % of runs. Moreover, some convergence issues arise.

Fig. 15 Building Unisannio menu system: other constraints

	Min - Max Level Allowed	Priority
Root	4-5	1
Level 1	2-4	2
Level 2	2-5	2

Ordering	Priority
Ateneo-Didattica-Ricerca	1
Servizi-News	2

Menus	Min - Max Level Allowed	Priority	Min - Max Repetition Allowed	Priority
Postlaurea	0-2	3	1-2	3
Didattica	1-1	3	1-1	3
Ricerca	1-1	3	1-1	3
Ateneo	1-2	2	1-1	3
Bacheca	1-3	1	0-1	3
Utilità	-	-	0-1	2
Servizi online	-	-	0-1	2
News	2-2	2	0-1	2
Internazionale	-	-	0-1	2

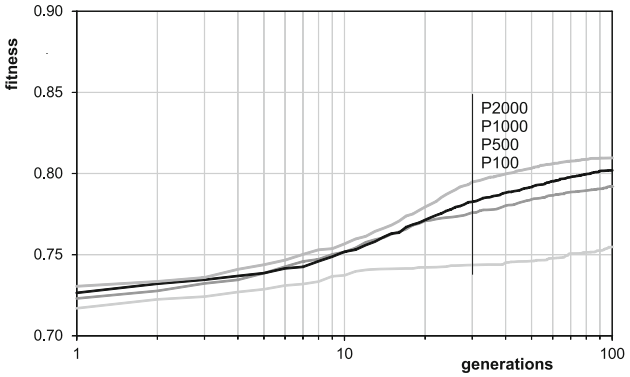


Fig. 16 GA: mean (10 runs) of best individual fitness at varying population size (a subset of 81 constraints)

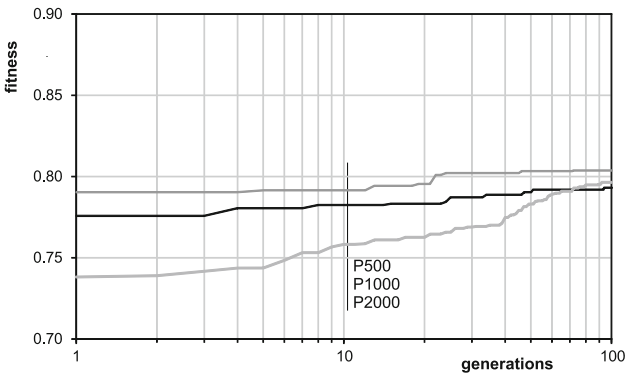


Fig. 17 GP: mean (10 runs) of best individual fitness at varying population size

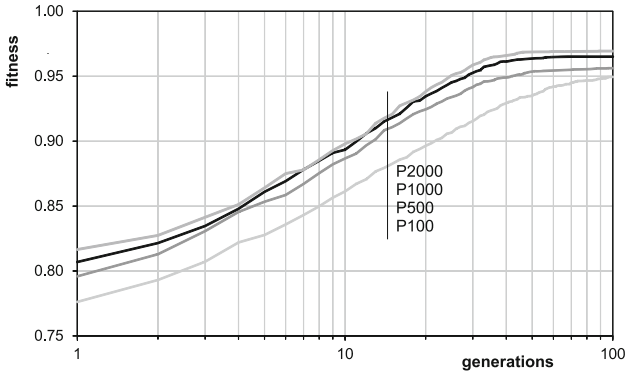


Fig. 18 LGP: mean (10 runs) of best individual fitness at varying population size

Finally, LGP, as depicted in Fig. 18 is able to converge towards a valid solution.

5.3.1 Mutation and convergence analysis

Mutation is widely used in GP approaches, even if its performances depends on both the problem and the details of the GP systems. Indeed, even if Koza suggested to use a low level of mutation in GP, Brameier and Banzhaf (2001) and Brameier and Banzhaf (2010) showed benefits in high mutation rate in LGP and justified it stating that "exchanging a variable can have an enormous effect on the program flow".

