Evaluating the Effectiveness of "Effective View Navigation" for Very Long Ordered Lists on Mobile Devices

Luca Chittaro and Luca De Marco

HCI Lab, Dept. of Math and Computer Science, University of Udine,
via delle Scienze 206, 33100 Udine, Italy
{chittaro,demarco}@dimi.uniud.it

Abstract. Searching for an item in a long ordered list is a frequent task when using any kind of computing device (from desktop PCs to mobile phones). This paper explores three different interfaces to support this task on the limited screen of mobile devices (e.g., PDAs, in-car systems, mobile phones). Two of the considered interfaces are based on the idea of tree-augmentation of a list proposed in Furnas' *Effective View Navigation* theory [6] and differ in their depth versus breadth ratio. The third interface adopts the traditional technique of list scrolling based on keyboard entry. We compare them in terms of search time, number of errors, and user's satisfaction. Results show that list scrolling based on keyboard entry outperforms both tree-augmented lists and that the broader tree-augmented list is better than the deeper one.

1 Introduction

In recent years, mobile devices such as personal digital assistants, smart phones and in-car navigation systems have become more and more widespread among consumers. These devices offer the user the opportunity to access a great deal of information coming from local or remote databases, anytime and anywhere. In this context, the small screen of mobile devices is a serious limitation, because it restricts the user's ability to view and interact with large amounts of information. This information is often organized into long ordered lists, e.g., contact information in address books, alphanumeric index entries in databases or multimedia catalogs, destinations in navigation systems, etc. Various mechanisms have been proposed to facilitate list scrolling, e.g., buttons for moving up and down, thumbwheels on the side of devices, scrollbars on touch-sensitive screens. Besides scrolling mechanisms, many interfaces support list scrolling based on keyboard entry. The idea is to select list items by combining the use of a (virtual or physical) keyboard and a scrolling list: pressing any key will automatically scroll the list up to the first item whose first character matches the pressed key, and any subsequent keypress will further refine the search considering also the second character, then the third and so on. Although this mechanism is an obvious, widely adopted and efficient solution (assuming the user is familiar with the keyboard layout), very few mobile devices offer a complete physical

keyboard and many can require multiple keypresses to select a single character (e.g., mobile phones), while adopting a virtual keyboard has the disadvantage of wasting precious screen space.

In general, the literature proposes alternative techniques to search items in long lists that are especially valuable in contexts where screen space is at premium, e.g., Alphaslider [1], Popup vernier [2], Fish-eye views [5] and, in a broader context, Zooming interfaces such as PAD++ [4]. In particular, Furnas [6] discussed how to improve the efficiency of retrieving an item in a long ordered list from a theoretical point of view. Among the alternatives he proposed, fisheye sampling has been recently explored by Bederson [3], while tree-augmentation techniques have been partially explored in previous studies ([7] and [8]) with encouraging results that motivate further investigation.

In our study, we focused on the latter idea and implemented two interfaces for mobile devices, inspired to those proposed in [7] and [8]. We then implemented a more traditional interface employing list scrolling based on keyboard entry. We carried out an experiment to compare the three interfaces, in terms of both users' performance and satisfaction, with a twofold goal. First, we wanted to compare the tree-augmentation technique versus list scrolling based on keyboard entry. Second, we wanted to compare tree-augmented lists that differ in their breadth versus depth ratio (as in [7]) on the limited screen of a mobile device.

The paper is organized as follows. In Section 2, we introduce Furnas' *Effective View Navigation* theory [6]. Section 3 describes the tree-augmentation algorithm we adopted. In Section 4, we summarize related work on evaluating item search interfaces for long lists. Section 5 and 6 respectively present the interfaces we implemented and their evaluation. Experiment results are illustrated in Section 7, while Section 8 presents conclusions and future work.

2 Theoretical Foundations: Effective View Navigation [6]

In this section, we will briefly illustrate the core ideas of Furnas' [6] *Effective View Navigation*.

Effective View Navigation is a theory of navigation in information spaces that explores some basic issues in moving around and finding data in different information structures (e.g., webs, trees, tables, simple lists). The focus is particularly on issues that arise when such structures get very large, and interaction is seriously limited by available space (e.g., screen real estate) and time (e.g., number of actions required to get somewhere).

The theory (for a complete description of it, see [6]) defines the two requirements that an information structure must satisfy to be *Efficiently View Traversable* (EVT); a comparison of several EVT structures is then made by means of a formal worst case characterization and suggestions about how to fix non-EVT structures (in particular, ordered lists) by laying over them an EVT structure are given; finally, the two requirements for *View Navigability* are discussed.

