

Search Vox: Leveraging Multimodal Refinement and Partial Knowledge for Mobile Voice Search

Tim Paek, Bo Thiesson, Y.C. Ju, Bongshin Lee

Microsoft Research

One Microsoft Way

Redmond, WA 98052

{timpaek, thiesson, yuncj, bongshin}@microsoft.com

ABSTRACT

Internet usage on mobile devices continues to grow as users seek anytime, anywhere access to information. Because users frequently search for businesses, directory assistance has been the focus of many voice search applications utilizing speech as the primary input modality. Unfortunately, mobile settings often contain noise which degrades performance. As such, we present Search Vox, a mobile search interface that not only facilitates touch and text refinement whenever speech fails, but also allows users to assist the recognizer via text hints. Search Vox can also take advantage of any partial knowledge users may have about the business listing by letting them express their uncertainty in an intuitive way using verbal wildcards. In simulation experiments conducted on real voice search data, leveraging multimodal refinement resulted in a 28% relative reduction in error rate. Providing text hints along with the spoken utterance resulted in even greater relative reduction, with dramatic gains in recovery for each additional character.

ACM Classification: H5.2 [Information interfaces and presentation]: User Interfaces. - Graphical user interfaces, Input Devices and Strategies, Interaction styles

General terms: Design, Human Factors, Performance

Keywords: Multimodal, speech recognition, mobile search

INTRODUCTION

According to market research, mobile devices are now poised to rival desktop and laptop PCs as the dominant Internet platform [4], providing users with anytime, anywhere access to information. One common request for information is the telephone number or address of local businesses. Because perusing a large index of business listings can be a cumbersome affair using existing mobile text and touch input mechanisms, directory assistance has been the focus of *voice search* applications, which utilize speech as the primary input modality [19]. Unfortunately, mobile environments pose problems for speech recognition, even for

native speakers [9]. First, mobile settings often contain non-stationary noise which cannot be easily cancelled. Second, speakers tend to adapt to surrounding noise in acoustically unhelpful ways [9]. Under such adverse conditions, task completion for voice search is less than stellar, especially in the absence of an effective correction user interface for dealing with speech recognition errors.

In light of the challenges of mobile voice search, we present *Search Vox* (Figure 1), a multimodal interface that tightly couples speech with touch and text in two directions; users can not only use touch and text to refine their queries whenever speech fails, but they can also use speech whenever text entry becomes burdensome. We facilitate this tight coupling through interaction techniques that leverage *wildcard queries*; that is, search queries that utilize wildcards (*) to match zero or more characters. Wildcard queries allow users to take advantage of any *partial knowledge* they may have about the words in the business listing. For example, a user may only remember that the listing starts with a word beginning with “s” and also contains “avenue”. Likewise, the user may only remember “saks something”, where “something” is used to express uncertainty about what words follow.

The contributions of this paper are threefold. First, we present the Search Vox user interface, highlighting interac-



Figure 1: Search Vox tightly couples speech with touch and text for multimodal refinement.

Keep this space free for the ACM copyright notice.

tion techniques that demonstrate its multimodal refinement capabilities and support of partial knowledge for mobile voice search. Second, we describe the overall system architecture and control flow, elucidating how we implemented the interaction techniques. Third, we evaluate Search Vox by conducting simulation experiments examining the effectiveness of its interaction techniques in recovering from speech errors on utterances collected from a previously deployed multimodal voice search product.

USER INTERFACE

Automated directory assistance (ADA), where users can request telephone or address information of residential and business listings using speech recognition, is a profitable industry with over 30 million U.S. callers per month [17]. Many voice search applications focus exclusively on telephony-based ADA. However, recent applications, such as *Live Search Mobile* [7] and *Yahoo! oneSearch* [18], have moved onto mobile devices, providing users with a richer client experience which includes, among other services, maps and driving directions in addition to ADA. Because mobile devices have at least a small graphical display, these applications display a list of possible choices for speech input whenever recognition is less than perfect. This list derives from word or phrase-level hypotheses in the recognizer, and is often referred to as an *n-best list*. The *n-best list* as a correction mechanism [1] has been around for several decades, most notably in dictation systems. The advantage of conducting ADA on a mobile device with a GUI is that the application can simply display an *n-best list* to the user for disambiguation instead of engaging the user in a prolonged confirmation dialogue about what was said.

For the sake of familiarity, in designing the Search Vox user interface, we decided to display an *n-best list* to the user whenever recognition is less than perfect, making the interface (Figure 1) appear, at least at first blush, like any other voice search application. However, because re-speaking does not generally increase the likelihood that the utterance will be recognized correctly [16], and furthermore, because mobile usage poses distinct challenges not encountered in desktop settings [12], the interface endows users with a larger arsenal of recovery strategies than just selecting from an *n-best list*.