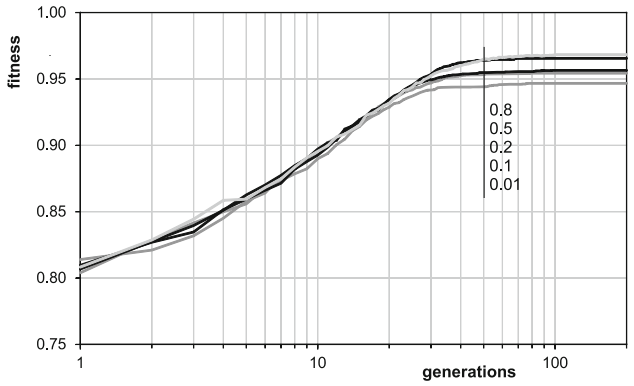


Fig. 19 LGP: mean (10 runs) of best individual fitness at varying mutation rate

Table 2 Wilcoxon paired test: average best fitness $\overline{f_b}$ at generation 100 and p-values

Mutation	$\overline{f_b}$	p-value	
		m_4	m_5
$m_1 = 0.01$	0.9467	3.264e−04	2.435e−04
$m_2 = 0.1$	0.9543	5.745e−04	2.422e−04
$m_3 = 0.2$	0.9564	6.441e−04	7.615e−04
$m_4 = 0.5$	0.9655	–	1.399e−01
$m_5 = 0.8$	0.9682	1.399e−01	–

In our experimentation, we compare algorithm convergence at varying the mutation rate from 0.01 to 0.8, as depicted in Fig. 19. We confirm best results with high value of mutation rate and we set the mutation rate equal to 0.5 in according with a deeper investigation of results reported in Table 2. Indeed, no statistical evidence of benefits if we increase this value.

6 Experimentation with user panel: building a menu from scratch and as evolution

Sections below will provide experimentation details and results.

6.1 Participants

For the experimentation we enrolled 34 participants within the students of Computer Science Engineering of University of Sannio and the CiseLab team. We split the participant in two groups: the former consists of 14 designer, the latter includes 20 usability tester.

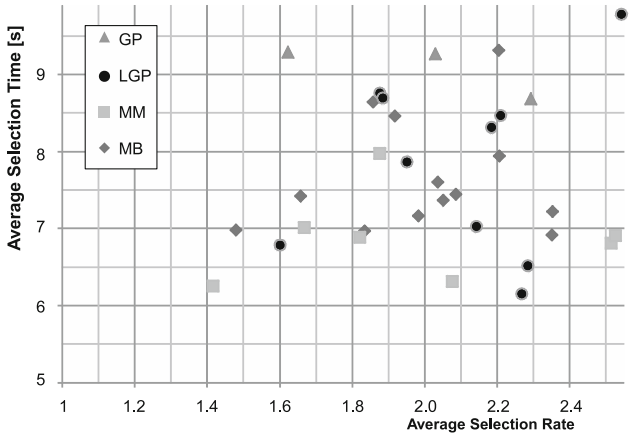


Fig. 21 Comparison of GP, LGP and manual solutions by means of selection rate and selection time

6.2 Equipment and materials

In order to guarantee adequate and same conditions for performing the test to all participants, we arranged a one-pc room in CiseLab Laboratory in conformance to the ISO 9241 standard. The experimental observations were carried out on Intel Pentium IV machine with 2 GB of RAM running Windows XP Professional Edition SP2 equipped with BenQ T720 LCD monitor, standard keyboard and optical mouse.

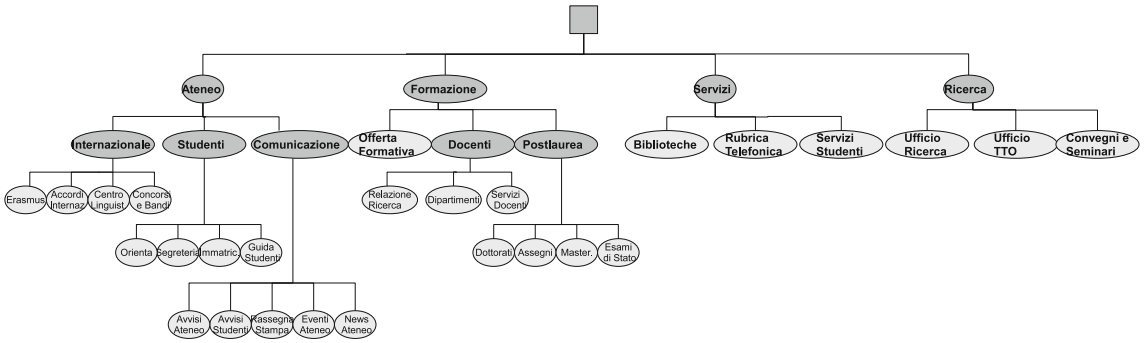
Experimentation was supported by a tool able to measure the value of accessibility index defined in Eq. (1) and to show the fitness of designed menu system. Moreover, in a proper section the requirements, i.e., the constraints and preferences to satisfy, are shown.