The basic assumption of Effective View Navigation theory is that the user navigates an information structure to find a specific *target*. The information structure

can be represented by a *viewing graph*. At any given time, the user is *at* some *node* in the viewing graph. Each node represents a *view* that contains *items* (e.g., a window that shows a subset of items of a list). Users can make *selections* in a view to reach other nodes of the viewing graph (e.g., a click on a displayed item could change the set of displayed items).

A structure is defined to be *Efficiently View Traversable* (EVT) if the following two requirements are met:

- EVT1: the number of available selections at each view is small compared to the size of the structure.
- EVT2: the number of selections needed to travel from one view to any other is small compared to the size of the structure.

Moreover, a structure is defined to be *View Navigable* (VN) if the following two requirements are met:

- VN1: all views must contain enough information to make clear which is the correct selection to find the target.
- VN2: each available selection must make its outcome clear by presenting a small amount of information.

These last two requirements conflict with each other so that a View Navigable structure results only from a proper balance between them.

Finally, a structure is defined to be *Effectively View Navigable* if it is both Efficiently View Traversable and View Navigable.

Among the Efficiently View Traversable structures analyzed in [6] (by means of a worst case characterization), the balanced tree turns out to be one of the best with respect to EVT requirements and, for this reason, Furnas suggests that the efficiency of view-traversal of a long list could be improved by means of tree-augmentation. In the next section, we will see how tree-augmentation works.

3 The Adopted Tree-Augmentation Algorithm

In this section, we present the algorithm for tree-augmentation of ordered lists that we employed in two of the interfaces we studied.

The algorithm considers the n alphabetically ordered items of a list as the leaves of a tree, and generates a number (linear in n) of nodes to build a balanced tree of degree k and depth $d = \lceil \log_k n \rceil$. Obviously, if the number of items n in the list is not a power of k, not all the subtrees rooted at the same level of the tree will contain the same number of nodes (and in particular leaves) as we will see in the following.

We call $L(i,j)$ the number of leaves that are descendants of the i-th node at level j. We start by creating the root of the tree at level 0 and all n list items as its descendants (i.e., $L(1,0)=n$). We then repeat the following iterative process until we reach the leaves level. We consider level j (starting from level 0) and for each node i at that level we create k children. Then, for each node i at level j, we organize the $L(i,j)$ leaves that are descendants of that node as follows: each of the first $k - 1$ children of node i will have $\lceil L(i,j)/k \rceil$ leaves in its descendants and the remaining

leaves (i.e., $L(i,j)$-$(k$-$1)\lceil L(i,j)/k \rceil$) will be descendants of its rightmost child. Then we move to the j+1 level and repeat the process, unless we have reached the leaves level. In this way, we guarantee that every pair of subtrees rooted at the same level will differ at most by $k-1$ leaves.

Fig. 1 illustrates an example where the list had $n = 594$ items, and we wanted to build a tree of degree $k = 5$ (and thus depth $d = \lceil log_5 594 \rceil = 4$). Each of the first $k-1$ children of the root has $\lceil L(1,0)/k \rceil$=119 leaves in its descendants, while the remaining 118 leaves are descendants of the rightmost child. Analogously, each of the first $k-1$ children of the first node at level 1 has $\lceil L(1,1)/k \rceil$=24 leaves in its descendants and the remaining 23 leaves are descendants of the rightmost child of that node. The resulting number of nodes in the example is 1 at level 0; 5 at level 1; 25 at level 2; 125 at level 3; 594 at level 4.

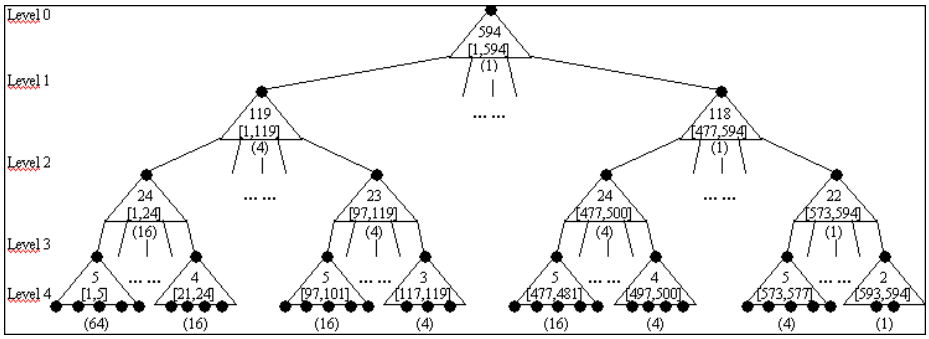


Fig. 1. A tree-augmented structure of degree 5. A triangle at each node provides numerical information about the subtree rooted at the node: (inside the triangle) number of leaves in the subtree and their range; (below the triangle) number of subtrees with the same structure contained in the whole tree

