General Indexation of Weighted Automata – Application to Spoken Utterance Retrieval

Cyril Allauzen and **Mehryar Mohri** and **Murat Saraclar**
AT&T Labs - Research
180 Park Avenue, Florham Park, NJ 07932
{allauzen, mohri, murat}@research.att.com

Abstract

Much of the massive quantities of digitized data widely available, e.g., text, speech, hand-written sequences, are either given directly, or, as a result of some prior processing, as weighted automata. These are compact representations of a large number of alternative sequences and their weights reflecting the uncertainty or variability of the data. Thus, the indexation of such data requires indexing weighted automata.

We present a general algorithm for the indexation of weighted automata. The resulting index is represented by a deterministic weighted transducer that is *optimal* for search: the search for an input string takes time linear in the sum of the size of that string and the number of indices of the weighted automata where it appears. We also introduce a general framework based on weighted transducers that generalizes this indexation to enable the search for more complex patterns including syntactic information or for different types of sequences, e.g., word sequences instead of phonemic sequences. The use of this framework is illustrated with several examples.

We applied our general indexation algorithm and framework to the problem of indexation of speech utterances and report the results of our experiments in several tasks demonstrating that our techniques yield comparable results to previous methods, while providing greater generality, including the possibility of searching for arbitrary patterns represented by weighted automata.

1 Motivation

Much of the massive quantities of digitized data widely available is highly variable or *uncertain*. This uncertainty affects the interpretation of the data and its computational processing at various levels, e.g., natural language texts are abundantly ambiguous, speech and hand-written sequences are highly variable and hard to recognize in presence of noise, biological sequences may be altered or incomplete.

Searching or indexing such data requires dealing with a large number of ranked or weighted alternatives. These may be for example the different parses of an input text, the various responses to a search engine or information extraction query, or the best hypotheses of a speech or hand-written recognition system. In most cases, alternative sequences can be compactly represented by *weighted automata*. The weights may be probabilities or some other weights used to rank different hypotheses.

This motivates our study of the general problem of indexation of weighted automata. This is more general than the classical indexation problems since, typically, there are many distinct hypotheses or alternatives associated with the same index, e.g., a specific input speech or hand-written sequence may have a large number of different transcriptions according to the system and models used. Moreover, the problem requires taking into consideration the weight of each alternative, which does not have a counterpart in classical indexation problems.

We describe a general indexation algorithm for weighted automata. The resulting index is represented by a deterministic weighted transducer that is *optimal* for search: the search for an input string takes time linear in the sum of the size of that string and the number of indices of the weighted automata where it appears.

In some cases, one may wish to search using sequences in some level, e.g. word sequences, different from the level of the sequences of the index, e.g. phonemic sequences. One may also wish to search for complex sequences including both words and parts-of-speech, or restrict the search by either restricting the weights or probabilities or the lengths or types of sequences. We describe a general indexation framework covering all these cases. Our framework is based on the use of filtering weighted transducers for restriction or other transducers mapping between distinct information levels or knowledge structures. We illustrate the use of this framework with several examples that demonstrate its relevance to a number of indexation tasks.

We applied our framework and algorithms to the particular problem of speech indexation. In recent years, spoken document retrieval systems have made large archives of broadcast news searchable and browsable. Most of these systems use automatic speech recognition to convert speech into text, which is then indexed using standard methods. When a user presents the system with a query, documents that are relevant to the query are found using text-based information retrieval techniques.

As speech indexation and retrieval systems move beyond the domain of broadcast news to more challenging spoken communications, the importance for the indexed material to contain more than just a simple text representation of the communication is becoming clear. Indexation and retrieval techniques must be extended to handle more general representations including for example syntactic information. In addition to the now familiar retrieval systems or search engines, other applications such as data mining systems can be used to automatically identify useful patterns in large collections of spoken communications. Information extraction systems can be used to gather high-level information such as named-entities.

For a given input speech utterance, a large-vocabulary speech recognition system often generates a *lattice*, a weighted automaton representing a range of alternative hypotheses with some associated weights or probabilities used to rank them. When the accuracy of a system is relatively low as in many conversational speech recognition tasks, it is not safe to rely only on the best hypothesis output by the system. It is then preferable to use instead the full lattice output by the recognizer.