6.3 Procedure

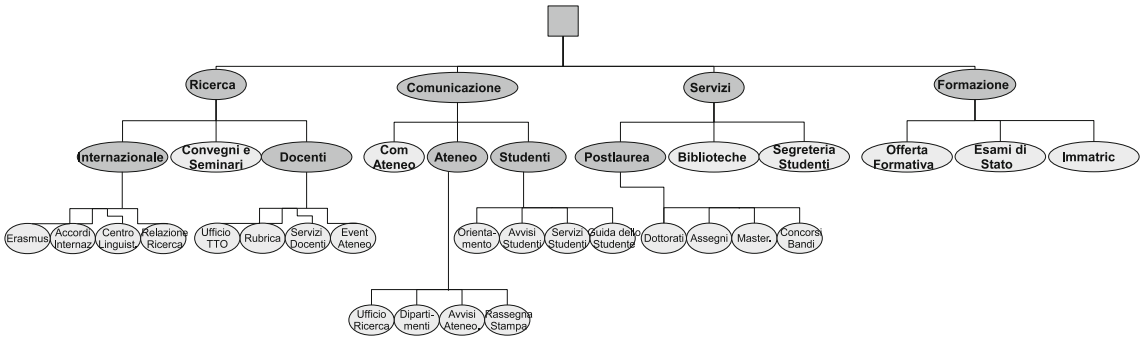
In this section we describe the procedures. In particular we define two steps of our experiment. The first step is aimed at building a menu from scratch according a set of constraints, while the second one is aimed at validating different menus in order to assess the easiness in selecting the items and navigating the menu system in term of number of clicks and time-to-completion the task.

6.3.1 Procedure of design step

The task assigned to the first group is to build a menu from scratch. The designers have to provide a solution for the Unisannio menu system satisfying the set of 90 constraints reported in Figs. 14 and 15. The maximum allowed time is an hour.



(a) Best manual solution provided by the first user group



(b) Best LGP solution after 100 generations

Fig. 20 Qualitative comparison of Best solutions provided by human and by LGP

Table 3 Average (200 trials) selection time and p-values

Menu system	Average selection time	p-value			
		GP	LGP	MM	MB
GP	13.134	–	2.00e−06	4.78e−07	4.55e−09
LGP	9.197	2.00e−06	–	6.58e−01	3.21e−01
MM	9.054	4.78e−07	6.58e−01	–	6.16e−01
MB	8.281	4.55e−09	3.21e−01	6.16e−01	–

6.3.2 Procedure of testing step

The task implemented consisted in select a menu item within the proposed menu system, in order to focus the test on effects of menu layout in recognizing the action.

In this step, selection rate and time-to-task is recorded. Selection rate is defined as the ratio between the effective number of clicks and the expected one, while time-to-task measures the elapsed time to detect a given word.

Wrong selection is not allowed because the test stops only if the right menu item is selected (a threshold time is defined). Test is iterated along 10 menu items per user and per menu system solution. In order to reduce user fatigue a time of 5 minutes was given between the two solutions being tested.

The menu items are randomly proposed in order to avoid a results strictly influenced by the familiarity of a set of items and in order to reduce the recognition of the same items in the different solutions. The menu system solution selected for experimentation are 4: (i) the best GP solution (the fitness is equal to 0.821), (ii) the best LGP legal solution (the fitness is equal to 0.963) among 500 individuals after 100 generations (depicted in Fig. 20b), (iii) the best solution provided by humans shown in Fig. 20a whose fitness is equal to 0.967 (named MB), and (iv) the median one provided by 14 designers (the fitness is equal to 0.919), named MM.

6.4 Results

The best solutions provided by human and by LGP are depicted in Fig. 20a, b respectively.

Analyzing the results of the second group, it is possible to highlight how shorter selection time occurs when LGP and MB solutions are adopted (Fig. 21).