The tree we produce for a list can be used to navigate the list as follows. Every *internal* node of the tree corresponds to a view where the user is able to select which of the k subtrees to visit by choosing among k ranges of list items. A range is identified by its bounds (i.e., by a pair of list items), and indicates which range of leaves is included in the corresponding subtree. At each node, the user has also the opportunity to go back to the father of the current node.

The obtained tree structure is *Efficiently View Traversable* for $1<k<<n$, because:

- EVT1: the number of selections at each view (being less or equal to k+1, i.e., the maximum number of children k, plus the back-link to the father), is small compared to the size of the structure (that is linear in n).
- EVT2: the number of selections needed to travel from one view to any other (being upper-bounded by the distance between two leaves, which is twice the depth of the tree that is logarithmic in n) is small compared to the size of the structure (that is linear in n).

4 Related Work

In this section, we summarize the results of two user studies [7,8] that applied tree-augmentation techniques for item search tasks in long lists and we compare them with the results of another study [1], that used Alphaslider and traditional scrollbars.

In [7] and [8], the employed tree-augmented structures are similar to the one described in Section 3. Besides range selection, selection of a range bound as target was supported (fully by [7] and only for a central item by [8]). Note that adding this possibility corresponds to adding shortcuts in the tree structure that allow one to jump from internal nodes to some descendant leaves. Each range bound (except for the first one and the last one) was used for two adjacent ranges as the upper bound of the lower range and the lower bound of the upper range (see e.g., Fig. 2a). Both EVT requirements are still met. In fact, with respect to EVT1, the number of available selections remains constant (being equal to k ranges plus k+1 bounds) and thus small compared to the size of the structure. With respect to EVT2, the upper bound for the number of selections needed to travel from one node to another does not change with respect to the structure described in Section 3, but the shortcuts to the leaves reduce the number of selections for some items.

In [7], a study of the tradeoff between depth and breadth in large tree-augmented lists is presented. Eight users were asked to find a target word among 4096 dictionary words (each one between 4 and 14 characters in length) by subsequent selections on a touch screen, carried out by finger pointing. The screen was organized into 5 to 33 horizontal stripes alternating alphabetically ordered *range bounds* on odd (blue-colored) stripes and unlabeled *range-buttons* on even (red-colored) stripes. Users could select one of the range bounds if it already displayed the target, otherwise they could select a range-button to refine the search. The latter action resulted in the display of a new set of alternating range bounds and range-buttons (or only words, if the leaves level of the tree was reached). The procedure was repeated until the target word was found. To study the depth versus breadth trade-off, the number of ranges in a single screen was varied. There were a total of 4 sessions (consisting of 5, 10, 15, and 20 trials) with 2, 4, 8 and 16 ranges, which required 12, 6, 4 and 3 selections per trial, respectively. The average search time varied from 23.4 down to 12.5 seconds as the number of ranges increased from 2 to 16, thus demonstrating the advantage of using a broad and shallow structure over a narrower and deeper one. Subjects went through a long training time, consisting in 30 trials with 6 ranges at the beginning of the experiment and a supplementary training session before each of the 4 test sessions.

In a more recent study [8], a binary search method (called *BinScroll*) for finding a target item in long ordered lists was tested on 24 users. The method employed a specific instance (k=2) of the general tree-augmentation technique we described in Section 3. Consequently, the resulting structure was a binary tree. Participants were asked to find a target movie title among 10000 movie titles in English (with an average length of 17 characters) by subsequent selections carried out with the four arrow buttons of a standard keyboard. A CRT display showed three movie titles in a single column. The top and middle items were, respectively, the lower and upper bound of a range of movie titles. The middle and bottom items were, respectively, the

lower and upper bound of another range. The right arrow was used to select the middle item when it was the target; the up and down arrows were used to refine the search, selecting the upper or the lower range, respectively; the left arrow was used to go back to previous screens one by one. The average selection time for a search with the *BinScroll* technique turned out to be 14 seconds, which is close to the best result obtained in [7]. Participants were given a long training time also in this study, consisting of 15 minutes of training before the 25 test search tasks.