In this section, we highlight three novel interaction techniques that demonstrate two concepts: first, tight coupling of speech with touch and text, so that whenever one of the three modalities fails or becomes burdensome, users can switch to another modality in a complementary way; and second, leveraging of any partial knowledge a user may have about the constituent words of their intended listing.

1. Word palette

When speech recognition fails to correctly identify the listing the user wants, it is sometimes the case that part of the listing shows up among different choices in the *n-best list* (in the last section, we even evaluate how often this happened with a deployed application). Unfortunately, the typical *n-best list* only allows users to select an entire recognized phrase, and not the words of the phrase. In Search Vox, the *n-best list* is transformed into a *word palette* from which users can compose and refine their queries. Figure 2 shows how users can leverage the word palette for multimodal refinement.

Suppose a user utters “first mutual bank” (Figure 2a). The system returns an *n-best list* that unfortunately does not include the intended utterance. However, it does include parts of the utterance in the choice “2. source mutual bank”. As such, the user can now select the word “mutual” (Figure 2b) and then “bank” (Figure 2c) which gets added to the query textbox in the order selected. In essence, the textbox functions as a scratch pad upon which users can add and edit words until they click the Search Vox button on the top left-hand side (Figure 2d). Likewise, words in the *n-best list* effectively act as buttons to add words to the scratch pad. Finally, when the query in the textbox is submitted as search query (which we describe in more detail in discussing the Search Vox architecture later), a new result list with words matching the query highlighted in red appears. Given that the intended query is now among the list of choices, the user simply selects the choice and is finished (Figure 2e).

2. Text hints

Just in the way that users can resort to touch and text when speech fails, they can also resort to speech whenever typing becomes burdensome, or when they feel they have provided enough *text hints* for the recognizer to identify their query. Figure 3 shows how text hints can be leveraged. Here, the

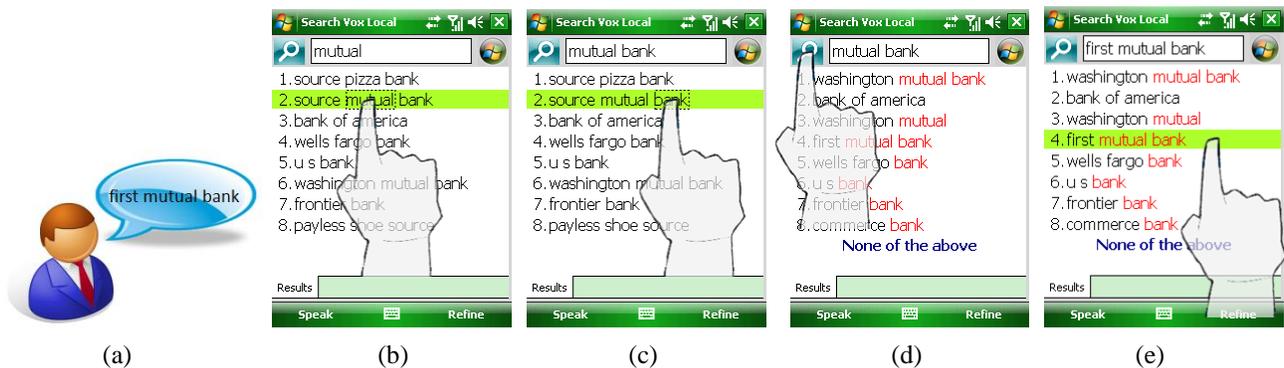


Figure 2: The word palette helps the user compose a search phrase from the *n-best list*.

user starts typing “m” for the intended query “mill creek family practice” (Figure 3a), but because the query is too long to type, the user utters the intended query after pressing the ‘Refine’ soft key button at the bottom of the screen (Figure 3b). Text hints essentially constrain the recognition results to match the query; hence, all choices in the word palette now begin with the letter “m” and indeed include the desired user utterance (Figure 3c).

Currently, Search Vox achieves this functionality by first converting the text hint in the textbox into a wildcard query and then using that query to filter the n-best list as well as to retrieve additional text matches. In principle, we acknowledge that the query should be used to bias the recognition of the utterance in the speech engine itself, as opposed to doing this as a post-process on the n-best list. We consider this future research.

3. Verbal wildcards

Sometimes users may not remember exactly the name of the listing they are looking for, but only parts of it. For example, they may remember that the first word is “pacific” and some word thereafter starts with an “n”. In Figure 4, the user is looking for “black angus restaurant” but only remembers that the first word is “black”. Here, the user can simply say, “black something restaurant” (Figure 4a), where “something” is used as a *verbal wildcard* or placeholder indicating uncertainty about the word(s) that belong in that position. In ADA call logs, users have been known to use verbal wildcards to explicitly express their uncertainty

about portions of their query [13]. Noticing that there is no “black something restaurant” in the listings, Search Vox converts the utterance into a wildcard query and returns matches (Figure 4b). Now, the desired listing appears among the choices and the user simply selects the appropriate choice and is finished (Figure 4c).