Table 4 Average (200 trials) selection rate and p-values

Menu system	Average selection rate	p-value			
		GP	LGP	MM	MB
GP	2.834	–	4.279e−02	2.228e−02	1.858e−02
LGP	2.383	4.279e−02	–	7.667e−01	8.025e−01
MM	2.644	2.228e−02	7.667e−01	–	6.789e−01
MB	2.324	1.858e−02	8.025e−01	6.789e−01	–

Table 5 Average and standard deviation (10 trials of the user 1) of selection time and p-values

Menu system	Selection time		p-value	
	Average	Std. dev	GP	LGP
GP	14.653	7.549	–	3.55e−02
LGP	7.870	3.325	3.55e−02	–
MM	6.318	2.316	1.26e−02	2.57e−01
MB	8.281	4.002	2.88e−02	5.29e−01

Table 6 Average and standard deviation (10 trials of the user 1) of selection rate and p-values

Menu system	Selection rate		p-value	
	Average	Std. dev	GP	LGP
GP	3.158	2.101	–	2.692e−01
LGP	1.950	0.923	2.692e−01	–
MM	2.075	1.149	2.351e−02	9.694e−01
MB	1.478	0.419	3.288e−02	4.431e−01

Investigating these results more deeply, we consider a two-sided Wilcoxon rank-sum test, reporting results in Table 3, where the average value of Selection Time of 10 different trials per technique and user (i.e., 200 trials per technique) is shown and in Table 4, where Selection rate is investigated. The null hypothesis are: (i) the investigated techniques belongs to the same population entailing a comparable Selection Time for a target menu item, and (ii) the investigated techniques require the same Selection Rate for a target menu item Assuming 0.05 as upper limit to reject the null hypothesis, we can affirm that there is statistical difference between GP and other approaches. We prove that LGP outperforms GP, but we cannot reject the null hypothesis when LGP and MB is compared. Tables 5 and 6 report results obtained by a single user (user 1).

Therefore we can state that LGP outperforms GP in terms of lower selection rate and lower time-to-task, but in order to exploit the comparison between LGP and manual approach we present an in-depth analysis per user, as shown in Table 7.

Table 7 Average (10 trials per user) selection Rate and p-values

User	Selection rate			p-value	
	LGP	MM	MB	LGP vs. MM	LGP vs. MB
1	1.950	2.075	1.478	9.694e−01	4.431e−01
2	2.450	2.883	1.917	9.368e−01	6.992e−01
3	3.042	3.658	2.733	6.200e−01	4.016e−01
4	2.783	2.712	2.035	1	1.397e−01
5	2.575	1.875	3.043	8.765e−02	5.697e−01
6	1.875	2.410	1.833	1.372e−01	9.052e−01
7	2.208	2.513	2.203	4.343e−01	9.394e−01
8	2.250	3.240	3.345	2.857e−01	3.191e−01
9	2.283	1.415	1.982	9.678e−01	4.624e−01
10	2.917	3.042	2.350	9.696e−01	6.148e−01
11	2.058	2.160	1.857	3.536e−01	6.195e−01
12	2.183	3.965	2.352	3.692e−01	7.867e−01
13	3.833	2.670	2.743	2.845e−01	5.857e−01
14	2.267	3.180	2.085	9.687e−01	9.697e−01
15	2.275	2.868	2.372	5.164e−01	9.090e−01
16	1.883	1.820	2.050	5.005e−01	6.990e−01
17	1.600	2.962	1.657	2.915e−01	6.957e−01
18	2.983	2.525	2.205	5.879e−01	8.477e−01
19	2.142	1.667	2.812	3.893e−01	4.003e−01
20	2.542	3.245	3.433	6.451e−01	6.454e−01

7 Conclusion

In this paper we presented a generative approach based on LGP for optimizing the layout of a GUI menu system. We kept into the account accessibility, user preferences and standard guidelines. Experimental results provide very encouraging outcomes, proving the ability of LGP in converging towards solutions with high fitness, also in a real case study. Moreover, the algorithm has been proven to give solutions comparable with those provided by humans. Although human solutions can introduce the semantics which is not specified in the constraints and generative approach is obviously not able, the comparison of the different solution in terms of usability of the menu system and in terms of the ease of identification a target menu items, shows similar results. To sum up, linear

genetic programming has been proved to be a valid approach able to assist the designer in a design-process of menu systems. On the one hand, the proposed solution can reduce human fatigue in the design step, and on the other hand guarantee high quality solutions. More in general, the design of UI elements can benefit of different techniques developed by Computational Intelligence. Other approaches in recommendation systems (Troiano et al. 2014) and data mining (Troiano and Scibelli 2014a, b) offer interesting directions to develop new assisting techniques for UI design.