In [1], three different versions of the well-known Alphaslider were compared with each other and with a traditional scrollbar on a list item search task. 24 users were asked to find a target movie among 10000 movie titles in English (with an average length of 19 characters) by means of three Alphaslider versions and a traditional scrollbar. A CRT display was employed and selections were carried out by moving the slider-thumb of the Alphaslider or the scrollbar using a mouse. The interfaces' appearance and behavior was very similar to traditional horizontal scrollbars. The slider-thumb of each Alphaslider or scrollbar could be moved by directly dragging it or by clicking in the slide area. Below the slide area, an index provided cues about the alphabetical distribution of the elements, while above the slide area, the currently selected item was displayed. The three Alphasliders differed from the scrollbar because the granularity of mouse movements (and consequently slider speed) could be varied by clicking on different segments (top, center and bottom) of the slider-thumb or by moving the mouse at different speeds. The average selection time with the fastest of the Alphaslider versions was 24 seconds, while the average selection time with the traditional scrollbar was 25 seconds. Unlike the two previous studies, there were only 5 training trials per interface.

Although the results of the three studies cannot be directly compared, because they used very different input techniques (finger pointing, keypresses and mouse point-and-drag, respectively), the following considerations can be made:

- The interfaces described in [7] and [8] were both based on a tree-augmented list, resulting in an *Effectively View Navigable* structure, following the theory formulated by Furnas in [6].
- The *BinScroll* technique [8] seems to be very efficient with respect to the other ones; only the interface showing 16 ranges used in [7] offered a better performance, but the number of items was less than half (4096 vs. 10000 items); however, it must be pointed out that the input technique used in [8] might have improved performance, because users did not need to move their hand on the screen to select the displayed items, but just keep it on the 4 arrow keys.
- The two tree-augmented lists proposed in [7] and [8] seem to offer a notable advantage with respect to Alphasliders and scrollbars employed in [1], although it must be pointed out that in the latter study the training time was much shorter.
- The study by Landauer and Nachbar [7] suggests that increasing the number of ranges can lead to total task time reduction.

5 The Interfaces Considered

5.1 6-ary Tree-Augmentation of a List ("Six-Tree" Interface)

The first interface ("Six-Tree") exploits the tree-augmented structure obtained by applying the algorithm described in Section 3 and is inspired to the one presented in [7]. The screen is organized into 13 horizontal stripes alternating range bounds on odd stripes and range-buttons on even (dotted) stripes. An example of a starting screen is shown in Fig. 2a.

