Designing Phrase Builder: A Mobile Real-Time Query Expansion Interface

Tim Paek, Bongshin Lee, Bo Thiesson
Microsoft Research
One Microsoft Way
Redmond, WA 98052, USA
{timpaek, bongshin, thiesson}@microsoft.com

ABSTRACT

As users enter web queries, real-time query expansion (RTQE) interfaces offer suggestions based on an index garnered from query logs. In selecting a suggestion, users can potentially reduce keystrokes, which can be very beneficial on mobile devices with deficient input means. Unfortunately, RTQE interfaces typically provide little assistance when only parts of an intended query appear among the suggestion choices. In this paper, we introduce Phrase Builder, an RTQE interface that reduces keystrokes by facilitating the selection of individual query words and by leveraging back-off query techniques to offer completions for out-of-index queries. We describe how we implemented a small memory footprint index and retrieval algorithm, and discuss lessons learned from three versions of the user interface, which was iteratively designed through user studies. Compared to standard auto-completion and typing, the last version of Phrase Builder reduced more keystrokes-per-character, was perceived to be faster, and was overall preferred by users.

Categories and Subject Descriptors

H5.2 Information interfaces and presentation: User Interfaces. - Graphical user interfaces. H.3.3 Information Search and Retrieval: Query formulation.

General Terms

Design, Human Factors

Keywords

Interactive query expansion, real-time, mobile, auto-completion

1. INTRODUCTION

Nowadays, users of major search engines are probably all familiar with having a drop-down box appear as they type their web queries showing selectable popular matches garnered from search query logs. Some engines (e.g., [13]) use *prefix matching* to show query matches which complete the text entered so far (e.g., "**spea**keasy" for "spea"), along with the number of web results that will appear for that query. Other engines (e.g., [16]) also enable *infix matching* to show query matches which contain the entered text anywhere in the suggestion (e.g., "britney **spea**rs" for "spea"). In all cases, matches are shown in rank order according to a relevance function. What looks like simple auto-completion to the typical user is known in the information retrieval (IR) community as *query expansion*, the process of supplementing an original query with additional terms that are ranked by some numeric score [11]. Because query quality directly impacts search result quality [9], query expansion has been pursued as a method for improving IR performance. With *interactive* query expansion, users participate in the selection of terms for the initial and/or subsequent queries [11]. *Real-time query expansion* (RTQE) is a variant of interactive query expansion in which expansion choices are presented while users are still formulating their queries [32]. RTQE interfaces can display expansion choices (or simply *suggestions*) after each word (e.g., [32]) or each character. Suggestions can constitute completions [13][16], substring matches [16], and even spelling corrections [11][13][16]. RTQE interfaces have also been used as a mechanism for pseudo-relevance feedback [32].

Although previous research has shown that RTQE interfaces can lead to initial queries of better quality and more user engagement in search [31], the question of whether RTQE interfaces can also serve the purpose of reducing keystrokes for mobile search has only recently begun to be explored [20]. By reducing keystrokes, RTQE interfaces could alleviate some of the burden of using mobile devices with deficient input means. In this paper, we present a new RTQE interface called *Phrase Builder* that not only retains the benefits of interactive query expansion, but also captures an additional benefit: reduced keystrokes for mobile web queries. In theory, RTQE interfaces should be able to reduce keystrokes because users can select choices instead of having to type them out in full. However, this benefit does not apply when the query users have in mind either does not show up among the suggestions or only partially appears among the suggestions. To handle the former case (i.e., out-of-index queries), Phrase Builder leverages back-off query techniques to suggest possible phrases composed of words that are in the index. To handle the latter case (i.e., partial matches), it enables the selection of individual words as well as whole phrases in the suggestions. Shipped as part of Live Search Mobile (Figure 1), Phrase Builder is available for download[1] on all Windows Mobile phones.

This paper consists of three contributions. First, we elaborate on the motivation for Phrase Builder and detail how we implemented a small memory footprint index and retrieval algorithm with back-off query techniques on a mobile device. Second, we describe three versions of the user interface, which was iteratively designed through user studies. Third, we evaluate the different versions at both quantitative and qualitative levels, and discuss trade-offs that were made in order to arrive at the final product version.

MobileHCI'09, September 15 - 18, 2009, Bonn, Germany.
ACM 978-1-60558-281-8.

[1] Live Search Mobile download: http://wls.live.com

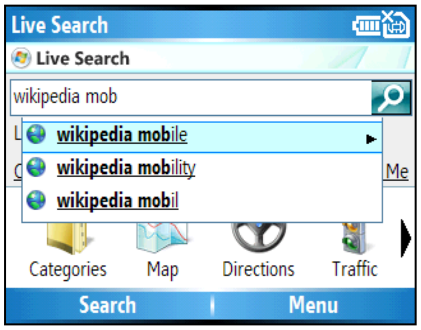


Figure 1. Screenshot of Phrase Builder for Live Search Mobile on a Windows Mobile smartphone

2. APPROACH

2.1 Motivation

According to market research, mobile phones are poised to rival the PC as the dominant Internet platform in the near future [17]. As mobile Internet usage continues to rise, a recent analysis has shown that the limited text-input capabilities of mobile devices do impact the way users search for information [5]. Furthermore, because the average number of queries per session is significantly less for mobile devices than for desktop computers, researchers have highlighted the importance of getting useful query expansions to users during their initial formulation of queries [21].

With the above mobile search findings in mind, we sought to develop an RTQE interface customized for mobile usage. We were motivated by three goals:

1. Always provide suggestions immediately.
2. Reduce keystrokes for web queries.
3. Design a user interface that requires little cognitive effort and no training.

On the desktop, these three goals are easily achieved using standard auto-completion. On the mobile device, where typing is constrained, we endeavored to improve upon auto-completion by allowing users to select the individual words of a suggestion and complete queries that are out-of-index but whose words are in the index.

To meet our first goal, our index and retrieval algorithm must not only provide suggestions quickly, but *always* do so even when the data connection is slow. For this reason, and the fact that we wanted to allow for updating of our index with user-generated queries (which can be cleared), we decided to perform RTQE on the mobile device itself. As such, we needed a small memory footprint index and retrieval algorithm, as well as techniques for maximally utilizing whatever index we stored on the device.

For our second goal, we decided to minimize keystrokes instead of time because measures of speed can be easily confounded by factors such as familiarity with the keyboard layout, finger size, nail length, age group, etc. By focusing on keystrokes (like previous research [20]), we can avoid having to average across lots of different participants for our experiments. In the long term, as users become more proficient with an RTQE interface, reducing keystrokes should also improve speed of entry.

It is important to note that our second goal is to reduce keystrokes for web queries, not general text entry. Predictive text entry methods such as T9 [15] and POBox [30] utilize lexicons or dictionaries that are general-purpose and domain-independent. However, our index is a corpus of web queries with entries and relevance (or popularity) scores that change over time, and the back-off techniques we discuss in Section 2.4 relate specifically to query structure. Although it is possible to use both predictive text entry methods and RTQE, in practice, most RTQE interfaces disable predictive text entry to avoid confusing users with multiple suggestions.