We report the results of our experiments in several tasks demonstrating that our techniques yield comparable results to the previous methods of Saraclar and Sproat (2004), while providing greater generality, including the possibility of searching for arbitrary patterns represented by weighted automata.

The paper is organized as follows. Section 2 introduces the notation and the definitions used in the rest of the paper. Section 3 describes our general indexation algorithm for weighted automata. The algorithm for searching that index is presented in Section 4 and our general indexation framework is described and illustrated in Section 5. Section 6 reports the results of our experiments in several tasks.

2 Preliminaries

Definition 1 *A system* $(\mathbb{K}, \oplus, \otimes, \overline{0}, \overline{1})$ *is a* semiring *(Kuich and Salomaa, 1986) if:* $(\mathbb{K}, \oplus, \overline{0})$ *is a commutative monoid with identity element* $\overline{0}$*;* $(\mathbb{K}, \otimes, \overline{1})$ *is a monoid with identity element* $\overline{1}$*;* $\otimes$ *distributes over* $\oplus$*; and* $\overline{0}$ *is an annihilator for* $\otimes$*: for all* $a \in \mathbb{K}, a \otimes \overline{0} = \overline{0} \otimes a = \overline{0}$*.*

Thus, a semiring is a ring that may lack negation. Two semirings often used in speech processing are: the *log semiring* $\mathcal{L} = (\mathbb{R} \cup \{\infty\}, \oplus_{\log}, +, \infty, 0)$ (Mohri, 2002) which is isomorphic to the familiar real or probability semiring $(\mathbb{R}_+, +, \times, 0, 1)$ via a $\log$ morphism with, for all $a, b \in \mathbb{R} \cup \{\infty\}$:

$$a \oplus_{\log} b = -\log(\exp(-a) + \exp(-b))$$

and the convention that: $\exp(-\infty) = 0$ and $-\log(0) = \infty$, and the *tropical semiring* $\mathcal{T} = (\mathbb{R}_+ \cup \{\infty\}, \min, +, \infty, 0)$ which can be derived from the log semiring using the Viterbi approximation.

Definition 2 *A* weighted finite-state transducer T *over a semiring* $\mathbb{K}$ *is an 8-tuple* $T = (\Sigma, \Delta, Q, I, F, E, \lambda, \rho)$ *where:* Σ *is the finite input alphabet of the transducer;* Δ *is the finite output alphabet;* Q *is a finite set of states;* $I \subseteq Q$ *the set of initial states;* $F \subseteq Q$ *the set of final states;* $E \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times (\Delta \cup \{\epsilon\}) \times \mathbb{K} \times Q$ *a finite set of transitions;* $\lambda : I \to \mathbb{K}$ *the initial weight function; and* $\rho : F \to \mathbb{K}$ *the final weight function mapping* F *to* $\mathbb{K}$*.*

A *Weighted automaton* $A = (\Sigma, Q, I, F, E, \lambda, \rho)$ is defined in a similar way by simply omitting the output labels. We denote by $L(A) \subseteq \Sigma^*$ the set of strings accepted by an automaton A and similarly by $L(X)$ the strings described by a regular expression X. We denote by $|A| = |Q| + |E|$ the size of A.

Given a transition $e \in E$, we denote by $i[e]$ its input label, $p[e]$ its origin or previous state and $n[e]$ its destination state or next state, $w[e]$ its weight, o[e] its output label (transducer case). Given a state $q \in Q$, we denote by $E[q]$ the set of transitions leaving q.