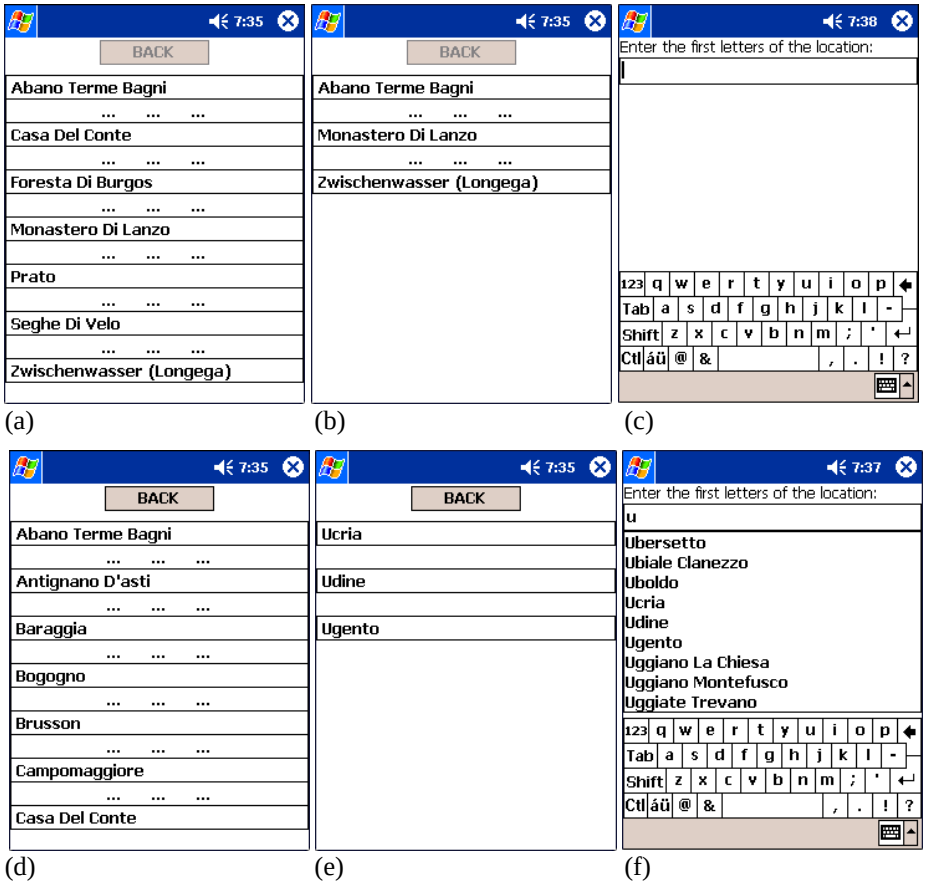


Fig. 2. Starting screens of Six-Tree (a), Bin-Tree (b) and Keyb (c); Six-Tree after the selection of the first range-button (d); example of last selection screen in Six-Tree and Bin-Tree (e); Keyb after entering a first character (f)

Each screen presents up to 6 range-buttons and 7 (directly selectable) range bounds. If the target is not one of the displayed range bounds, the user has to select

the proper range-button (e.g., Fig. 2d shows the effect of selecting the first range-button of Fig. 2a). The “BACK” button allows users to recover from possible errors by going back to previous views. Each selection produces a “click” sound, to give the user auditory feedback.

When the last selection screen is reached (which means that the number of possible choices is less or equal to 7) a screen without range-buttons is presented (see e.g., Fig. 2e). This event produces a “ding” sound, to give the user auditory feedback. After selecting a list item, a final confirmation screen is presented to the user. The same screen is presented when a range bound is selected as a shortcut to the last level. This screen (used also in the other interfaces we implemented) allows the user to check that the selected item was really the desired target, pressing an “OK” button to confirm her choice or a “BACK” button to return to the previous screen.

The adopted structure is Effectively View Navigable, because it is both Efficiently View Traversable (as already discussed in Sections 3 and 4) and also View Navigable. The two VN requirements are indeed met:

- VN1: in each screen, the range bounds make it unambiguously clear which selection is needed to reach the desired target.
- VN2: the outcome of each available selection is made clear by presenting only a small amount of information (specified by a single list item or by a pair of range bounds).

“Six-Tree” differs from the interface described in [7] in the following details:

- The range-buttons are distinguishable from the range bounds because they contain dots, instead of using a different color; this avoids undesirable effects due to possible color perception problems of users and takes into account the case of mobile devices with monochrome displays.
- The number of items in a screen has been determined by the available space of a typical PDA screen.
- The selections are carried out on a PDA touch screen by means of a stylus pen, instead of finger pointing on a larger touch screen.

5.2 Binary Tree-Augmentation of a List (“Bin-Tree” Interface)

The second interface (“Bin-Tree”), inspired by the one in [8], is operated exactly as “Six-Tree” and differs from it only in the number of range bounds and range-buttons (Fig. 2b). Each screen presents 3 range bounds on odd stripes and 2 range-buttons on even stripes.

“Bin-Tree” differs from the interface described in [8] in the following details:

- User input is based on stylus pointing instead of arrow-keypresses to become consistent with “Six-Tree” and thus allow a fair comparison between the two tree-augmented structures; stylus pointing is also typical of PDAs and high-end smartphones.
- Due to the change in user input style, range-buttons have been introduced between range bounds; range-buttons were unnecessary in [8], where ranges were selected by pressing the up or down arrow keys.

– While only the central range bound could be selected as the target item in [8], in our design all three range bounds are directly selectable for consistency with "Six-Tree".

5.3 List Scrolling Based on Keyboard Entry ("Keyb" Interface)

The last interface ("Keyb") implements the traditional idea of list scrolling based on keyboard entry described in the introduction of this paper.

This interface (Fig. 2c and 2f) employs the default QWERTY virtual keyboard of Pocket PCs to allow one entering characters of the target item. The text entry field is devoted to display entered characters and it is initially blank (Fig. 2c). A list of 9 items that can be directly selected appears after entering the first character (Fig. 2f) and contains only items starting with that character. The "BACKSPACE" button of the keyboard allows the user to delete the most recently entered characters one by one. At each selection of a character, accompanied by a "click" sound, the list is automatically scrolled down until reaching the first entry whose first characters match those entered by the user. This process continues until the user recognizes the target item in the displayed list and selects it.