In order to support the recognition of verbal wildcards, we adjusted the statistical language model to allow for transitions to the word “something” before and after every word in the training sentences as a bigram. Business listings that actually contain the word “something” were far and few, and appropriately tagged semantically to not generate a wildcard during recognition. We also transformed the training sentences into one character prefixes so that we could support partial knowledge queries such as, “b something angus” for “b* angus” (Figure 4d-e). We describe our language modeling techniques in greater detail in [13].

Related research

The Search Vox interface belongs to a long tradition of “taming” speech recognition errors with a multimodal interface (see [11] for an overview). In coining the concept of a recognition-based interface designed for imperfect input, Rhyne and Wolf [14] were the first researchers to investigate the benefits of switching to other modalities for correction. As Oviatt and van Gent [9] showed using a Wizard-of-Oz simulation study, users naturally switch to other modalities for speech correction if given the possibility. Likewise, Suhm et al. [16] demonstrated how user choice between

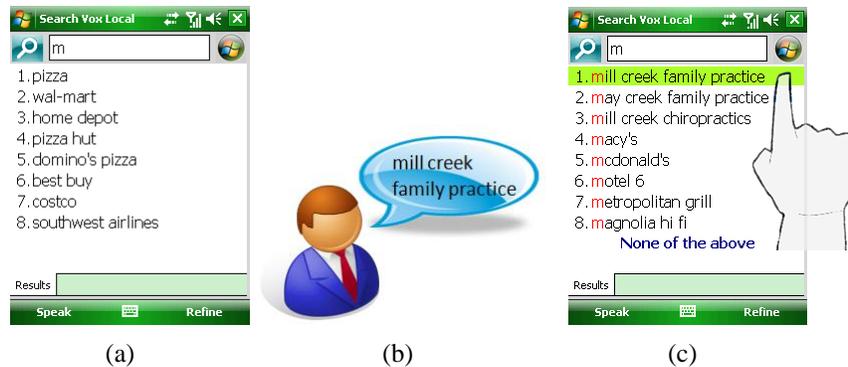


Figure 3: Text hints helps the speech recognizer better identify the query.



Figure 4: The user can specify uncertain information using the word “something”.

modalities in a dictation task is largely determined by recognition accuracy. In their desktop interface for correcting dictation errors, users could not only switch to verbal spelling or handwriting, but also edit errors using pen-based gestures. Although Search Vox was designed specifically with mobile voice search in mind, in certain situations it may make sense to exploit richer gestures other than simply selecting via touch or d-pad. For example, users could use gestures to separate words that they want in their query from those they wish to exclude. With Speech Pad, Hsu et al. [3] ported dictation to the mobile device and allowed users to correct mistakes by selecting a misrecognized word and choosing from among a list of alternates. What was unique about their implementation was that instead of just showing an n-best list of word-level alternates from the recognizer, they used an n-gram language model to suggest additional words, given the surrounding context. As we discuss in the next section, this is similar to the way Search Vox architecture supplements an n-best list with backend matches.

Apart from switching modalities, a fair amount of research has been devoted to simultaneous multimodal disambiguation. Oviatt [10] demonstrated a multimodal architecture for the QuickSet system that could fuse simultaneous pen and voice input for mutual disambiguation. Acknowledging the distinct challenges of mobile speech recognition, Oviatt [12] then evaluated QuickSet in noisy mobile settings and found that error rate could be substantially reduced by 19-35%. In Search Vox, text hints could be construed as a way of fusing speech and text, though technically, the text would have to bias the internal processing of the speech recognizer, which it currently does not.

ARCHITECTURE

What allows the Search Vox user interface to engage in multimodal refinement via the word palette and text hints, and to leverage any partial knowledge users may have about their queries via verbal wildcards is a powerful backend search engine that accepts wildcard queries. In this section, we outline the overall architectural design and control flow of Search Vox, comparing it against typical voice search. We also delve into each of the architectural components and describe their role in supporting the interaction techniques we introduced in the previous section.

Typical voice search architecture

With voice search, whether users utilize telephony or a mobile data channel to recognize utterances, the speech recognition task is always dispatched to speech servers, due to the fact that decoding utterances for large domains with many choices (i.e., high perplexity domains) requires sufficient computational power, which to date does not exist on mobile devices. Furthermore, because there are over 18 million listings in the US Yellow Pages alone, and users frequently do not use the exact name of the listing as found in the directory (e.g., “Maggiano’s Italian Restaurant” instead of “Maggiano’s Little Italy”), grammar-based recognition approaches that rely on lists fail to scale properly. As

such, recent approaches to ADA have focused on combining speech recognition with information retrieval techniques [19].