Finally, our third goal, which is the main focus of this paper, relates to mobile usability. As will be evident in our user studies, this goal required us to ultimately sacrifice some novelty in the user interface for familiarity and ease-of-use. Starting from Section 3, we discuss how we iteratively designed our user interface to achieve this critical goal.

2.2 Device Details

Because most Windows Mobile users, our target audience for Live Search Mobile, own smartphones with a directional pad (i.e., d-pad), we specifically designed our RTQE interaction around this input modality. Although we could have evaluated our interface on a smartphone with a numeric keypad, we decided to use a smartphone with a miniature QWERTY keyboard for three reasons: 1) any significant keystroke reduction we find on a Qwerty device is likely to be more significant for a numeric keypad device, because typing is not as constrained, 2) QWERTY devices make up the majority of the smartphone market [23], and 3) previous research has already demonstrated that RTQE can reduce keystrokes on numeric keypad devices [20].

We implemented our Phrase Builder prototype on a Samsung Blackjack SGH-i608 smartphone with a 320x240 pixel QVGA screen. The user interface was developed using Microsoft Windows Mobile 5 Smartphone SDK and PocketPiccolo.NET [14].

2.3 Index and Retrieval Algorithm

In this section, we describe how we implemented a small memory footprint index and retrieval algorithm for mobile search based on k-best suffix arrays. A k-best suffix array is a convenient data structure for encoding an index which facilitates fast and efficient retrieval of *k* "best" matches according to some numeric score. Here, we provide sufficient technical details for those familiar with a k-best suffix array to reproduce our implementation. To learn more about k-best suffix arrays and how they compare to other data structures and retrieval algorithms, we refer the reader to [6][8].

Similar to traditional suffix arrays [29], k-best suffix arrays arrange all suffixes in the dictionary (in our case, the query logs) into an array. However, k-best suffix arrays arrange the suffixes according to two alternating orders – the usual lexicographical ordering and an ordering based on a numeric figure of merit [3]. Because the k-best suffix array can be sorted by both lexicographic order and the figure of merit, it is a convenient data structure for finding the k-most popular matches for a substring. For notation, we henceforth express substrings as *wildcard queries*, or queries that utilize wildcards (*) to match zero or more characters. We also denote the text the user has entered via typing or selection at the time of the query as the *text-so-far*. In providing suggestions, k-best suffix arrays can support both prefix matching (by appending a wildcard to the end of the text-so-far; e.g., "spea*" retrieves "**spea**keasy") as well as infix matching (by appending a wildcard to the beginning and end of the text-so-far;

e.g., "*spea*" retrieves "britney **spea**rs") in a computationally efficient way.

In order to minimize memory footprint, we decided to exclude substring matching within words (e.g., "i-**spea**k") and to modify k-best suffix arrays to support only substring matching of word prefixes (e.g., "britney **spea**rs"). Technically, this modification is achieved by allowing the k-best suffix array to contain only pointers (see [6] for more details) to the beginning of words. We viewed the word prefix matching as sufficient for mobile RTQE. By making this modification, we were able to reduce the memory footprint by a factor of 5. Overall, our index is roughly 1.3 times[2] the size of the raw dictionary text (without the figure of merit).

Besides meeting memory constraints, our retrieval algorithm has to be computationally efficient. With k-best suffix arrays, the k-most popular matches can be found in time close to *O(log N)* for most practical situations, with a worst case guarantee of *O(sqrt N)*, where *N* is the number of words in the query logs [6]. In contrast, a standard suffix array finds *all* matches to a substring in *O(log N)* time [29], but does not order the matches by popularity (one of the most commonly used figure of merits). Finding the k-best matches for short substrings, as the users begin to type their intended queries, can therefore be prohibitively demanding for the standard suffix array, because in this case we will have to sequentially search through a large set of returned matches in order to determine the k-best.

The query logs used for the Phrase Builder index were collected from Live Search Mobile for a period of four months. Because the Blackjack smartphone only has about 25 MB of available RAM, in encoding the k-best suffix arrays, we limited the query logs to include just those queries which had at least 5K popularity hits. Overall, this left us with an index of roughly 122K unique queries taking up only 2.2 MB of storage space.

2.4 Back-off Techniques

The size of the index used for RTQE naturally affects the quality of the suggestions. With only 122K entries, many intended queries may not exist in our index. In this section, we describe back-off query techniques that allow Phrase Builder to provide suggestions for many out-of-index queries, thereby increasing the coverage of our index.

In order to fully capitalize on the vocabulary of the index, we use the following algorithm to generate supplementary suggestions. This algorithm is used whenever we cannot retrieve enough suggestions for the text-so-far:

(1) From the text-so-far, generate back-off queries by iteratively replacing the token words w_1 to w_{n-1} of the text-so-far with wildcards until only the last word w_n is left.

(2) For each back-off query, retrieve matches from the index (which we henceforth denote as *back-off matches*).

(3) For each back-off match, replace the substring in the back-off match corresponding to the wildcard with the substring in the text-so-far that was initially replaced in step (1).

[2] The factor depends on the average length of entries in the index. The factor of 1.3 is calculated for our index of query logs which had an average entry length of 14 characters. The factor is smaller for an index with longer entry lengths, and the other way around.

Table 1. Back-off queries, back-off matches and generated suggestions for the query "chai tea i"

Query	Index Match	Suggestion
chai tea i*	×	×
* tea i*	green tea ice cream	chai tea ice cream
* i*	google images	chai tea images
i*	itunes	chai tea itunes

To illustrate how this algorithm works, suppose the user is intending to search for "chai tea ice cream" and has already typed "chai tea i." We first attempt to find suggestions for the query "chai tea i*." If too few choices are returned, according to step (1), we generate the back-off queries listed in the first column of Table 1. The first back-off query is "* tea i*" which was obtained by replacing the first word "chai" with a wildcard. Submitting this as a new query in step (2), we retrieve "green tea ice cream" from the index. Applying step (3), we now replace the word "green" in the back-off match corresponding to the wildcard with the originally replaced word in the text-so-far in step (1), viz. "chai." This produces the suggestion "chai tea ice cream" in the third column. If we still do not have enough suggestions, we back-off to "* i*" which retrieves "google images," resulting in "chai tea images." Finally, we continue backing-off until we have enough suggestions or until we ultimately back-off to the individual word, or *unigram*, "i*." By the time we reach the unigram, we have lost all surrounding context; hence, "i*" retrieves "itunes" resulting in the peculiar suggestion "chai tea itunes."