6 Experimental Evaluation

6.1 Experimental Design

We recruited 48 subjects, 28 males and 20 females, from diverse backgrounds (e.g., university students, office clerks, engineers). Age ranged from 20 to 59, averaging at 30. Participants had at least a basic computer knowledge: only a few subjects used computers rarely (a few times in a year for 2 subjects and once in a month for 3 subjects), some subjects used computers at least weekly (7 subjects), while most subjects used computers daily (36 subjects). However, only five subjects owned a PDA and used it almost every day, while most of them (30 subjects) had never used a PDA, the remaining ones (13) had the opportunity to try a PDA for a very limited time (e.g., in shops or with a friend that owned one).

For our test we employed a Pocket PC with 320x240 screen resolution. Logging code automatically collected data about the number of selections and task completion times. The interfaces were operated by means of a stylus.

A between-subjects design was adopted, with interface type as the only independent variable (with three levels: "Six-Tree", "Bin-Tree" and "Keyb"). Subjects have been carefully assigned to the three groups to minimize differences in computer experience, age and sex among groups. Two subjects have been excluded from the data analysis due to their behavior during the test: one from the "Bin-Tree" group became confused by the binary search and one from the "Keyb" group stopped and began talking to the experimenter.

The following dependent variables were measured to characterize efficiency and usability:

- *Search time* to find a given target.
- *Number of errors* in terms of wrong range selections (for "Six-Tree" and "Bin-Tree"), wrong characters entered (for "Keyb"), and wrong list item selections (for all three interfaces).
- *Subjective assessment* of the ease of learning and using the interface, the ease of finding the target item (all on a five-point Likert scale, ranging from "Difficult" to "Easy") and the appropriateness of the number of items displayed on the screen (on a three-point scale: "Too few", "Appropriate", "Too many").

The list used for the evaluation contained all the 13926 unique names of Italian locations. This list is employed in real applications such as navigation systems that run on in-car devices and subsets of it are employed in applications such as automatic ticket vending machines in railway stations. A detailed analysis of all list items was carried out to identify a set of locations that were representative of the average complexity of selection. In particular, for each item and for each interface, the following data has been calculated:

- N_TAPS: the number of taps (each corresponding to a selection) needed to have the searched target displayed on the screen; in the interfaces based on tree-augmentation, this corresponded to the number of range selections needed; in "Keyb", this corresponded to the number of characters that needed to be entered.
- N_MOVES: this parameter measured the number of times a user must move the stylus to a different area of the screen to carry out the needed (N_TAPS) selections. It was calculated by subtracting from N_TAPS the number of times that a tap must be made on the same position of the previous tap.

Only locations with N_TAPS and N_MOVES values close to the mean or modal ones (shown in Table 1) were chosen as targets for the test and trial sessions (the "Chosen" rows in Table 1 shows the resulting ranges of values for the chosen target locations).

Table 1. Mean, mode and values for the chosen targets of N_TAPS and N_MOVES in the three different interfaces

Interf.	Stats	N_TAPS	N_MOVES
Six-Tree	Mean	4.35	2.79
	Mode	5.00	3.00
	Chosen	4-5	2-3
Bin-Tree	Mean	11.82	5.47
	Mode	13.00	5.00
	Chosen	12-13	5-6
Keyb	Mean	4.40	3.28
	Mode	3.00	2.00
	Chosen	3-4	2-3

During each test session, subjects completed 20 training trials (to familiarize with the assigned interface) and 10 trials during which dependent variables were measured.

This high number of training trials is deliberately consistent with the studies in [7] and [8]. At the end of each test session, the previously described subjective assessment was collected by means of a questionnaire.

Based on the results reported in [7], we expected that users' performance at finding an item in the list by means of a tree-augmented structure would have been better in the broad and shallow "Six-Tree" with respect to the narrow and deep "Bin-Tree". As anticipated in Section 4, we conjectured that the better performances reported in [8] for the binary search interface could be due to the arrow-keypresses input technique. We also hypothesized that users' performance at finding an item in the list and their assessment of the ease of use of the interface would have been better in "Keyb" than in the interfaces based on tree-augmentation, for the following reasons:

- "Keyb" requires a smaller number of selections than the interfaces based on tree-augmentation (especially "Bin-Tree", as it can be seen from Table 1).
- In [8], many users reported difficulties and frustration in dealing with the alphabetical ordering in the interfaces based on tree-augmentation.
- Despite the familiarity with the alphabetical search task (frequently performed by anyone with dictionaries, phone directories, etc.), retrieving an item in a tree-augmented ordered list requires a large number of comparisons between the target and list items and, for this reason, it seems to be more cognitively demanding than the simpler task of (possibly partially) spelling and typing a name, as in list scrolling based on keyboard entry.