A *path* $\pi = e_1 \cdots e_k$ is an element of E^* with consecutive transitions: $n[e_{i-1}] = p[e_i]$, $i = 2, \ldots, k$. We extend n and p to paths by setting: $n[\pi] = n[e_k]$ and $p[\pi] = p[e_1]$. A cycle π is a path whose origin and destination states coincide: $n[\pi] = p[\pi]$. We denote by $P(q, q')$ the set of paths from q to q' and by $P(q, x, q')$ and $P(q, x, y, q')$ the set of paths from q to q' with input label $x \in \Sigma^*$ and output label y (transducer case). These definitions can be extended to subsets $R, R' \subseteq Q$, by: $P(R, x, R') = \cup_{q \in R,\, q' \in R'} P(q, x, q')$. The labeling functions i (and similarly o) and the weight function w can also be extended to paths by defining the label of a path as the concatenation of the labels of its constituent transitions, and the weight of a path as the $\otimes$-product of the weights of its constituent transitions:

$i[\pi] = i[e_1] \cdots i[e_k]$, $w[\pi] = w[e_1] \otimes \cdots \otimes w[e_k]$. We also extend w to any finite set of paths Π by setting: $w[\Pi] = \bigoplus_{\pi \in \Pi} w[\pi]$. The output weight associated by A to each input string $x \in \Sigma^*$ is:

$$[\![A]\!](x) = \bigoplus_{\pi \in P(I,x,F)} \lambda(p[\pi]) \otimes w[\pi] \otimes \rho(n[\pi])$$

$[\![A]\!](x)$ is defined to be $\overline{0}$ when $P(I,x,F) = \emptyset$. Similarly, the output weight associated by a transducer T to a pair of input-output string (x,y) is:

$$[\![T]\!](x,y) = \bigoplus_{\pi \in P(I,x,y,F)} \lambda(p[\pi]) \otimes w[\pi] \otimes \rho(n[\pi])$$

$[\![T]\!](x,y) = \overline{0}$ when $P(I,x,y,F) = \emptyset$. A *successful path* in a weighted automaton or transducer M is a path from an initial state to a final state. M is *unambiguous* if for any string $x \in \Sigma^*$ there is at most one successful path labeled with x. Thus, an unambiguous transducer defines a function.

For any transducer T, denote by $\Pi_2(T)$ the automaton obtained by projecting T on its output, that is by omitting its input labels.

Note that the second operation of the tropical semiring and the log semiring as well as their identity elements are identical. Thus the weight of a path in an automaton A over the tropical semiring does not change if A is viewed as a weighted automaton over the log semiring or vice-versa.

Given two strings u and v in Σ^*, v is a *factor* of u if $u = xvy$ for some x and y in Σ^*; if $y = \epsilon$ then v is also a *suffix* of u. More generally, v is a *factor* (resp. *suffix*) of $L \subseteq \Sigma^*$ if v is a suffix (resp. factor) of some $u \in L$. We denote by $|x|$ the length of a string $x \in \Sigma^*$.

3 Indexation Algorithm

This section presents an algorithm for the construction of an efficient index for a large set of speech utterances.

We assume that for each speech utterance u_i of the dataset considered, $i = 1, \ldots, n$, a weighted automaton A_i over the alphabet Σ and the log semiring, e.g., phone or word lattice output by an automatic speech recognizer, is given. The problem consists of creating a full index, that is one that can be used to search directly any factor of any string accepted by these automata. Note that this problem crucially differs from classical indexation problems in that the input data is uncertain. Our algorithm must make use of the weights associated to each string by the input automata.

The main idea behind the design of the algorithm described is that the full index can be represented by a weighted finite-state transducer T mapping each factor x to the set of indices of the automata in which x appears and the negative log of the expected count of x. More precisely, let P_i be the probability distribution defined by the weighted automaton A_i over the set of strings Σ^* and let $C_x(u)$ denote the number of occurrences of a factor x in u, then, for any factor $x \in \Sigma^*$ and automaton index $i \in \{1, \ldots, n\}$:

$$[\![T]\!](x,i) = -\log(E_{P_i}[C_x]) \qquad (1)$$

Our algorithm for the construction of the index is simple, it is based on general weighted automata and transducer algorithms. We describe the consecutive stages of the algorithm.

This algorithm can be seen as a generalization to weighted automata of the notion of *suffix automaton* and *factor automaton* for strings. The suffix (factor) automaton of a string u is the minimal deterministic finite automata recognizing exactly the set of suffixes (resp. factors) of u (Blumer et al., 1985; Crochemore, 1986). The size of both automata is linear in the length of u and both can be built in linear time. These are classical representations used in text indexation (Blumer et al., 1987; Crochemore, 1986).