In order to avoid peculiar suggestions in the shipped version of Phrase Builder, which has 6 separate indexes, we decided to rank back-off suggestions lower than non-back-off suggestions, and to sort back-off suggestions by how much of the original text-so-far was replaced by wildcards. A promising research direction for improving the relevance of back-off suggestions is to generate back-off queries using semantic (e.g., <NamedEntity>) and syntactic categories (e.g., <Adjective>) to inform which words to replace with wildcards.

Because Phrase Builder employs back-off queries that ultimately match unigrams, it almost always provides suggestions. Even for out-of-index queries such as "this is a test of the american broadcast system," as long as the words are somewhere in the index, and individual word selection is possible (as we discuss in Section 3), we can provide suggestions for each word, and hence, compositionally for the entire phrase. In the worst case, for out-of-vocabulary (OOV) words, the user will have to type out the word verbatim. But because the shipped version of Phrase Builder adds recent queries to the index, OOV words quickly become in-vocabulary (IV) and hence available for back-off matching.

2.4.1 *Pruning the Index*

Our back-off query techniques are very similar to those used in speech recognition for pruning language models [18], where the probability of an unseen or infrequent *n*-word sequence (i.e., *n-gram*) is estimated by the probability of its (*n-1*)-word sequence until finally, the unigram (*1*-word) estimate is used [24]. The goal of language model pruning is to exclude those n-grams which can be estimated by their back-off probabilities so as to take a small loss in coverage but acquire a big gain in memory performance (since pruning reduces the number of parameters to estimate). Similarly, for RTQE we can prune what queries are included in

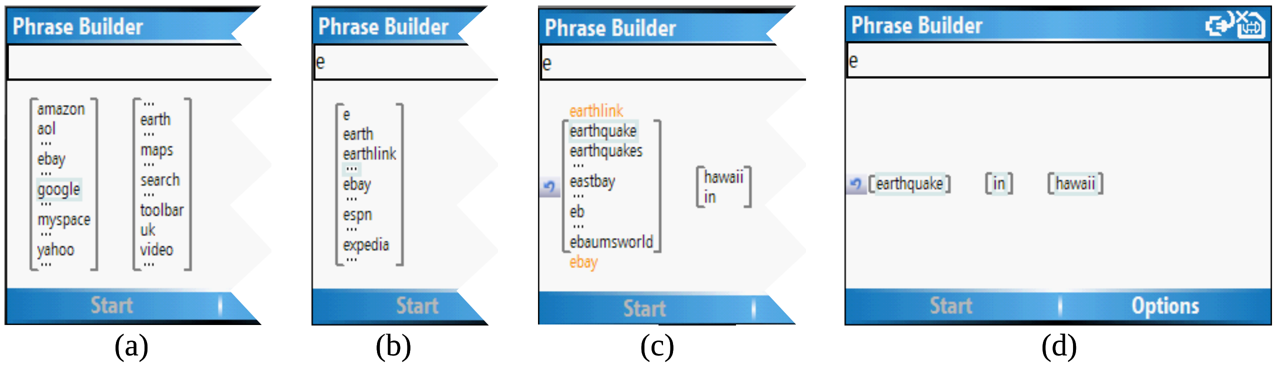


Figure 2. Screenshots of Phrase Builder V1 showing how users can compositionally build query phrases from word columns.

the index to reduce the storage size on the mobile device. One simple pruning algorithm is to iterate through the query logs in rank order building up a lexicon and adding only queries with OOV words to the index. For example, if the index already contains "britney spears music" and "music video," we do not need to add "britney spears music video" since our back-off query techniques would provide that as a suggestion.

2.5 Related Research

Besides the previously mentioned RTQE interface that initiates query suggestions after each word [31], very little research has investigated mobile RTQE. Recently, [20] found that RTQE on a numeric keypad phone could reduce keystrokes by about 50%, though this result was for multi-tap and only for queries that existed in the index. To our knowledge, no prior research has examined how to facilitate RTQE when the query users have in mind does not appear in the index (for which we use back-off query techniques), or partially appears among the choices (for which we use individual word selection), nor how to deal with these cases within the constraints imposed by mobile devices.

In IR, many different methods for generating query recommendations and expansions have been pursued. Among methods that utilize query logs, [34] combined a model of sequential search behavior with content-based similarity to recommend related queries. [10] used probabilistic correlations between query words and document words in the query logs to generate expansion words. For mobile devices, [22] used contextual signals such as time of day and inferred location to predict and rank expansions.

Outside of query expansion, [4] investigated automatically enriching mobile page content by adding additional relevant words for indexing. Because many mobile devices have small displays, [7] explored displaying related queries instead of snippet text in the search results. [19] proposed design guidelines for mobile search interfaces on small screens, but did not consider any interfaces with RTQE. [31] explored treating individual words of a suggestion list as buttons that can be touched in multimodal refinement of a voice search results.

To deal with the difficulties of text entry on numeric keypad phones, various consecutive and concurrent text-entry techniques, which typically utilize a general-purpose lexicon as discussed previously, have been developed [27][30][33], with T9 being the most commercially successful [15].

3. USER INTERFACES

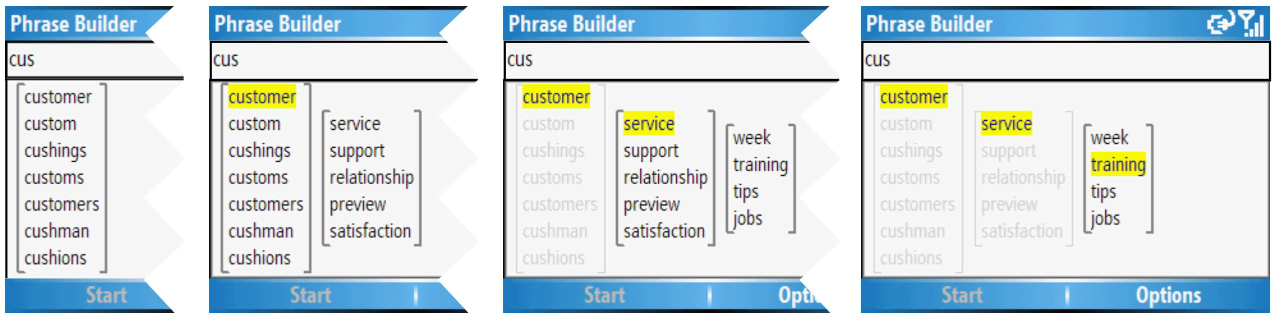
We now survey three versions of the Phrase Builder user interface corresponding to three attempts to realize the second and third goals outlined in Section 2.1. Our intention is not to argue that one is definitively better than another, but to highlight the trade-offs that had to be made in order to better fulfill our goals. Other researchers may find the lessons we learned to be useful for developing their own RTQE interface. For a live demonstration of the three versions, please see our video supplement.

3.1 Version 1

Previous studies have shown that increasing the level of user control over query term selection in general seems to improve search effectiveness [1][25]. For the first version, we designed a user interface where users could not only select individual words and compositionally build up phrases, but also retrace their word selections. In formulating queries, users could in effect "browse" the search query logs. We facilitated this kind of browsing through *word columns*.