References

Accot J, Zhai S (1997) Beyond fitts' law: models for trajectory-based hci tasks. In: CHI '97: Proceedings of the SIGCHI conference on Human factors in computing systems, ACM, New York, NY, USA, pp 295–302. doi:10.1145/258549.258760

Ahlström D (2005) Modeling and improving selection in cascading pull-down menus using fitts' law, the steering law and force fields. In: CHI '05: Proceedings of the SIGCHI conference on Human factors in computing systems, ACM, New York, NY, USA, pp 61–70. doi:10.1145/1054972.1054982

Ahlström D, Alexandrowicz R, Hitz M (2006) Improving menu interaction: a comparison of standard, force enhanced and jumping menus. In: CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems, ACM, New York, NY, USA, pp 1067–1076. doi:10.1145/1124772.1124932

Amant RS, Horton TE, Ritter FE (2007) Model-based evaluation of expert cell phone menu interaction. ACM Trans Comput-Hum Interact 14(1):1. doi:10.1145/1229855.1229856

Apple Computer Inc (2006) Apple human interface guidelines. In: Tech. rep., Apple Computer Inc

Bernard ML (2002) Examining a metric for predicting the accessibility of information within hypertext structures. Ph.D. thesis, Wichita, KS, USA, adviser-Charles G. Halcomb

Birtolo C, Armenise R, Troiano L (2010) Supporting menu layout design by genetic programming. In: Filipe J, Cordeiro J (eds) ICEIS 2010–proceedings of the 12th international conference on enterprise information systems, HCI, Funchal, Madeira, Portugal, June 8–12, 2010, vol 5. SciTePress, pp 248–251

Botafogo RA, Rivlin E, Shneiderman B (1992) Structural analysis of hypertexts: identifying hierarchies and useful metrics. ACM Trans Inf Syst 10(2):142–180. doi:10.1145/146802.146826

Brameier M, Banzhaf W (2001) A comparison of linear genetic programming and neural networks in medical data mining. IEEE Trans Evolut Comput 5(1):17–26. doi:10.1109/4235.910462

Brameier MF, Banzhaf W (2010) Linear genetic programming, 1st edn. Springer, Berlin

Cockburn A, Gutwin C, Greenberg S (2007) A predictive model of menu performance. In: CHI '07: proceedings of the SIGCHI conference on Human factors in computing systems, ACM, New York, NY, USA, pp 627–636. doi:10.1145/1240624.1240723

Downey C, Zhang M, Browne WN (2010) New crossover operators in linear genetic programming for multiclass object classification. In: Proceedings of the 12th annual conference on Genetic and evolutionary computation, ACM, New York, NY, USA, GECCO '10, pp 885–892. doi:10.1145/1830483.1830644

du Plessis MC, Barnard L (2008) Incorporating layout managers into an evolutionary programming algorithm to design graphical user interfaces. In: Proceedings of the 2008 annual research cONFERENCE of the South African institute of computer scientists and information technologists on IT research in developing countries: riding the wave of technology, ACM, New York, NY, USA, SAICSIT '08, pp 41–47. doi:10.1145/1456659.1456665

Fagan D, Nicolau M, Hemberg E, O'Neill M, Brabazon A, McGarraghy S (2011) Investigation of the performance of different mapping orders for ge on the max problem. Proceedings of the 14th European conference on Genetic programming, EuroGP'11. Springer, Berlin, pp 286–297

Findlater L, McGrenere J (2004) A comparison of static, adaptive, and adaptable menus. In: CHI '04: Proceedings of the SIGCHI conference on Human factors in computing systems, ACM, New York, NY, USA, pp 89–96. doi:10.1145/985692.985704

Fitts PM (1954) The information capacity of the human motor system in controlling the amplitude of movement. J Exp Psychol 47(6):381–391. http://view.ncbi.nlm.nih.gov/pubmed/13174710

Goldberg DE (1989) Genetic algorithms in search, optimization and machine learning. Addison Wesley, Boston

Hardman N, Colombi J, Jacques D, Hill R, Miller J (2009) Application of a seeded hybrid genetic algorithm for user interface design. In: Proceedings of the 2009 IEEE international conference on systems, man and cybernetics, IEEE Press, Piscataway, NJ, USA, SMC'09, pp 462–467. http://dl.acm.org/citation.cfm?id=1732323.1732402

Hick WE (1952) On the rate of gain of information. Q J Exp Psychol 4:11–26

Hollink V, Someren M, Wielinga BJ (2007) Navigation behavior models for link structure optimization. User Model User-Adapt Interact 17(4):339–377. doi:10.1007/s11257-007-9030-0

