Voice Query Refinement

Cyril Allauzen[1]*, Edward Benson*[2]*, Ciprian Chelba*[1]*,*
Michael Riley[1]*, Johan Schalkwyk*[1]

[1]Google, Inc, 76 Ninth AV, NY, NY, USA
[2]MIT CSAIL, 32 Vassar ST, Cambridge, MA, USA

allauzen@google.com, eob@csail.mit.edu, ciprianchelba@google.com,
riley@google.com, johans@google.com

Abstract

We describe a system for the refinement of spoken search queries. Given an initial query (`Northern Italian restaurants in New York`), instead of requiring a fully-specified followup query (`Korean restaurants in New York`), a more natural, abbreviated update query (`Korean instead`) may be spoken. The system consists of a parsing step to identify the type and arguments of the refinement, a candidate generation step to enumerate the possible refinements, and a model classification step to select the best refinement. We present results on test query refinements given both to this system and to human judges that show the automated system outperforms the human judges on that data set.

Index terms: spoken dialog systems, voice search, query refinement

1. Introduction

Speech recognition technology is approaching a quality sufficient to enable a wide variety of devices to incorporate spoken natural language interfaces. While these interfaces are already appearing, they tend to be "one shot" in nature: the user issues a single voice command or input, to which the device responds. A step beyond these initial deployments is conversational interfaces: the computer having the ability to sustain a stateful natural language conversation with the user about a particular topic. Apple's Siri interface has rudimentary capabilities in this direction.

This paper examines a subset of the conversational interface problem that we call input refinement. Input refinement occurs whenever the user wishes to refine or correct some previous utterance. We look specifically at query refinement, the act of taking some initial query (e.g., `Northern Italian restaurants in New York`) and modifying that query to obtain different results (e.g., `Korean restaurants in New York`) in a search setting.

With a keyboard interface, query refinement is explicit: the user manually edits the initial query to produce the refined one, which is then sent in its entirety.

q_1	"used books"
q_2	"paperback"
q_3	$\rightarrow$ *used paperback books*
q_1	"sports clubs in Boston"
q_2	"Cambridge not Boston?"
q_3	$\rightarrow$ *sports clubs in Cambridge*
q_1	"Northern Italian restaurant"
q_2	"Korean instead"
q_3	$\rightarrow$ *Korean restaurant*

Figure 1: Example spoken query refinements

With voice-based interfaces, however, we wish to allow the user to express only the difference between the first query and the new query, permitting a briefer and more natural speech interface. Several examples of how this system might work are shown in Figure 1.

The overall problem formulation is first described in Section 2. The models used to predict the correct refined query from the voice input are presented in Section 3 and the data used to train and test the models are described in Section 4. Experimental results are presented in Section 5 and a discussion of the results concludes in Section 6.

2. Problem Formulation

We define a query refinement as a triple $\langle q_1, q_2, q_3 \rangle$ where a user issues an initial query q_1, then utters a refining phrase q_2 with the intent of producing query q_3, as in the examples in Figure 1. At serving time, the challenge is producing an appropriate q_3 given only $\langle q_1, q_2 \rangle$.

We divide the problem into two basic steps: first a *parsing* step to determine the *type* and *arguments* of the refinement specified by q_2 and then an *editing* step that applies the refinement in the appropriate place in q_1 to generate q_3. In the first example in Figure 1, q_2 specifies the word `paperback` is to be inserted, in the second example the word `Cambridge` is to be substituted for `Boston`, and in the third example the word `Korean` is to be substituted but the text to be replaced is not explicitly specified in q_2. In general, q_2 can be classified into one of the refinement types

Northern Italian	restaurant
Korean	restaurant

(a)

Northern	*Italian*	restaurant
Nortern	*Korean*	restaurant

(b)

Northern Italian	*restaurant*
Northern Italian	*Korean*

(c)

Figure 2: Example candidate alignments. The top row is the initial query q_1, while the bottom row are candidate refined queries q_3.