Figure 5 displays the typical architecture for voice search applications. First, an utterance is recognized using an *n*-gram statistical language model (SLM) [5] that compresses and generalizes across training sentences. In the case of ADA, the training sentences comprise not only the exact listings and business categories but also alternative expressions for those listings. Because an n-gram is based on word collocation probabilities, the output of the recognizer is an n-best list containing phrases that may or may not match any of the training sentences. This is acceptable if the phrases are submitted to an information retrieval (IR) Engine that utilizes techniques which treat the phrases as just bags of words. Of course, the IR Engine itself retrieves matches from an index, which is typically a subset of the language model training sentences, such as the exact listings along with their categories. Hence, if an utterance is recognized with high confidence, it is immediately sent to the IR Engine to retrieve the best matching listing. However, if an utterance is ambiguous in any way, as indicated for example by medium to low confidence scores, voice search applications with a user interface often present the n-best list to users for selection, at which point users can either pick a choice or retry their utterance.

Search Vox architecture

In the typical voice search architecture, the IR Engine exists on a server. This is acceptable if the user interface does not demand real-time search. With Search Vox, however, we sought to enable dynamic interactions with the IR Engine via the word palette. Because simple pings to an http server over the data channel can take over 2 seconds, which we mitigated the value of multimodal refinement, we decided to perform client-side search.

Figure 6 presents the Search Vox architecture. Notice that the IR Engine no longer sits on the server but is now part of the Search Vox application along with a RegEx Engine. For client-side search, we were able to achieve response latencies of less than 1 second on an HTC Touch device running Windows Mobile Professional 6.0 with a 201 MHz processor and 64 MB of RAM. We now discuss how we implemented both the RegEx Engine and the IR Engine on the device.

RegEx Engine. In order to facilitate both multimodal refinement and leveraging of partial knowledge, Search Vox relies on fast and efficient search of business listings using wildcard queries. Because Search Vox always displays a list of choices that are rank-ordered, and not just a single best match for a wildcard query, we decided to utilize k-best suffix arrays as our primary data structure (see [2] for more details). Similar to traditional suffix arrays [3], k-best suffix arrays arrange all suffixes of the listings into an array. However, k-best suffix arrays arrange the suffixes according to two alternating orders – the usual lexicographical ordering plus an ordering based on a figure of merit

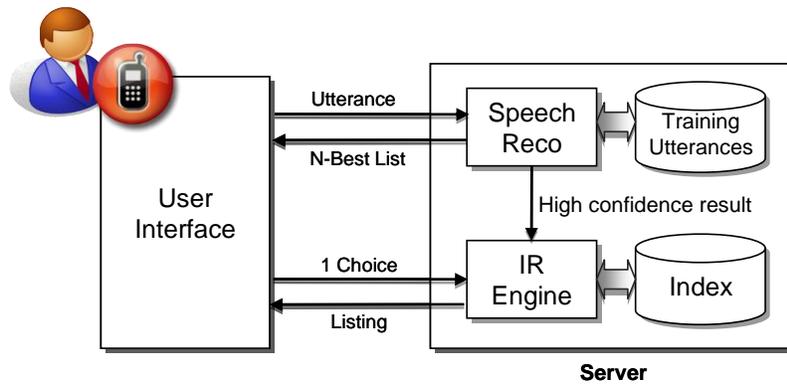


Figure 5: Typical voice search architecture and control flow.