Figure 2 shows a sequence of interactions with Phrase Builder Version 1 (V1) for the intended query "earthquake in hawaii." In Figure 2(a), the user has not yet entered any characters into the textbox. The first column contains suggestions for the first word. Once a word is put into focus in the first column, the second column is updated to show suggestions for the second word conditioned on the first word. V1 animates this update to help users stay in context. Focused words are marked with a sky blue border. Since "google" has the focus in the first column, V1 shows all suggestions for the second word, conditioned on "google" as the first word. In Figure 2(b), the user has typed "e," so V1 displays all suggestions in the first column that start with "e." The ellipsis represents all suggestions that occur lexicographically within a word range. For example, in Figure 2(b), the range is between "earthlink" and "ebay." When the user shifts the focus to an ellipsis (i.e., "…"), no words are displayed in the next column. When the user selects the ellipsis by clicking the 'OK' button at the center of the d-pad, suggestions between "earthlink" and "ebay" are displayed in the column, as shown in Figure 2(c). Notice that word ranges are marked in orange at the top and bottom of the column. Furthermore, the back button icon is now shown to the left of the column to allow users to return to the previous choices.

Each word column is conditioned on selections in the previous columns. Similar to typing, queries are composed in a left-to-right fashion. To compose an entire phrase, the user simply moves the



(a) (b) (c) (d)

Figure 3. Screenshots of Phrase Builder V2 showing word columns without word ranges for drilling down.

focus from one word column to the next column on the right by clicking the right arrow on the d-pad. Once a word is selected, the column is collapsed to a focused word (Figure 2(d)). If the entire sequence of focused words is what the user desires, the user just clicks 'OK' on the d-pad. This causes the entire phrase to be pasted into the textbox. To retrace, the user can simply move the focus to the left, to whichever word column is desired. For example, if the user wishes to just type "earthquake," the user can move left until "earthquake" is highlighted and then click 'OK.' Finally, the user is never prohibited from inserting new characters into their queries, which can be performed by changing the focus to the textbox and placing the cursor wherever desired.

3.1.1 Generating Word Column Suggestions

We now describe how we generate suggestions for the word columns. For any word column, we only have *n word-only* slots in which to place words, and *(n+1) open slots* in which to place either words or ellipses. The middle slot always contains the highest ranked expansion word. For example, because "google" is the most popular word in the query logs, it is placed in the middle slot of the first column in Figure 2(a). Above and below the middle slots are open slots and word-only slots in alternating fashion.

Our algorithm for filling suggestions in the word columns proceeds as follows: After placing the highest ranked word in the middle slot with a focus, we iterate through the k-best words that have been retrieved from the index and place them in the word-only slots either before or after the highest ranked word, depending on lexicographical order. After the word-only slots are filled, if only one word can fill an open slot, we place the word in that slot (e.g., "amazon" in the first column in Figure 2(a)). Otherwise, we place an ellipsis. In actuality, our algorithm for retrieving k-best words retrieves k-best phrase matches. To generate the words, we split the phrase matches into words and place the words into the appropriate columns.

3.1.2 Usability Study & Lessons

We conducted a usability study to evaluate V1 using the "think-aloud" protocol [26], where participants verbalized their thoughts about the interface as they entered their own queries as well as specified queries. Due to space limitations, here we just summarize the Methods and Results.

Ten participants (5 males and 5 females) from the Seattle metropolitan area were recruited by a professional contracting service. Because no screening was conducted for age, the age of the participants ranged from 42 to 66 with an average of 56.2, which was unexpectedly higher than the age of our target audience. In terms of Procedure, we first taught participants the basics of using a Blackjack Smartphone and then introduced them to the V1 interface. As the participants interacted with the V1 interface, they voiced aloud their impressions.

In terms of Results, with respect to query formulation, several participants liked how the interface allowed them to browse what other people had searched for on the web. They noted how surprised they were to see certain queries in the logs. Regarding composition, many participants enjoyed how they could paste all or part of a query phrase into the textbox by simply clicking the 'OK' button on the d-pad. Several participants also enjoyed the spelling correction aspect of V1, which has also been noted as a benefit of other RTQE interfaces [11].

On the other hand, participants were less enthusiastic about entering queries for which they knew their desired terms. Almost all participants stated that selecting query terms in the word columns seemed "disruptive." This observation is consistent with previous research in which users were more likely to use interactive query expansion when their information needs were vague and unarticulated [12][32]. In particular, many participants found the task of examining suggestions to be cognitively demanding, which is also consistent with prior research showing how reluctance to use interactive query expansion may be linked to the added cognitive load of judging the relevance of suggested terms [2]. Finally, most participants found it burdensome to keep track of the word ranges in drilling in and out of an ellipsis. Many participants also felt that the animation used to splay suggestions and collapse word columns slowed them down.

3.2 Version 2

Because allowing users to browse the search query logs is not one of our goals but reducing keystrokes for mobile web queries is, we decided to re-design our user interface to overcome the drawbacks participants had noted in the previous usability study. In particular, we removed all animation, made all open slots into word-only slots (i.e., no ellipsis and drill-downs), and re-ordered the word-only slots by popularity from top to bottom. These changes were all made as an effort to reduce cognitive load. Figure 3 shows a sequence of interactions with Phrase Builder V2 for the intended query "customer service training."

Notice that the appearance of V2 is more in the direction of RTQE interfaces that users are accustomed to on the desktop, namely, auto-completion. The major difference is that instead of showing suggestions as whole phrases for selection, individual words can be selected via the word columns. For example, in Figure 3(a), the user has typed "cus." Seeing that the word "customer" has come

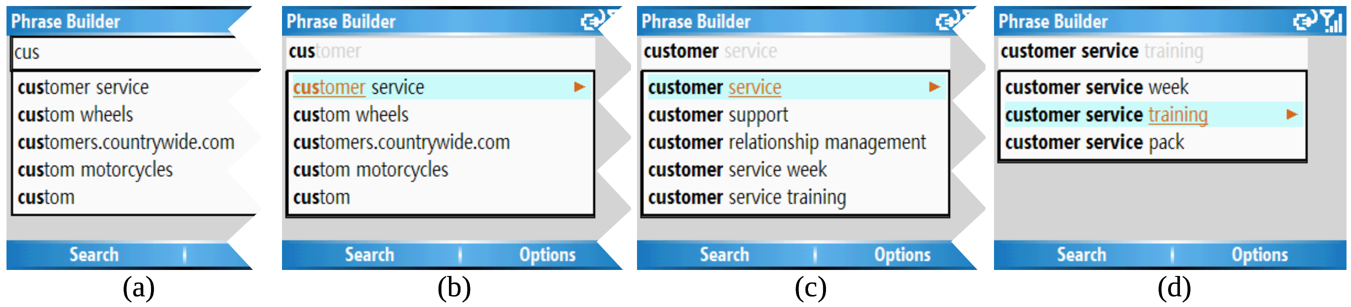


Figure 4. Screenshots of Phrase Builder V3 showing word selection integrated into the expansion choices.

up on top, the user focuses on the word in Figure 3(b) and then selects the next desired words in Figures 3(c)-(d) by moving right. Because suggestions are organized into word columns, if the user decides to remove a word, they can easily retrace by moving left, just as in the V1 user interface.