$\mathcal{T} = \{\texttt{insert}, \texttt{delete}, \texttt{substitute}, \texttt{new}\}$ where `new` means that q_2 is a new search unrelated to the previous query. Further in the substitution case, either the text to be replaced is specified in q_2 or not.

Thus the parsing step consists of converting the voice input $\langle q_1, q_2 \rangle$ into a parse (q_1, s, τ, r) where s specifies the the text to be inserted, deleted or substituted, $\tau \in \mathcal{T}$, and r is the text to be replaced when specified in q_2 or is ϵ otherwise. When $r = \epsilon$, we may write the parse as (q_1, s, τ). How we perform the parsing step is described in Section 3.1.

Given the result of the parsing step, the editing step needs to be performed on q_1 to produce q_3. This itself can be divided into two steps. First, all possible (or plausible) edits are generated. A possible edit is specified by an alignment between q_1 and s and the refinement type. This alignment can then be extended to one between q_1 and a candidate q_3 by preserving the unmatched words in q_1 in q_3. For example, Figure 2 shows several possible alignments for the third example in Figure 1. The italic text shows how the substituted text s is matched in q_1, while the unmatched text in q_1 is otherwise preserved to form an alignment between q_1 and a candidate q_3. Note a multi-word phrase may need to be aligned to a word or another phrase (e.g., in Figure 2(a)). In Section 3.2, we describe precisely how the candidate alignments are generated. In Section 3.3 we describe data-driven approaches to scoring these edits to identify the best one.

3. Models

3.1. Query Parsing

As described in the previous section, the parsing step converts the query pair $\langle q_1, q_2 \rangle$ into a parse (q_1, s, τ, r). We will make the simplifying assumption that the parse can be performed by a context-free grammar applied to q_2. See the discussion in Section 6 for more general settings.

A simple CFG grammar $(\Sigma, \mathcal{N}, Q, \mathcal{P})$ with terminal symbols in Σ, non-terminal symbols $\mathcal{N} = \{Q, Ins, Del, Sub, New, S, R\}$, initial symbol Q and productions $\mathcal{P}$:

Q	→	*Ins* | *Del* | *Sub* | *New*
Ins	→	`insert` *S* | *S*
Del	→	`delete` *S*
Sub	→	*S* `not` *R* | *S* `instead`
New	→	`search for` *S*
S	→	Σ | Σ *S* / δ
R	→	Σ | Σ *R* / δ

cover the examples in Figure 1. Clearly, a parse tree of q_2 can be used to determine (q_1, s, τ, r). In an ambiguous case, the weighted rules (with weight δ) can be used to penalize parses with fewer non-terminals, so the most detailed parse can be selected. In our actual system, many more productions are added to increase the grammar's converage.

If the parse determines that q_2 is a new search, it is issued. Otherwise, the editing step is performed on the refinement as described in the next sections.

3.2. Candidate Generation

We now describe how alignments between query pairs are generated. These alignments can be between words or contiguous phrases up to a given length k (typically $k = 3$ here). Formally, a *k-gram alignment* between two strings x and y over an alphabet Σ is a sequence $\pi = a_1 \dots a_l$ of alignment terms with

$$a_i = (i[a_i], o[a_i]) \in \mathcal{A}_k = (\Sigma^{\leq k} \times \Sigma^{\leq k}) - \{(\epsilon, \epsilon)\} \quad (1)$$

and such that $i[\pi] = i[a_1] \dots i[a_l] = x$ and $o[\pi] = o[a_1] \dots o[a_l] = y$. Given an edit cost function $c : \mathcal{A}_k \to \mathbb{R}$, the *cost* of an alignment π is defined as $c(\pi) = \sum_{i=1}^{l} c(a_i)$. The *edit distance* between x and y is the minimal cost of an alignment between x and y. This is essentially the classical edit distance except we allow edit operations on n-grams of order up to k. The Levenshtein distance can be obtained by setting $k = 1$ and $c(a) = 0$ if $i[a] = o[a]$ and $c(a) = 1$ otherwise.