3.1 Preprocessing

When the automata A_i are word or phone lattices output by a speech recognition or other natural language processing system, the path weights correspond to joint probabilities. We can apply to A_i a general weight-pushing algorithm in the log semiring (Mohri, 1997) which converts these weights into the desired (negative log of) posterior probabilities. More generally, the path weights in the resulting automata can be interpreted as log-likelihoods. We denote by P_i the corresponding probability distribution. When the input automaton A_i is acyclic, the complexity of the weight-pushing algorithm is linear in its size ($O(|A_i|)$). Figures 1(b)(d) illustrates the application of the algorithm to the automata of Figures 1(a)(c).

3.2 Construction of Transducer Index T

Let $B_i = (\Sigma, Q_i, I_i, F_i, E_i, \lambda_i, \rho_i)$ denote the result of the application of the weight pushing algorithm to the automaton A_i. The weight associated by B_i to each string it accepts can be interpreted as the log-likelihood of that string for the utterance u_i given the models used to generate the automata. More generally, B_i defines a probability distribution P_i over all strings $x \in \Sigma^*$ which is just the sum of the probability of all paths of B_i in which x appears.

For each state $q \in Q_i$, denote by $d[q]$ the shortest distance from I_i to q (or -log of the forward probability) and by $f[q]$ the shortest distance from q to F (or -log of the backward probability):

$$d[q] = \bigoplus_{\substack{\log \\ \pi \in P(I_i,q)}} (\lambda_i(p[\pi]) + w[\pi]) \qquad (2)$$

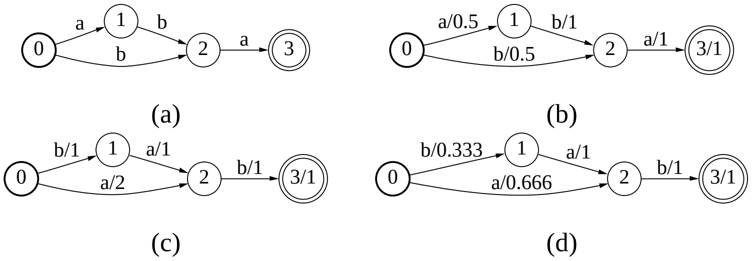


Figure 1: Weighted automata over the real semiring (a) A_1, (b) B_1 obtained by applying weight pushing to A_1, (c) A_2 and (d) B_2 obtained by applying weight pushing to A_2.

$$f[q] = \bigoplus_{\substack{\log \\ \pi \in P(q,F_i)}} (w[\pi] + \rho_i(n[\pi])) \tag{3}$$

The shortest distances $d[q]$ and $f[q]$ can be computed for all states $q \in Q_i$ in linear time ($O(|B_i|)$) when B_i is acyclic (Mohri, 2002). Then,

$$-\log(E_{P_i}[C_x]) = \bigoplus_{\substack{\log \\ i[\pi]=x}} d[p[\pi]] + w[\pi] + f[n[\pi]] \tag{4}$$

From the weighted automaton B_i, one can derive a weighted transducer T_i in two steps:

1. Factor Selection. In the general case we select all the factors to be indexed in the following way:
   - Replace each transition $(p, a, w, q) \in Q_i \times \Sigma \times \mathbb{R} \times Q_i$ by $(p, a, a, w, q) \in Q_i \times \Sigma \times \Sigma \times \mathbb{R} \times Q_i$;
   - Create a new state $s \notin Q_i$ and make s the unique initial state;
   - Create a new state $e \notin Q_i$ and make e the unique final state;
   - Create a new transition $(s, \epsilon, \epsilon, d[q], q)$ for each state $q \in Q_i$;
   - Create a new transition $(q, \epsilon, i, f[q], e)$ for each state $q \in Q_i$;
2. Optimization. The resulting transducer can be optimized by applying weighted ϵ-removal, weighted determinization, and minimization over the log semiring by viewing it as an acceptor, i.e., input-output labels are encoded a single labels.