In the next section, we discuss the results of a controlled experiment we conducted assessing V2 against standard auto-completion, which does not allow individual word selection. Since the evaluation of the final Phrase Builder user interface utilizes the same methods and evaluation criteria, we now introduce the final version and postpone discussion of V2's evaluation for Section 4.

3.3 Version 3

The final user interface can be viewed as a marriage of V2 and auto-completion, the RTQE interface most common on desktop search engines. Figure 4 shows a sequence of interactions with Phrase Builder V3 for the intended query "customer service training." As users type characters, a drop-down box appears showing suggestions for the text-so-far. If they notice the intended query among the suggestions in the drop-down box, they can use the d-pad to select it. Contrary to auto-completion on the desktop, in V3, the focused phrase (highlighted with the sky blue background) is not automatically added to the textbox. This difference facilitates individual word selection, which is conveyed to users by the orange font color and underlining of an individual word that will be inserted into the textbox if they move right on the d-pad. The underlined text always matches the text-so-far up to a word boundary. For example, in Figure 4(b), even though "customer service" matches "cus*," only "customer" is colored in orange and underlined. To further convey to users that the colored and underlined word will be added to the textbox if they move right on a selection, we placed a right-arrow icon of the same color on the far right-hand side of the of the focused phrase (shown in Figures 4(b)-(d)). We also displayed what word would show up in the textbox as grayed-out "phantom" text. Note that in auto-completion, moving right on a selection typically serves no function. Finally, users select whole phrases in the usual way – by hitting the 'OK' button at the center of the d-pad. In short, V3 supports individual word selection in addition to the typical selection method that is afforded by auto-completion.

Notice that the sequence of word selections in Figure 4 for V3 is exactly the same as for V2 in Figure 3: After typing "cus," the user notices that the word "customer" has appeared on top (Figure 4(a) and moves down into the drop-down box (Figure 4(b)). The underlined word is then selected by moving right in Figure 4(c). Henceforth, that word is *pinned* (i.e., added to the textbox) and the rest of the suggestions all begin with the pinned text (i.e., "customer*"). Pinning is akin to setting a focus on a word in the word columns of V2 and collapsing to a focused word in V1. In Figure 4(c), the user finds and pins the next word. Seeing the intended query among the suggestions in Figure 4(d), the user simply clicks the 'OK' button and is done.

4. EVALUATION

In order to assess whether users could easily learn the different versions of the Phrase Builder user interface and leverage individual word selection to reduce keystrokes, we conducted two controlled experiments. In Experiment 1, we tested V2 against auto-completion (*AC*) as it functions on the desktop. In Experiment 2, we tested V3 against both AC and typing. For typing, we simply turned off all suggestions. In order to create the AC interface, which is shown in Figure 5, we simply disabled individual word selection and removed all associated visual cues (e.g., underlining). AC displays suggestions in exactly the same way as V3 in Figure 4(a), and uses the same sky blue background for the focused phrase. Because we expected that users would be more familiar with suggestions based on prefix matching, we also disabled infix matching (e.g., britney **spea**rs) for AC, V2, and V3. Furthermore, Phrase Builder V2 in Experiment 1 did not leverage back-off suggestions simply because we did not conceive of the techniques described in Section 2.4 until after we created V2. As such only V3 presented users with back-off suggestions.

4.1 Method

4.1.1 Participants

For Experiment 1, we recruited 18 participants (16 males and 2 females) between the ages of 19 and 52 from the Seattle metropolitan area by a professional contracting service. The average age of participants was 35.5. Participants came from a wide variety of occupational backgrounds. For Experiment 2, we recruited 12 participants (5 males and 7 females) from the same demographics as Experiment 1. The average age of participants

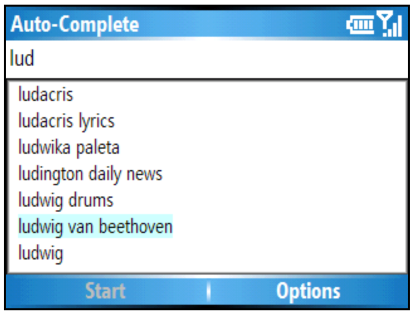


Figure 5. Screenshot of the auto-completion interface.

was 30.4. All participants were compensated for their time.

During recruiting, all participants answered that they were familiar with the QWERTY layout and could type on a normal size keyboard without frequently looking at the keys. We also tried to counter-balance the number of participants who owned a numeric keypad phone, a QWERTY keyboard phone, and a touch-only phone. For both experiments, we found that the type of phone owned by the participants was not statistically significant in predicting our dependent measures.

4.1.2 Procedure

All participants were first taught the basics of using a Blackjack smartphone. We then repeated the following procedure for each RTQE interface. We provided a short tutorial on how to use the interface and walked them through training stimuli. We presented participants with target queries on a desktop computer, which they then had to type into their mobile devices using the interface. Participants were encouraged to take as much time as necessary to look at the target queries before starting. We informed the participants that they would be timed, but that they should not sacrifice accuracy for speed. We also informed them that they always had the option of simply typing in the entire target query. Once the entire target query was entered into the textbox, with or without assistance from any RTQE interface, participants pressed a soft-key button on the Blackjack smartphone for ‘Done’ and moved on to the next item. At the end of the session, participants answered a questionnaire comparing the different interfaces. The entire session lasted about 1.5 hours.

Note that for entering the target queries, following the “unconstrained text entry evaluation paradigm” [28], we did not disable backspace and other error correction mechanisms. In Experiment 1, we gave participants the option of skipping target queries, but this occurred in only 1.8% of the data, so we did not present this option in Experiment 2.

4.1.3 Design

Our primary independent variable was *UI*. For Experiment 1, we compared V2 against AC. For Experiment 2, we compared V3 against AC and typing (our control condition). Because Phrase Builder was designed to be particularly useful when intended queries only partially matched the suggestions, as our second independent variable we examined *Query Log Presence* (*LogPresence*): whether a target query could be retrieved as a complete phrase in the search query logs (*Complete*) or only partially (*Partial*). We hypothesized that both V2 and V3 would reduce keystrokes more than AC when the *LogPresence* of the target query was *Partial* than when it was *Complete*.

In short, for Experiment 1, we conducted a 2 (*UI*) x 2 (*LogPresence*) within-subjects factorial design experiment, where participants used both RTQE interfaces in counter-balanced order for two sets of target queries (see next section). For Experiment 2, we conducted a 3 (*UI*) x 2 (*LogPresence*) experiment. We again counter-balanced the order of the three *UI* conditions.