An *alignment rewriting* ρ is a morphism mapping $\pi = a_1 \dots a_l$ to $\rho(\pi) = \rho(a_1) \dots \rho(a_l)$. In the following we will consider the *substitution rewriting* ρ_{sub}, the *deletion rewriting* ρ_{del} and the *refinement rewriting* ρ_{ref} defined as follows, for $a \in \mathcal{A}_k$:

$$\rho_{sub}(a) = \begin{cases} (i[a], i[a]) & \text{if } o[a] = \epsilon \\ a & \text{otherwise,} \end{cases} \quad (2)$$

$$\rho_{del}(a) = \begin{cases} (i[a], i[a]) & \text{if } o[a] = \epsilon \\ (i[a], \epsilon) & \text{otherwise and} \end{cases} \quad (3)$$

$$\rho_{ref}(a) = \begin{cases} (i[a], \epsilon) & \text{if } i[a] = o[a] \\ a & \text{otherwise.} \end{cases} \quad (4)$$

Given a parse (q_1, s, τ, r), we generate a set of possible candidate alignments which are alignments π such that $i[\pi] = q_1$ and $o[\pi]$ is a candidate for q_3. Let us assume that $r = \epsilon$. We first compute a set Π_e of possible edits, that is k-gram alignments between q_1 and s.

$$\Pi_e = \{\pi \in \mathcal{A}_k^* | i[\pi] = q_1, o[\pi] = s, c(\pi) < \lambda, \phi(\pi) < \theta\} \quad (5)$$

where c is an edit cost function and $\phi : \mathcal{A}_k^* \rightarrow \mathbb{R}$ is an alignment-level scoring function. Both c and ϕ depend on the type τ of refinement considered. We chose a function ϕ that penalizes non-contiguous alignments. The thresholding using c and ϕ is done to limit the number of alignments to consider.

The set Π_c of candidate alignments is generated by applying a rewriting r to each alignment in Π_e:

$$\Pi_c = \{\pi | \pi = \rho(\pi'), \pi' \in \Pi_e\}. \quad (6)$$

The rewriting ρ used depends on τ: for $\tau = \texttt{delete}$, we use $\rho = \rho_{del}$ and for $\tau \in \{\texttt{insert}, \texttt{substitute}\}$, we use $\rho = \rho_{sub}$. The case when $\tau = \texttt{substitute}$ and $r \neq \epsilon$ is handled as follows. We first align q_1 and r on one side, and r and s on the other. We then apply a rewriting similar to ρ_{sub} to the combined alignments.

In Figure 2(a), the alignment between q_1 and s is $(\texttt{Northern Italian}, \texttt{Korean})(\texttt{restaurant}, \epsilon)$ and the appplication of ρ_{sub} results in the candidate alignment $(\texttt{Northern Italian}, \texttt{Korean})(\texttt{restaurant}, \texttt{restaurant})$. The set of candidate alignments is computed and compactly represented using weighted finite-state transducers.

3.3. Candidate Selection

Given a pair $\langle q_1, q_2 \rangle$ and set of candidate alignments $\Pi = \{\pi_1, \pi_2, \ldots \pi_n\}$, the next step is to decide which candidate alignment is best.

3.3.1. Language Model-Based Selection

One approach is to measure how likely each candidate $q_3 = o[\pi_i]$ is to be a query irrespective of q_1. For example, in Figure 2, (a) `Korean restaurant` is probably a much more likely query than (c) `Northern Italian Korean`. A simple method to evaluate this is to find the probability of each $o[\pi_i]$ according to an n-gram language model $Pr[q]$ trained on queries and select $\arg\max_i Pr[o[\pi_i]]$. The language model used in our experiments was an interpolated 4-gram trained using Katz backoff from the data sources described in Section 4 and having 55 million n-grams.[1]

3.3.2. Refinement Model-Based Selection

The language model-based approach does not take advantage of the information in the original query during candidate selection. For this, we will build a query refinement model that uses the complete alignment information. Assume we have available a corpus of actual query refinements $\langle q_1, q_2, q_3 \rangle_j$ that will serve as our training corpus. For each $\langle q_1, q_2 \rangle_j$ pair, generate candidate alignments Π_j as described in the previous sections. Those alignments that correspond to q_3 are labeled with 1 as correct, the others are labeled with 0 as incorrect.