It is clear from Equation 4 that for any factor $x \in \Sigma^*$:

$$[\![T_i]\!](x, i) = -\log(E_{P_i}[C_x]) \tag{5}$$

This construction is illustrated by Figures 2(a)(b). Our full index transducer T is the constructed by

- taking the $\oplus_{\log}$-sum (or union) of all the transducers T_i, $i = 1, \ldots, n$;
- defining T as the result of determinization (in the log semiring) applied to that transducer.

Figure 3 is illustrating this construction and optimization.

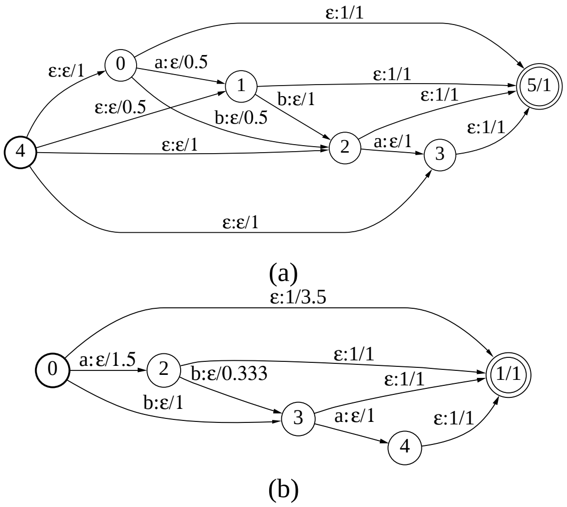


Figure 2: Construction of T_1 index of the weighted automata B_1 given Figure 1(b): (a) intermediary result after factor selection and (b) resulting weighted transducer T_1.

4 Search

The full index represented by the weighted finite-state transducer T is optimal. Indeed, T contains no transition with input ϵ other than the final transitions labeled with an output index and it is deterministic. Thus, the set of indices I_x of the weighted automata containing a factor x can be obtained in $O(|x| + |I_x|)$ by reading in T the unique path with input label x and then the transitions with input ϵ which have each a distinct output label.

The user's query is typically an unweighted string, but it can be given as an arbitrary weighted automaton X. This covers the case of Boolean queries or regular expressions which can be compiled into automata. The response to a query X is computed using the general algorithm of composition of weighted transducers (Mohri et al., 1996) followed by projection on the output:

$$\Pi_2(X \circ T) \tag{6}$$

which is then ϵ-removed and determinized to give directly the list of all indices and their corresponding log-

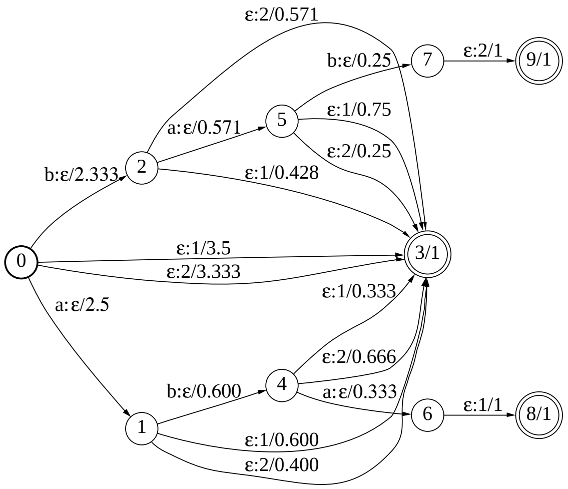


Figure 3: Weighted transducer T obtained by indexing the weighted automata B_1 and B_2 given in Figures 1(b)(d)

likelihoods. The final result can be pruned to include only the most likely responses. The pruning threshold may be used to vary the number of responses.

5 General Indexation Framework

The indexation technique just outlined can be easily extended to include many of the techniques used for speech indexation. This can be done by introducing a transducer F that converts between different levels of information sources or structures, or that filters out or reweights index entries. The filter F can be applied (i) before, (ii) during or (iii) after the construction of the index. For case (i), the filter is used directly on the input and the indexation algorithm is applied to the weighted automata $(F \circ A_i)_{1 \leq i \leq n}$. For case (ii), filtering is done after the factor selection step of the algorithm and the filter applies to the factors, typically to restrict the factors that will be indexed. For case (iii), the filter is applied to the index. Obviously different filters can be used in combination at different stages.