Because our second goal for designing a mobile RTQE interface is to reduce keystrokes, we decided to directly assess *Keystrokes per Character* (*KSPC*) as our primary dependent variable, which is computed as:

$$KSPC = \frac{|IS|}{|T|}$$

where |*IS*| denotes the length of the input stream, including all d-pad keystrokes as well as backspaces, and |*T*| denotes the length of the target query [28]. Although KSPC for the different *UI* conditions could have been theoretically calculated for the stimuli, we decided to measure KSPC in an experimental setting in order to account for button-pressing mistakes, which are common on miniature QWERTY keyboard phones. In other words, we wanted to assess real performance on real devices. Assuming that no RTQE interface or any predictive text entry method is in place, participants must type every character of a target query using the QWERTY keyboard. As such, if participants make no button-pressing mistakes, the baseline KSPC for a QWERTY phone is 1.

As secondary dependent variables, we also examined the elapsed *Duration* for entering target queries as well as two accuracy metrics: *IsCorrect* measures whether the final user text matched the target query, and *MSDErrorRate* measures error rate as a function of the minimum string distance (MSD) between two strings. MSD computes the distance between two strings in terms of the lowest number of error-correction operations required to turn one string into the other (see [28] for more details). Turned into an error rate measure, *MSDErrorRate* is calculated as:

$$MSDErrorRate = \frac{MSD(T,U)}{MAX(|T|,|U|)}$$

where *T* denotes the target query, *U* denotes the user text, and *MSD* is the minimal edit distance between *T* and *U*.

4.1.4 Stimuli

In order to obtain the target queries for the experiments, we wrote a script to randomly sample queries from the search query logs of Live Search Mobile, described in Section 2.3. As discussed previously, to reduce storage space, we only encoded those queries which had at least 5K popularity hits. This produced an *encoded index* of roughly 122K queries from a *base index* of over 1 million queries. Because users do not typically use RTQE for short queries, we constrained our sampling method to select only queries of length greater than 14 characters, which was the average query length of the base index. To obtain stimuli matching the *LogPresence* conditions, we sampled *Complete* queries directly from the encoded index, and *Partial* queries from the base index excluding queries in the encoded index. Note that the *Partial* queries in this way are by definition out-of-index (i.e., not in the encoded index). In sampling the *Partial* queries, we selected only queries that contained at least one word which could be found in the vocabulary of the encoded index.

For Experiment 1, we created 2 stimuli sets of 20 target queries, and for Experiment 2, we created 3 stimuli sets of 20 target queries for the different *UI* conditions. The target queries were then randomly shuffled. Note that the average character length of the target queries between the stimuli sets was not found to be significantly different. We also created training stimuli sets of 8 (for Experiment 1) and 10 (for Experiment 2) target queries, which we repeatedly used in the experiments for the different *UI* conditions during the tutorials.

5. RESULTS

5.1 Experiment 1

5.1.1 Quantitative

In performing descriptive statistics, we immediately noticed an interesting trend. *KSPC* and *Duration* seemed to be decreasing as

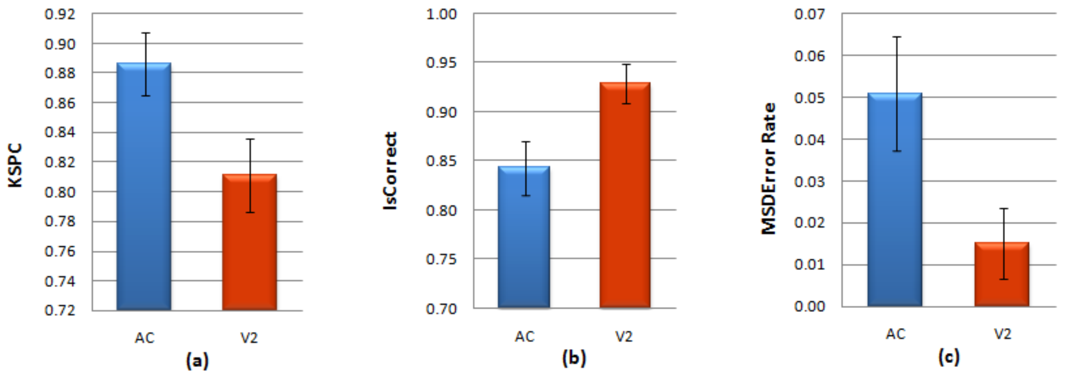


Figure 6. (a) Mean *KSPC* for AC and Phrase Builder V2. (b) Mean *IsCorrect* for AC and V2. (c) Mean *MSDErrorRate* for AC and V2. Error bars represent standard errors about the mean.

participants used V2 and became more familiar with this interface. This learning effect was not observed for AC, presumably because participants were already familiar with the interface. We conducted one-way ANOVAs to examine the relationship between our dependent variables and two independent variables, *UI* and a new variable, *ItemOrder*. In particular, we divided *ItemOrder* into *Beginning* or *End*, depending on whether a stimuli item (target query) occurred among the first 10 or the last 10 items. For *KSPC*, we found a main effect for *ItemOrder* ($F_{1,714}$=6.74, *p*<.05) and an interaction effect between *ItemOrder* and *UI* ($F_{1,714}$=4.99, *p*<.05). For *Duration*, we found a main effect for *UI* ($F_{1,714}$=16.19, *p*<.001), *ItemOrder* ($F_{1,714}$=9.35, *p*<.01), as well as an interaction effect between the two ($F_{1,714}$=6.51, *p*<.05). Because these results imply that there was a learning effect for V2, we decided to utilize data from only the last 10 items of each stimuli set in order to explore how well V2 could reduce keystrokes once users became accustomed to the interface.

For our primary dependent variable, *KSPC*, we found a significant main effect for *UI* ($F_{1,192}$=6.13, *p*<.05), with V2 exhibiting lower KSPC than AC, as shown in Figure 6(a). We also found a significant interaction effect between *UI* and *LogPresence* ($F_{1,192}$=31.16, *p*<.001). In particular, AC displayed lower KSPC (μ=.67, σ=.03) than V2 (μ=.77, σ=.03) for *Complete* queries, most likely because V2 requires participants to select every word in the target query, in contrast to the full query selection method of AC (and V3, as we discuss later). On the other hand, just as we hypothesized, for *Partial* queries, V2 (μ=.86, σ=.04) had lower KSPC than AC (μ=1.10, σ=.03), which, being greater than 1.0, was as bad as typing with occasional errors – this finding is consistent with [20] which found that users often ignored the benefit of accepting a suggestion, presumably because it requires less cognitive load.

Among our secondary dependent variables, although we did not find a significant main effect of *UI* on *Duration*, we did find main effects for the accuracy metrics *IsCorrect* ($F_{1,192}$=6.83, *p*<.05) and *MSDErrorRate* ($F_{1,192}$=5.66, p<.05). Overall, V2 clearly stood out for its higher *IsCorrect* accuracy (Figure 6(b)) and lower MSDErrorRate (Figure 6(c)).