This is now a learning problem that can be approached in various ways: e.g. multiclass classification or ranking. We take a simple approach of training a binary classifier on such data and then selecting the highest scoring alignment from the classifier. In particular, we train a maximum entropy model:

$$Pr[y|\pi] = Z(\pi)e^{\theta \cdot f(\pi)}, \; y \in \{0, 1\}$$

with feature function f on the alignment, feature weights θ, and normalization $Z(\pi)$. We then select $argmax_i Pr[1|\pi_i]$ as our best scoring candidate.

Various features are used on each alignment $\pi = a_1 \ldots a_l$ including a binary feature for each observed phrase $i[a_i]$ and $o[a_i]$ and for each phrase pair a_i. Features for part-of-speech and category tags and for word shape on each $i[a_i]$ and $o[a_i]$ and tag pairs on each a_i allow syntactic and semantic generalization. The part-of-speech tags (`Korean:ADJ`, `restaurant:Noun`) are dictionary-derived, while the category tags (`Korean:{cuisine,nationality}`, `restaurant:{place,business}`) are web-derived using Hearst patterns [2]. We additionally include binary features for the edit location (prefix, suffix, infix) and edit terms.

4. Data

4.1. Language Model

The training data used for the language model in Section 3.3.1 was drawn from six anonymized and randomized sources that include voice input text from Android, transcribed and typed search queries, and SMS messages and totaled over 2 billon words.[1].

4.2. Refinement Model

Ideally, we would use transcribed voice refinement logs to train the refinement model in Section 3.3.2. That would require $\langle q_1, q_2 \rangle$ be logged and q_3 be hand-annotated or otherwise derived. However, no such data is available without an existing voice-refinement system.

To bootstrap this process, web search query logs were instead used to synthesize pseudo voice-refinement logs. Assume we observe in our logs two consecutive typed queries q_1 and q_3 from the same session such that the edit distance between q_1 and q_3 is small. This might have been a textual refinement and we can recover the underlying s and τ. We compute the best contiguous unigram alignment π between q_1 and q_3. We set $\tau = \texttt{insert}$ if $\pi \in \{(\epsilon, \sigma), (\sigma, \sigma) \,|\, \sigma \in \Sigma\}^*$ and $\tau = \texttt{delete}$ if $\pi \in \{(\sigma, \epsilon), (\sigma, \sigma) \,|\, \sigma \in \Sigma\}^*$. Otherwise, we set $\tau = \texttt{substitute}$. We can then recover s by applying the relevant alignment rewriting. For instance, if $\tau \in \{\texttt{insert}, \texttt{substitute}\}$, we apply the rewriting ρ_{ref} from Section 3.2 and set s to $o[\rho_{ref}(\pi)]$.

The training data for the refinement model was drawn from anonymized typed search query logs for a single day. To ensure no user-identifiable information is exposed, all pairs, culled by the above process, containing a query which occured in less then 50 distinct web sessions that day were discarded. This resulted in approximately 100 thousand $(q_1, s, \texttt{substitute})$ tuples and large numbers of deletions and insertions as well.

5. Results

Without true voice refinement data, it is not obvious how to evaluate the parsing step in Section 3.1. In practice, the user will simply have to stay within the grammar at this point. However, it is possible to evaluate the editing step by collecting a test set from Google's logs similar to but distinct from the training data used in Section 3.3. As with that training data, the logs were used to identify typed query refinements and to simulate a voice refinement. The evaluation task is to correctly produce q_3 given only the tuple (q_1, s, τ, r). In our experiments we only considered $\tau = \texttt{substitute}$ and $r = \epsilon$ since our initial explorations showed that the deletions, insertions, and substitutions where text to be replaced is specified were much easier to solve. Our test set consisted of 690 such substitutions. A result was scored correct only if the prediction exactly matched q_3.