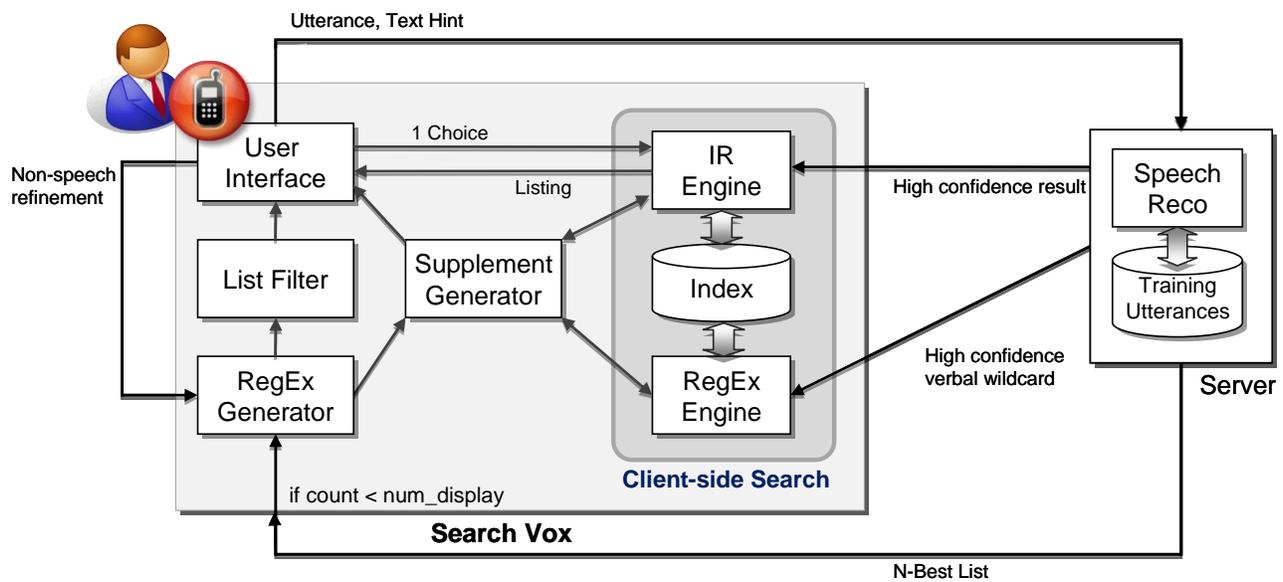


Figure 6: Search Vox architecture and control flow showing server and client-side components.

such as popularity. Because the k-best suffix array is sorted by both lexicographic order and popularity, it is a convenient data structure for finding k-most popular matches for a substring, as used in wildcard queries.

Because performing client-side search on a mobile device imposes tight constraints on memory usage, we need the RegEx Engine to be 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 characters in the listings [2]. In contrast, a standard suffix array finds *all* matches to a substring in $O(\log N)$ time [3], but does not order the matches by popularity.

Consider a simple example which explains why this subtle difference is important to our RegEx Engine. The standard suffix array may be sufficiently fast when searching for the k-best matches to a large substring since there will not be many matches to traverse in this case. However, the situation is completely different for short substrings, such as the

text hint “a”. In this case, we would have to traverse all dictionary entries containing an “a”, which is not much better than traversing all suffixes in the listings – in $O(N)$ time.

IR Engine. Besides wildcard queries, which provide *exact matches* to the listings, it is also important to retrieve *approximate matches* to the listings. For this purpose, we implemented an IR Engine based on an improved term frequency – inverse document frequency (TFIDF) [14] algorithm, which is fully described in [19]. To improve efficiency, the algorithm makes a few simplifications that fit ADA which allows it to employ an A*-like search algorithm for fast retrieval.

What is important to note about the IR Engine is that it treats queries and listings as simply bags of words. This is advantageous when users either incorrectly remember the order of words in a listing, or add additional words that do not actually appear. This is not the case for the RegEx Engine where order and the presence of substrings in the query matter.

Control flow

In order to explain how we utilize efficient client-side search using the RegEx Engine and the IR Engine, we now walk through the control flow of the interaction techniques, detailing the operation of the different components in Figure 6.

Word palette. Suppose a user states “home depot”, but because of noise, the recognizer returns an n-best list containing the phrases “home department” and “home office design”. The first step Search Vox takes in processing the n-best list is to see if the number of items in the n-best list is less than *num_display*, the number of readable items that can be displayed within the screen real-estate of the device. If the number of n-best list items exceeds *num_display*, then Search Vox displays the top *num_display* items from the n-best list in its word palette. Otherwise, Search Vox attempts to fill up the word palette with supplemental results until *num_display* items are obtained, instead of wasting the screen real estate with empty space.

As shown in Figure 6, supplementary results are generated by passing the n-best list to the *RegEx Generator*, which generates a wildcard query from the n-best list. This is done by using minimal edit distance [6] (with equal costs for the edit operation) to align the phrases at the word level. For the current “home depot” example, the RegEx Generator would first align “home” in both n-best list items and then align “depot” with “design”. Once the words are aligned, minimal edit distance is again applied to align the characters. Whenever there is disagreement between any aligned words or characters, a wildcard is substituted in its place. Hence, for the n-best list for “home depot”, the RegEx Generator would create the wildcard query “home * de*”. Note that the RegEx Generator also applies a few simple heuristics to clean up the query (e.g., no word can have more than one wildcard, and consecutive wildcards are collapsed into one).

With a wildcard query generated from the n-best list, the *Supplement Generator* in Figure 6 retrieves supplementary results first from the RegEx Engine using the wildcard query, and then from the IR Engine using the top n-best list item as a query. We recognize that the question of how best to mix exact matches from the RegEx Engine and approximate matches from the IR Engine remains to be settled. We consider this future research.