5.1.2 Qualitative

After participants finished the study, we asked them to decide which interface they preferred overall. 8 participants chose V2 while 10 participants chose AC. When asked which interface they perceived to be the fastest, 13 out of 18 participants answered AC, despite the fact that V2 had lower KSPC and there was no significant difference in *Duration*. In follow-up discussions, we learned that participants who preferred AC did so primarily because of familiarity and perceived ease-of-use (which is related to familiarity). On the other hand, participants who preferred V2 did so because it "seemed to require less typing." Some participants even mentioned that they thought they would be more efficient over time with V2. The qualitative feedback we received from Experiment 1 motivated us to the design V3 as a marriage between V2 and AC. In short, we aimed to capture the familiarity and ease-of-use of AC but with the extra functionality of individual word selection.

5.1.3 Discussion of Experiment 1

Although V2 demonstrated lower KSPC than AC, and allowed for greater accuracy in terms of both *IsCorrect* and *MSDErrorRate*, it had two major flaws: 1) participants did not overwhelmingly prefer V2, due primarily to their greater familiarity with AC, and 2) AC exhibited lower KSPC for *Complete* queries. Although the first flaw was reason enough for us to re-design V2, the second flaw gave us concern. This is because in our experiment, we had an equal number of *Complete* and *Partial* target queries, but in real usage, it is likely that users will be intending mostly *Complete* queries – since that is how these queries become popular. As such, in formulating a new design for V3, we needed to make sure that Phrase Builder was as good as AC on *Complete* queries.

5.2 Experiment 2

5.2.1 Quantitative

Having experienced an *ItemOrder* effect in Experiment 1, we decided to spend more time teaching participants about the different interfaces in our tutorials. Hence, we slightly increased the number of training stimuli from 8 to 10 target queries. As a result, for Experiment 2, we did not find any statistically significant effect of *ItemOrder* on our dependent variables nor did we find any significant interaction effects with *UI*, as we did in Experiment 1.

For our primary dependent variable, KSPC, we again found a significant main effect for *UI* ($F_{1.76,205.7}$=84.94, *p*<.001)[3] with V3 (μ=.80, σ=.02) exhibiting lower KSPC than both AC (μ=.93, σ=.02) and typing (μ=1.12, σ=.02). Post-hoc, pairwise

[3] Because the sphericity assumption had been violated for UI (χ^2(2)=17.17, *p*<.001), we corrected the degrees of freedom (1.76, 205.7) using the Greenhouse-Geisser estimates (ε=.88).

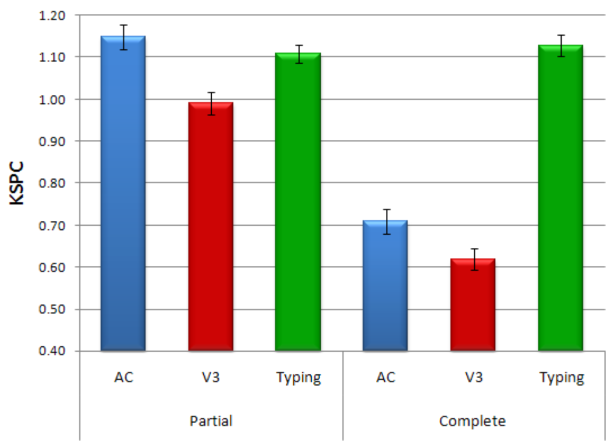


Figure 7. Mean *KSPC* for AC, Phrase Builder V3, and typing separated into *Partial* and *Complete* queries (*LogPresence*). Error bars represent standard errors about the mean.

comparisons revealed that all three of the *UI* conditions were significantly different from each other (all *p*<.001). We also found a main effect for *LogPresence* ($F_{1,117}$=152.58, *p*<.001), as well as an interaction effect with *UI* ($F_{2,234}$=49.25, *p*<.001). Figure 7 shows the mean KSPC for each user interface broken down by whether the *LogPresence* was *Complete* or *Partial*. As expected, typing, our baseline, showed no significant difference in KSPC between *Complete* and *Partial* queries. Interestingly, AC did not show significantly lower KSPC than typing for *Partial* queries, but V3 did (*p*<.001). We were pleased to see that for even *Complete* queries, V3 exhibited lower KSPC than AC (*p*<.05).

Finally, unlike Experiment 1, we did not find any significant effects for our secondary dependent variables. In particular, we lost our previous main effects for *IsCorrect* and *MSDErrorRate*.

5.2.2 Qualitative

After participants finished the study, we asked them to rank the interfaces in order of preference. 10 out of 12 participants rated V3 as their top choice. V3 was significantly preferred over AC, as revealed in a non-parametric Kruskal-Wallis test ($\chi^2(2)$=16.29, *p*<.001) and a follow-up pair-wise comparison. We also asked participants to rank the interfaces in terms of which they perceived to be the fastest. In 10 out 12 cases, V3 was rated as the fastest. The difference between V3 and AC was again significant ($\chi^2(2)$=19.20, p<.001). During discussions, the most common reason participants gave for their preference was that V3 seemed just like AC except more efficient.

5.2.3 Discussion of Experiment 2

In designing V3, we deliberately assayed to capture the familiarity of AC. In fact, if the user never moves right on the d-pad to pin words, the functionality of V3 is equivalent to AC. However, V3 has more functionality than AC; it supports individual word selection to facilitate completion of *Partial* queries. Because participants recognized the added benefit of V3, they ranked it higher in user preference to AC. Interestingly, perceived V3 to be faster than AC, despite that fact that there was no statistically significant difference in *Duration* between the two.

In terms of KSPC, V3 outperformed AC as well as the typing baseline. For *Partial* queries, we hypothesized that this would be the case. However, for *Complete* queries, the difference came as a pleasant surprise. One possible explanation is that by underlining the currently focused word, V3 may be assisting recognition of target queries when they appear among the suggestions. Overall, the difference between AC and V3 was greater for *Partial* than *Complete* queries. This may have to do with the availability of back-off suggestions, though teasing apart the effect of back-off suggestions requires a separate study. We consider this future research.

5.3 Discussion

Although both V2 and V3 significantly reduced keystrokes, as measured by KSPC, participants perceived V3 to be faster than AC but not V2. The reason for this perception is unclear, though it may relate to participants realizing that V3 enables the same kind of interaction as AC except with more functionality – particularly for *Partial* queries.