Hollink V, van Someren M (2006) Validating navigation time prediction models for menu optimization. In: Althoff KD, Schaaf M (eds) LWA, University of Hildesheim, Institute of Computer Science, Hildesheimer Informatik-Berichte, vol 1/2006, pp 47–52

Hu T, Banzhaf W (2009) Neutrality and variability: two sides of evolvability in linear genetic programming. In: Proceedings of the 11th annual conference on Genetic and evolutionary computation, ACM, New York, NY, USA, GECCO '09, pp 963–970. doi:10.1145/1569901.1570033

Humayoun SR, AlTarawneh R, Ebert A, Dubinsky Y (2014) Automate the decision on best-suited ui design for mobile apps. In: Proceedings of the 1st international conference on mobile software engineering and systems, ACM, New York, NY, USA, MOBILESoft 2014, pp 66–68. doi:10.1145/2593902.2593919

Inc SM (2001) Java look and feel design guidelines: advanced topics. Addison-Wesley, Boston

Kong J, Zhang WY, Yu N, Xia XJ (2011) Design of human-centric adaptive multimodal interfaces. Int J Hum-Comput Stud 69(12):854–869

Masson D, Demeure A, Calvary G (2011) Examples galleries generated by interactive genetic algorithms. In: Procedings of the second conference on creativity and innovation in design, ACM, New York, NY, USA, DESIRE '11, pp 61–71. doi:10.1145/2079216.2079225

McGrenere J, Baecker RM, Booth KS (2002) An evaluation of a multiple interface design solution for bloated software. In: CHI '02: proceedings of the SIGCHI conference on Human factors in computing systems, ACM, New York, NY, USA, pp 164–170. doi:10.1145/503376.503406

Norman KL (1991) The psychology of menu selection: designing cognitive control at the human/computer interface. Greenwood Publishing Group Inc., Westport

Oliver A, Regragui O, MonmarchT N, Venturini G (2002) Genetic and interactive optimization of web sites. The 11th international World wide web conference, Honolulu, Hawaii, USA, pp 7–11

Quiroz JC, Louis SJ, Dascalu SM (2007) Interactive evolution of xul user interfaces. In: Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation, ACM, New York, NY, USA, GECCO '07, pp 2151–2158. doi:10.1145/1276958.1277373

Russo G, Birtolo C, Troiano L (2008) Generative ui design in sapi project. In: CHI '08 extended abstracts on human factors in computing systems, ACM, New York, NY, USA, CHI EA '08, pp 3627–3632. doi:10.1145/1358628.1358903

Singh N, Bhattacharya S (2010) A ga-based approach to improve web page aesthetics. In: Proceedings of the first international conference on intelligent interactive technologies and multimedia, ACM, New York, NY, USA, IITM '10, pp 29–32. doi:10.1145/1963564.1963569

Troiano L, Rodríguez-Muñiz L, Díaz I (2014) Discovering user preferences using dempster-shafer theory. Fuzzy Sets Syst. doi:10.1016/j.fss.2015.06.004

Troiano L, Birtolo C (2014) Genetic algorithms supporting generative design of user interfaces: examples. Inf Sci 259:433–451. doi:10.1016/j.ins.2012.01.006

Troiano L, Birtolo C, Miranda M (2008) Adapting palettes to color vision deficiencies by genetic algorithm. In: Proceedings of the 10th annual conference on genetic and evolutionary computation, ACM, New York, NY, USA, GECCO '08, pp 1065–1072. doi:10.1145/1389095.1389291

Troiano L, Scibelli G (2014a) Mining frequent itemsets in data streams within a time horizon. Data Knowl Eng 89:21–37. doi:10.1016/j.datak.2013.10.002

Troiano L, Scibelli G (2014b) A time-efficient breadth-first level-wise lattice-traversal algorithm to discover rare itemsets. Data Min Knowl Discov 28(3):773–807. doi:10.1007/s10618-013-0304-3

Tsandilas T, Schraefel MC (2007) Bubbling menus: a selective mechanism for accessing hierarchical drop-down menus. In: CHI '07: proceedings of the SIGCHI conference on Human factors in computing systems, ACM, New York, NY, USA, pp 1195–1204. doi:10.1145/1240624.1240806

Walker N, Smelcer JB (1990) 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

Wilson G, Leblanc D, Banzhaf W (2011) Stock trading using linear genetic programming with multiple time frames. In: Proceedings of the 13th annual conference on Genetic and evolutionary computation, ACM, New York, NY, USA, GECCO '11, pp 1667–1674. doi:10.1145/2001576.2001801