For now, everything we have been discussing relates to obtaining the first word palette from speech input (viz., “home depot”). Once the word palette with supplementary results is displayed, it is still incumbent upon the user to select words that belong to the desired query. Suppose the user just selects “home”, which gets immediately added into the textbox. Once the user hits the Search Vox button on the left of the textbox, whatever query is in the textbox is immediately sent to the RegEx Generator as a *non-speech refinement*, as shown in Figure 6. The RegEx generator transforms the textbox query into wildcard query by inserting wildcards before any white spaces, as well as to the end

of the entire query. For example, for the refinement “home”, the generator would produce the wildcard query “home*”. This gets passed again to the Supplement Generator which first retrieves results from the RegEx Engine. Given that there is no speech, the Supplement Generator has no top n-best list result to submit as a query to the IR Engine. In this case, the Supplement Generator simply uses the exact matches from the RegEx Engine as its queries.

Text hints. In order to support text hints, the Search Vox architecture utilizes a *List Filter* as follows: After an n-best list is obtained from the speech recognizer, Search Vox first checks to see if there is any text hint in the textbox. If not, the same control flow for the word palette applies. If so, the RegEx Generator creates a wildcard query from the text hint, treating the text hint as a non-speech refinement. Once a wildcard query is generated from the text hint, the List Filter uses it to remove any n-best list items that do not match the query. For example, if the user were to put in “h d” as a text hint for “home depot”, the RegEx Generator would produce “h* d*” as a wildcard query, and the List Filter would allow both “home department” and “home office design” to pass. If the n-best list also contained the phrase “home supply”, that would be filtered by the wildcard query. After Search Vox applies the List Filter to the n-best list, the same wildcard query is used to obtain supplementary results from the Supplement Generator, if necessary.

Verbal wildcards. In order to support verbal wildcards, as discussed earlier, changes need to be made to the language models of the speech server [13]. If the top recognition result contains a verbal wildcard with high-confidence, as shown in Figure 6, it is immediately sent to the RegEx Engine and used to retrieve exact matches. If supplementary results are required, as before, the Supplement Generator submits the exact matches to the IR Engine as queries. If the top result is not recognized with high confidence, the n-best list is passed to the RegEx Generator to derive a new set of results, unlike the typical case in which the n-best list is automatically presented to the user. This is due to the fact that showing wildcards to users in the word palette is likely to confuse them.

With the language modeling changes made to the speech server, it is possible for n-best list items to contain wildcards. As such, it is important to filter the n-best list to remove any wildcards. In practice, unless the user explicitly uses a verbal wildcard, it is very unlikely to show up in the n-best list, at least with our language modeling changes [13]. Indeed, using our changes resulted in an insignificant 0.5% loss in performance on all non-verbal wildcard utterances.

EVALUATION

In order to assess the effectiveness of the Search Vox interface in recovering from speech recognition errors, we conducted simulation experiment on utterances collected from a deployed mobile voice search product; namely, Microsoft Live Search Mobile [7], which provides not only ADA but

Case	Frequency	Percentage
Top 1 High Conf (Bull's Eye)	545	24%
Top 1 Med + Low Conf	1125	48%
Top N	183	8%
All Wrong	464	20%
Total:	2317	

Table 1. Breakdown of the simulation test data.

also maps, driving directions, movie times and local gas prices. Besides capturing the difficult acoustic conditions inherent in mobile environments, the collected utterances also represent a random sampling of speaker accents, speaker adaptation to surrounding noise, and even the variable recording quality of different mobile devices.

We obtained 2317 local-area utterances which had been transcribed by a professional transcription service and filtered to remove noise-only and yes-no utterances. The utterances were systematically collected to cover all days of the week as well as times in the day. For all of our simulation experiments, the utterances were submitted to a speech server which utilized the same acoustic and language models as Live Search Mobile [7]. Of the 2317 utterances, the transcription appeared in the top position of the n-best list 72.0% of the time, and somewhere in the n-best list 80.0% of the time, where again n was limited to 8 (the number of readable choices we could display on the Pocket PC form factor). As summarized in Table 1, in 20% of the utterances, the transcription did not appear at all in the n-best list. These failure cases constituted our opportunity for recovering from error, given that Search Vox performs the same as the existing product for the other 80% of the cases.

Experiments

Word palette

Looking at just the failure cases, the first set of experiments we conducted examined how much recovery rate could be gained by treating the n-best list as a word palette and allowing users to select words. Although the interface itself enables users to also edit their queries by inserting or deleting characters, we did not allow this for the experiments. Words in the n-best list that matched the transcription were always selected in the proper word order. For example, if the transcription was “black angus restaurant”, we selected “black” from the n-best list first before selecting either “angus” or “restaurant”. Furthermore, we selected as many words from the transcription as we could find in the word palette. For instance, although we could have submitted just “black” as a query in the previous example, because “angus” could also be found in the n-best list, we included it.