Having found significant main effects for the two accuracy measures, *IsCorrect* and *MSDErrorRate*, in Experiment 1 but not in Experiment 2, we investigated how this might have happened. Interestingly, for Experiment 2, we noticed that participants in general made very few errors. In fact, all three interfaces had mean accuracies between the ranges of .94 and .95, which is quite high. With longer tutorials, more stimuli, and a similar look-and-feel between AC and V3, participants may have just received more practice to get better at using AC on the mobile device. Another possible explanation relates to age. The average and median ages for Experiment 1 were 36 and 35 respectively whereas for Experiment 2, they were 30 and 26. It could be that we had younger participants in Experiment 2 who were more accustomed to typing on mobile devices, and as such made less errors.

6. Conclusion and Future Directions

In this paper, we introduced Phrase Builder, an RTQE interface that reduces keystrokes by facilitating the selection of individual words in addition to whole phrases, and by leveraging back-off query techniques to offer suggestions for out-of-index queries. We described how we implemented a small memory footprint index and retrieval algorithm as well as the back-off suggestions, and then discussed lessons learned from three versions of the Phrase Builder user interface.

Ultimately, for the purposes of productizing an easy-to-use mobile RTQE interface that can reduce keystrokes, we settled on V3. Although it is possible that V2 may facilitate higher accuracy for entering intended queries, the lack of familiarity with the interface and subsequently lack of user preference deterred us from productizing it. However, V2 may have potential outside of RTQE as a general text entry tool. Finally, although V1 had a dynamic user interface for browsing search query logs, users found it unhelpful for entering intended queries. Note that the final product version of Phrase Builder (Figure 1) is not identical to V3; in particular, the phantom text feature of V3 was removed due to lack of testing resources.

With regards to future research, one pressing direction is to conduct a more thorough analysis of the benefits of our back-off query techniques for generating relevant out-of-index suggestions that are in-vocabulary. As discussed previously, incorporating semantic and syntactic information in selecting words to replace with wildcards for generating back-off queries also seems promising.

With regards to the user interface, because our experiments were conducted in a laboratory setting, it will be interesting to see how users perceive the shipped Phrase Builder in real mobile scenarios. Finally, we plan to investigate how to best customize Phrase Builder for touch-only mobile devices.

7. REFERENCES

[1] Beaulieu, M. 1997. Experiments with interfaces to sup-port query expansion. *Jour. of Documentation*, 53(1), 8-19.

[2] Beaulieu, M., Do. T., Payne, A., & Jones, S. 1997. ENQUIRE Okapi Project. *British Library Research and Innovation Report 17*.

[3] Bentley, L. 1975. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9), 509-517.

[4] Church, K. & Smyth, B., 2007. Mobile content enrichment. *Proc. of IUI*, 112-121.

[5] Church, K., Smyth, B., Cotter, P. & Bradley, K. 2007. Mobile information access: A study of emerging search behavior on the mobile Internet. *ACM Transactions on the Web*, 1(1), 1-38.

[6] Church, K., Thiesson, B., & Ragno, R. 2007. K-best suffix arrays. *Proc. of NAACL-HLT*, companion volume, 17-20.

[7] Church, K., Keane, M.T., & Smyth, B. 2005. Towards more intelligent mobile search. *Proc. of IJCAI*, 1675-1676.

[8] Church, K. & Thiesson, B. 2005. The Wild Thing! *Proc. of ACL*, 93-96.

[9] Croft, W.B. & Thompson, R.H. 1987. I3R: A new approach to the design of document retrieval systems. *Jour. of the American Society for Information Science*, 38(6), 389-404.

[10] Cui, H., Wen, R.R., Nie, J.Y. & Ma, W. 2002. Probabilistic query expansion using query logs. *Proc. of WWW*, 325-332.

[11] Efthimiadis, E.N. 1996. Query expansion. *Annual Review of Information Systems and Technology*, 31, 121-187.

[12] Fowkes, H. & Beaulieu, M. 2000. Interactive searching behavior: Okapi experiment for TREC-8. *Proc. of the IRSG 2000 Colloquium on IR Research*.

[13] http://www.google.com

[14] http://www.piccolo2d.org

[15] http://www.t9.com

[16] http://www.yahoo.com

[17] Ipsos Insight. 2006. Mobile phones could soon rival the PC as world's dominant Internet platform. http://www.ipsosna.com/news/pressrelease.cfm?id=3049, April 2006. Accessed June 2009.

[18] Jelinek, F. 1997. *Statistical methods for speech recognition*. Cambridge, MA: MIT Press

[19] Jones, M., Buchanan, G., & Thimbleby, H. 2002. Sorting out searching on small screen devices, *Proc. of Mobile HCI*, 81-94.

[20] Kamvar, M. & Baluja, S. 2008. Query suggestions for mobile search: Understanding usage patterns. *Proc. of CHI*, 1013-1016.

[21] Kamvar, M. & Baluja, S. 2006. A large scale study of wireless search behavior: Google mobile search. *Proc. of CHI*, 701-709.

[22] Kamvar, M. & Baluja, S. 2006. The role of context in query input: Using contexual signals to complete queries on mobile devices. *Proc. of Mobile HCI*, 405-412.

[23] Kang, T. 2008. Value share: global handset vendor financial metrics in Q1 2008. *Strategy Analytics*. Dated: 06-01-2008.

[24] Katz, S. 1987. Estimation of probabilities from sparse data for the language model component of a speech recogniser. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 35(3), 400–401.

[25] Koenenman, J. & Belkin, N.J. 1996. A case for inter-action: A study of interactive information retrieval behavior and effectiveness. *Proc. of CHI*, 205-212.

[26] Lewis, C., & Rieman, J. 1993. *Task-Centered User Interface Design: A Practical Introduction*. Distributed via anonymous ftp (ftp.cs.colorado.edu).

[27] MacKenzie, I., Kober, H., Smith, D., Jones, T. & Skepner, E. 2001 LetterWise: Prefix-based disambiguation for mobile text input. *Proc. of UIST*, 111-120.

[28] MacKenzie, I., & Tanaka-Ishii, K. 2007. *Text entry systems: Mobility, accessibility, universality*. San Francisco: Morgan Kaufmann Publishers.

[29] Manber, U. & Myers, G. 1990. Suffix arrays: A new method for on-line string searches, *Proc. of SODA*, 319-327.

[30] Masui, T. 1999. POBox: An efficient text input method for handheld and ubiquitous computers. H. Gellersen, Ed. *Lecture Notes in Computer Science*, 1707, 288-300.

[31] Paek, T., Thiesson, B., Ju, Y.C., & Lee, B. 2008. Search Vox: Leveraging multimodal refinement and partial knowledge for mobile voice search. *Proc. of UIST*, 141-150.

[32] White, R. & Marchionini, G. 2007. Examining the effectiveness of real-time query expansion. *Information Processing and Management*, 43(3), 685-704.

[33] Wigdor, D. & Balakrishnan, R. 2004. A comparison of consecutive and concurrent input text entry techniques for mobile phones. *Proc. of CHI*, 81-88.

[34] Zhang, Z. & Nasraoui, O. 2006. Mining search engine query logs for query recommendations. *Proc. of WWW*, 1039-1040.