We used three human judges as the baseline since no comparable systems for spoken query refinement was available. For each data point, the human judges were given the $(q_1, s, \texttt{substitute})$ and a list of the four most likely candidates produced by the candidate generation step from Section 3.2. When the correct answer did not appear in this list, we replaced the fourth slot with the correct answer (giving the human judges the advantage of a short list known to contain the correct answer). Each judge was asked to select the best candidate among the choices given, or select NONE if they did not feel any of the choices were appropriate.

The results of the human and model performance on the test set are shown in Figure 3. The columns are for the three human judges, the language model-based system (Section 3.3.1) and the refinement model-based system (Section 3.3.2). The *Full* row shows the accuracy on the full test set, and the *2Agree* row is for the subset containing only test samples for which two human judges agree. The refinement model-based system outperforms the language model-based system and all human judges.

6. Discussion

The manually-generated CFG for the parsing step in Section 3.1 is clearly subject to coverage, accuracy and ambiguity problems. Once actual voice refinement data is collected from a live system, these problems can be evaluated and addressed by improved grammars, learned weights or productions and perhaps folding some or all of the parsing step into the learned editing step.

	Judge 1	**Judge 2**	**Judge 3**	**LM**	**RM**
Full	73.6	70.7	65.1	57.1	**76.3**
2Agree	74.9	72.2	66.5	57.6	**76.9**

Figure 3: Refinement percent accuracy for the language model (LM), refinement model (RM) and three human judges performing the same task on the full dataset and a subset filtered by 2-way human agreement.

Although the refinement-based system performed better than the human judges, there are several reasons for caution when extrapolating to actual voice refinements. First, this system does relatively well when the human judge doesn't know anything about the query topic; this would not normally happen in a real system. Second, the typed logs-derived data have idiosyncrasies. For example, users favor edits at the end of the input text, some due to automatic query suggestions by the search engine. In other cases, the users correct typographic errors. This bias could be improved by filtering and eliminated by using actual voice-refinement logs once available.

7. Related Work

Past query refinement work has mostly been for typed query applications. Some have focused on refinement clustering and suggestion given a query [3, 4], using refinements to segment search sessions [5], and classifying refinements into types (e.g., word reordering, substitution) [6, 7]. Other work shows that information-seeking users choose an initial query and then refine it [8], issuing a new query only after several unsuccessful, often non-systematic [9] refinement attempts.

8. References

[1] C. Allauzen and M. Riley, "Bayesian language model interpolation for mobile speech input," in *Proc. of Interspeech*, 2011, pp. 1429–1432.

[2] M. Hearst, "Automatic acquisition of hyponyms from large text corporaa," in *Proc. of COLING '92*, 1992, pp. 539–545.

[3] E. Sadikov, J. Madhavan, L. Wang, and A. Halevy, "Clustering query refinements by user intent," *WWW*, 2010.

[4] S. Riezler and Y. Liu, "Query rewriting using monolingual statistical machine translation," *Computational Linguistics*, vol. 36, no. 3, Jan 2010.

[5] D. He, A. Göker, and D. Harper, "Combining evidence for automatic web session identification," *Information Processing and Management: an International Journal*, vol. 38, no. 5, 2002.

[6] J. Huang and E. Efthimiadis, "Analyzing and evaluating query reformulation strategies in web search logs," *CIKM*, Jan 2009.

[7] M. Whittle, B. Eaglestone, N. Ford, V. Gillet, and A. Madden, "Data mining of search engine logs," *Journal of the American Society for Information Science and Technology*, vol. 58, no. 14, 2007.

[8] A. Aula, R. M. Khan, and Z. Guan, "How does search behavior change as search becomes more difficult?" *CHI*, 2010.

[9] A. Aula and K. Nordhausen, "Modeling successful performance in web searching," *Journal of the American Society for Information Science and Technology*, vol. 57, no. 12, Jan 2006.