As shown in Figure 7, words in the n-best list alone (without supplementary matches from the backend) covered the full transcription 4.31% of the time. Note that full coverage constitutes recovery from the error since the transcription can be completely built up word by word from the n-best list. In 24.6% of the cases, only part of the transcription

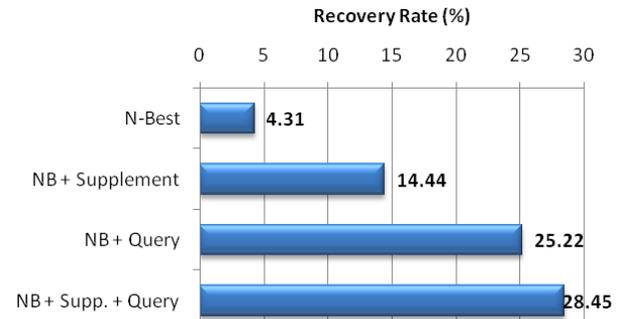


Figure 7. Recovery rates for using multimodal refinement with the word palette.

could be covered by the n-best list (not shown in Figure 7), in which case, we would need to submit another query to get a new list. If we instead supplement the n-best list with matches from our backend, we improve our recovery rate to 14.4%, which is a factor of 3.4 over using the n-best list as a word palette. For the transcriptions which were only partially covered by the n-best list, if we utilize the backend using whatever words could be found, the recovery rate jumps to 25.2%. If we use the n-best list padded by supplementary matches to submit a query, the recovery rate is 28.5% (which is also the relative error reduction with respect to the entire data set). This is 5.9 times the recovery rate of using just the n-best list as a word palette.

Text hints

Before examining the effect of providing text hints on the recovery rate, as a baseline, we first considered how well Search Vox could recover from an error by just retrieving the top 8 most popular listings from the index. This is shown in Figure 8 in the 0 character column. Surprisingly, guessing the top 8 most popular listings resulted in a recovery rate of 14.4%. Figure 3a shows these listings as a default list when starting Search Vox, which includes the general category “pizza” as well as popular stores such as “walmart” and “home depot”. We discuss the implication of such a high baseline in the next section.

In applying text hints for the experiment, we used a simple left-to-right assignment of prefixes for generating a wildcard query that proceeded as follows: Given m characters to assign, we would assign a character to the prefix of each word in the transcription. If there were still characters left to assign, we would loop back to the beginning word of the transcription and continue. For example, for a 3 character text hint, if the transcription contained three words such as “black angus restaurant”, we would assign prefix characters for all three words; namely, “b* a* r*”. If, on the other hand, the transcription was “home depot”, we would loop back to the first word so that the generated wildcard query would be “ho* d*”. After generating a wildcard query for the text hint from the transcription, we used it to filter the n-best list obtained from submitting the transcription utterance to the speech server. If we did not have enough list

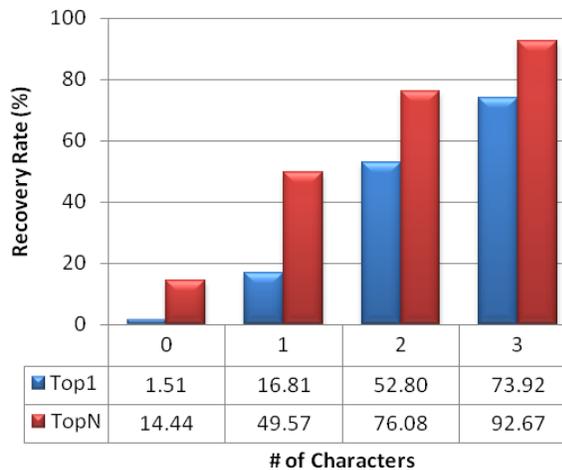


Figure 8. Recovery rates for text hints of increasing number of characters.

items after the filtering, we supplemented the list as described previously using the Supplement Generator.

When a 1-character text hint is used along with the spoken utterance, as shown in Figure 8, the recovery rate jumps to almost 50%, with 16.8% of the transcriptions appearing in the top position of the result list. That is 3.4 times better than guessing the listing using no text hints. As more and more characters are used in the text hint, the recovery rate climbs to as high as 92.7% for 3 characters.

Discussion

We were surprised to see such high recovery rates for 1 or more character text hints. While looking at the transcriptions to investigate how we were able to obtain those rates, we discovered an interesting self-reinforcement phenomenon: Users consistently asked for popular listings. As such, because our backend utilizes popularity to retrieve k-best matches, we frequently obtained the correct answer. This may be because users have found that low popularity listings do not get recognized as well as high popularity listings, so they do not even bother with those. Or it may be that popular listings are precisely popular because they get asked frequently. In any case, we hope that by providing users with a multimodal interface that facilitates a richer set of recovery strategies, they will be encouraged to try unpopular queries as well as popular ones.