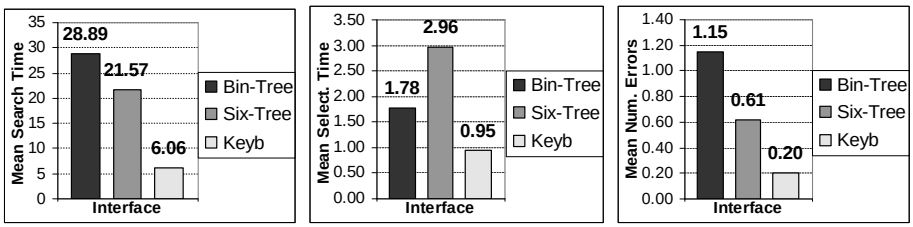
7 Results

7.1 Mean Search and Selection Times

For each test subject, the mean search time has been calculated considering all the 10 different test trials (Fig. 3a). A one-way ANOVA pointed out a significant effect of interface type ($F(2,43)=35.09$, $p<0.001$). A Tukey post-hoc test pointed out that the effect was significant for all pair-wise comparisons. In particular, "Keyb" (6.06s) was better than both the interfaces based on tree-augmentation ($p<0.001$) and "Six-Tree" (21.57s) was better than "Bin-Tree" (28.89s) ($p<0.05$). These results are consistent with our hypotheses and with the results of [7].

We also studied the time to make single selections (Fig. 3b) by dividing the search time by the number of selections, taking into consideration only error-free trials (those with no selection errors). It is interesting to note that tree-augmented list structures require more time than list scrolling based on keyboard entry. A one-way ANOVA pointed out that the effect was significant ($F(2,43)=18.48$, $p<0.001$) and the Tukey post-hoc showed that it was significant for all pair-wise comparisons ($p<0.05$). Mean selection times are: 1.78s with "Bin-Tree", 2.96s with "Six-Tree", 0.95s with "Keyb". The longer selection times of the interfaces based on tree-augmentation with respect to "Keyb" can be explained by a higher cognitive load required to perform each selection. The longer selection time of "Six-Tree" with respect to "Bin-Tree" is

explained by the need to make a greater number of comparisons per screen. However, mean search time of “Bin-Tree” is anyway larger with respect to “Six-Tree” due to the higher number of selections needed to find the target item.



(a) (b) (c)

Fig. 3. Means: search time (a), selection time in error-free trials (b), number of errors (c)

7.2 Number of Errors

For each subject, the mean number of errors has been calculated considering all the 10 different test trials (Fig. 3c). A Kruskal-Wallis test, pointed out a significant effect ($p<0.05$). A Dunn post-hoc comparison showed that subjects made less errors with “Keyb” than with the interfaces based on tree-augmentation at a significant level ($p<0.05$), while no significant differences ($p>0.05$) were found between “Six-Tree” and “Bin-Tree”.

Moreover, it is interesting to note that more than half of the subjects (9 out of 15) did not make selection errors at all with “Keyb” and the remaining subjects in the “Keyb” condition (6 out of 15) made at most 6 errors (considering all the 10 trials). On the contrary, nearly all subjects made at least one selection error with “Bin-Tree” and “Six-Tree”, and the number of errors made was also higher, as shown in Table 2.

Table 2. Number of subjects making errors with the different interfaces

Number of errors	Bin-Tree	Six-Tree	Keyb
No errors	1	0	9
From 1 up to 6 errors	6	11	6
More than 6 errors	8	5	0

7.3 Subjective Assessment

For each question on a 5-point scale, a Kruskal-Wallis test was employed to analyze data (Table 3 reports the means). With respect to ease of learning and ease of use, the different interfaces were all rated positively with no significant differences ($p>0.05$).

With respect to the ease of finding the target item, the Kruskal-Wallis test pointed out a significant effect ($p<0.05$). Dunn post-hoc showed that “Keyb” was better than “Bin-Tree” and “Six-Tree” at a significant level ($p<0.05$), while there were no

significant differences between the two interfaces based on tree-augmentation ($p>0.05$).

Table 3. Mean scores of the subjective assessments per interface

Assessment	Bin-Tree	Six-Tree	Keyb
Ease of learning (1-Difficult; 5- Easy)	3.60	3.19	3.87
Ease of use (1-Difficult; 5- Easy)	3.33	3.00	3.80
Ease of finding target item (1-Difficult; 5- Easy)	3.60	3.38	4.53

With respect to the appropriateness of the number of items displayed on the screen (rated on a three-point scale: "Too few", "Appropriate", "Too many"), it is interesting to note that most users of "Bin-Tree" and "Keyb" (respectively 12 and 13 out of 15) thought that the displayed number of items was appropriate, while nearly half of "Six-Tree" users (7 out of 16) thought that the interface displayed too many items.

8 Conclusions and Future Work

The experimental study presented in this paper provides designers with useful information to help them in choosing structures to improve item search tasks in long lists on the small screen of mobile devices. In the following, we outline the most important considerations about the findings of the experiment.

First, the idea of tree-augmentation of lists proposed by Furnas' [6] does not offer - in the context we considered - any benefit to the task of finding a target item in an alphabetically ordered list with respect to the more traditional technique of list scrolling based on keyboard entry. With respect to "Keyb", the two interfaces based on tree-augmentation were found to have a significantly longer task completion time (approximately three and four times greater) and to lead to a significantly greater number of errors. Moreover, they imposed a greater cognitive load on users as highlighted by both the significantly longer selection time and by the difficulties in dealing with the alphabetical ordering, directly observed by the experimenters on many users and explicitly reported by some of them. These results suggest that list scrolling based on keyboard entry should be preferred to tree-augmented lists, when it is possible to have a virtual (or physical) keyboard available. Nevertheless, techniques based on tree-augmentation of a list could still be useful when no full keyboard (either virtual or physical) can be provided (e.g., in some wearable computer scenarios as those envisaged in [8]).

With respect to the breadth versus depth trade-off in tree-augmentation techniques, task completion time was shorter with the broad and shallow tree-augmented list (21.6s with "Six-Tree"), rather than the narrow and deep one (28.9s with "Bin-Tree"). This confirms the results presented in [7], but in the different context of mobile devices. In particular, the advantage of breadth versus depth, shown with finger pointing on a large touchscreen in [7], holds also using stylus pointing on a PDA (the conditions of our study).

We are currently planning to investigate in more detail the following two interesting aspects, highlighted by the obtained results. First, the fact that nearly half of “Six-Tree” users thought that the interface displayed too many items seems to suggest that they perceived it as imposing a greater cognitive demand (at least for a single selection) with respect to the narrower “Bin-Tree”. Intermediate breadth levels would thus be worth investigating. Second, the mean single selection time of “Bin-Tree” (1.78s) was found to be much longer than that found in [8] (1s). This fact might be due to the different kind of input techniques used in the two experiments (respectively, stylus pointing versus arrow-keypresses). An experimental study would be necessary to directly compare these two different input techniques.

Finally, we are considering to carry out additional experiments to contrast tree-augmentation techniques with other techniques that are currently employed when no full keyboards (either physical or virtual) are available. For example, some mobile phones implement a sort of list scrolling based on keypad entry where pressing a keypad key corresponds to specifying an “OR” among the letters associated to that key (e.g., pressing the “2” key followed by the “3” key displays only list items whose first letter is “a”, “b” or “c” and whose second letter is “d”, “e” or “f”).

References

1. Ahlberg, C., Shneiderman, B.: The Alphaslider: A Compact and Rapid Selector. Proc. Conf. on Human Factors in Computing Systems (CHI ’94), ACM Press (1994) 365-371
2. Ayatsuka, Y., Rekimoto, J., Matsuoka, S.: Popup Vernier: a tool for sub-pixel-pitch dragging with smooth mode transition. Proc. Symp. User Interface Software and Technology (UIST ’98), ACM Press (1998) 39-48
3. Bederson, B. B.: Fisheye Menus. Proc. Symp. User Interface Software and Technology (UIST ’00), ACM Press (2000) 217-225
4. Bederson, B.B., Hollan, J.D.: Pad++: a zooming graphical interface for exploring alternate interface physics. Proc. Symp. User Interface Software and Technology (UIST ‘94), ACM Press (1994) 17-26
5. Furnas, G. W.: Generalized Fisheye Views. Proc. Conf. on Human Factors in Computing Systems (CHI ’86), ACM Press (1986) 16-23
6. Furnas, G. W.: Effective View Navigation. Proc. Conf. on Human Factors in Computing Systems (CHI ‘97), ACM Press (1997) 367-374
7. Landauer, T., Nachbar, D.: Selection from alphabetic and numeric menu trees using a touch screen: breadth, depth, and width. Proc. Conf. on Human Factors in Computing Systems (CHI ‘85), ACM Press 73-78 (1985)
8. Lehikoinen, J., Salminen, I.: An Empirical and Theoretical Evaluation of BinScroll: A Rapid Selection Technique for Alphanumeric Lists. Personal and Ubiquitous Computing Vol.6, Issue 2, Springer-Verlag (2002) 141-150