With respect to verbal wildcards, in a separate paper focusing on the language modeling aspects of supporting verbal wildcards, we evaluated their effectiveness in reducing errors by conducting an experiment in which users generated both verbal wildcard queries as well as guesses (see [13] for details). In that experiment, verbal wildcards reduced relative error rate by 31.8% compared to guessing.

CONCLUSION AND FUTURE DIRECTIONS

In this paper, we presented Search Vox, a multimodal interface for mobile voice search applications that not only facilitates touch and text refinement whenever speech fails, but also allows users to assist the recognizer via text hints.

Search Vox can also take advantage of any partial knowledge users may have about their queries by letting them express their uncertainty through verbal wildcards. We also discussed the overall architecture and detailed how Search Vox could quickly retrieve exact and approximate matches using client-side search of the listings. Finally, in evaluating Search Vox via simulation experiments conducted on real mobile voice search data, we found that leveraging multimodal refinement using the word palette resulted in a 28% relative reduction in error rate. Furthermore, providing text hints along with a spoken utterance resulted in dramatic gains in recovery rate, though this should be qualified by stating that users in the test data tended to ask for popular listings which we could retrieve quickly.

For future research, we plan to conduct usability studies of the different interaction techniques we presented. We also plan to deploy Search Vox to real users and to encourage them to try finding unpopular listings. Both the Search Vox interface and the architecture that supports it have many aspects that we desire to investigate. With respect to the interface, we acknowledge that showing just the n-best list gives the affordance of providing users with feedback about the quality of the recognition. As such, we plan to explore the issue of visually distinguishing n-best list items coming from the recognizer from those phrases that are being supplemented by our Supplement Generator. Or perhaps it makes sense to only supplement results for text hints, especially given the boost in recovery rate that we observed in our experiments.

ACKNOWLEDGMENTS

The authors would like to Chris Kim for helping us generate figures.

REFERENCES

1. Ainsworth, W.A. & Pratt, S.R. 1992. Feedback strategies for error correction in speech recognition systems. *International Journal of Man-Machine Studies*, 26(6), 833-842.
2. Church, K., Thiesson, B., & Rago, R. 2007. K-best suffix arrays. *Proc. of NAACL HLT, companion volume*, 17-20.
3. Hsu, P., Mahajan, M. & Acero, A. 2005. Multimodal text entry on mobile devices. *Proc. of ASRU*.
4. Ipsos Insight. 2006. Mobile phones could soon to rival the PC as world's dominant Internet platform. <http://www.ipsosna.com/news/pressrelease.cfm?id=3049>, April 2006. Accessed January 2008.
5. Jelinek, F. 1998. *Statistical methods for speech recognition*. MIT Press.
6. Levenshtein, V. I. 1966. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady* 10:707-710.
7. Live Search Mobile: <http://livesearchmobile.com/>

8. Manber, U. & Myers, G. 1990. Suffix Arrays: A New Method for On-line String Searches, *Proc. of SODA*, 319-327.
9. Oviatt, S. & Van Gent, R. 1994. Error resolution during multimodal human-computer interaction. In *Proc. of CHI*, 415-422.
10. Oviatt, S. 1999. Mutual disambiguation of recognition errors in a multimodal architecture. In *Proc. of the International Conference on Computer-Human Interaction*, 576-583.
11. Oviatt, S. 2000. Taming recognition errors with a multimodal interface. *Communications of the ACM*, 43(9), 45-51.
12. Oviatt, S. 2000. Multimodal system processing in mobile environments. *Proc. of UIST*, 21-29.
13. Paek, T. & Ju, Y.C. 2008. Accommodating explicit user expressions of uncertainty in voice search or something like that. *Proc. of Interspeech*.
14. Rhyne, J.R. & Wolf, C.G. 1993. Recognition-based user interfaces. In *Advances in Human-Computer Interaction*, H.R. Hartson & D. Hix, Eds. Ablex Publishing Corp, 191-212.
15. Salton, G. 1983. *Introduction to modern information retrieval*. McGraw-Hill.
16. Suhm, B., Myers, B. & Waibel, A. 2001. *Multimodal error correction for speech user interfaces*. ACM TOCHI, 8(1), 60-98.
17. Tellme Press Release. 2006. Tellme to power all Cingular wireless 411 calls: Expanded relationship focuses on enhancing 411 with personalization and mobile search services, <http://www.tellme.com/about/PressRoom/release/20061009>, October 2006. Accessed March 2008.
18. Yahoo oneSearch: <http://mobile.yahoo.com/onesearch>
19. Yu, D., Ju, Y.C., Wang, Y.Y., Zweig, G., & Acero, A. 2007. Automated directory assistance system: From theory to practice. *Proc. of Interspeech*.