When such a filter is used, the response to a query X is obtained using another transducer F' [1] and the following composition and projection:

$$\Pi_2(X \circ F' \circ T) \qquad (7)$$

Since composition is associative, it does not impose a specific order to its application. However, in practice, it is often advantageous to compute $X \circ F'$ before application of T. The following are examples of some filter transducers that can be of interest in many applications.

- **Pronunciation Dictionary:** a pronunciation dictionary can be used to map word sequences into their phonemic transcriptions, thus transform word lattices into equivalent phone lattices. This mapping can represented by a weighted transducer F. Using an index based on phone lattices allows a user to search for words that are not in the ASR vocabulary. In this case, the inverse transduction F' is a grapheme to phoneme converter, commonly present in TTS front-ends. Among others, Witbrock and Hauptmann (1997) present a system where a phonetic transcript is obtained from the word transcript and retrieval is performed using both word and phone indices.

- **Vocabulary Restriction:** in some cases using a full index can be prohibitive and unnecessary. It might be desirable to do partial indexing by ignoring some words (or phones) in the input. For example, we might wish to index only "named entities", or just the consonants. This is mostly motivated by the reduction of the size of the index while retaining the necessary information. A similar approach is to apply a many to one mapping to index groups of phones, or metaphones (Amir et al., 2001), to overcome phonetic errors.

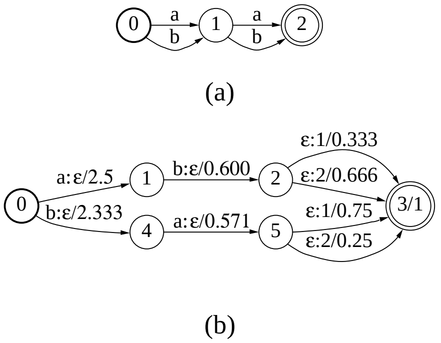
- **Reweighting:** a weighted transducer can be used to emphasize some words in the input while de-emphasizing other. The weights, for example might correspond to TF-IDF weights. Another reweighting method might involve edit distance or confusion statistics.

- **Classification:** an extreme form of summarizing the information contained in the indexed material is to assign a class label, such as a topic label, to each input. The query would also be classified and all answers with the same class label would be returned as relevant.

- **Length Restriction:** a common way of indexing phone strings is to index fixed length overlapping phone strings (Logan et al., 2002). This results in a partial index with only fixed length strings. More generally a minimum and maximum string length may be imposed on the index. An example restriction automaton is given in Figure 4. In this case, the filter applies to the factors and has to be applied during or after indexation. The restricted index will be smaller in size but contains less information and may result in degradation in retrieval performance, especially for long queries.

The length restriction filter requires a modification of the search procedure. Assume a fixed – say r – length restriction filter and a string query of length k. If $k < r$,

[1] In most cases, F' is the inverse of F.



(a)

(b)

Figure 4: (a) Filter F restricting to strings of length 2. (b) Restricted index $F \circ T$, where T is the weighted transducer given in Figure 3(b).

then we need to pad the input to length r with Σ^{r-k}. If $k \geq r$, then we must search for all substrings of length r in the index. A string is present in a certain lattice if all its substrings are (and not vice versa). So, the results of each substring search must be intersected. The probability of each substring x_i^{i+r-1} for $i \in \{1, \ldots, k+1-r\}$ is an upper bound on the probability of the string x_1^k, and the count of each substring is an upper bound on the count of the string, so for $i \in \{1, \ldots, k+1-r\}$

$$E_P[C(x_1^k)] \leq E_P[C(x_i^{i+r-1})].$$

Therefore, the intersection operation must use minimum for combining the expected counts of substrings. In other words, the expected count of the string is approximated by the minimum of the probabilities of each of its substrings,

$$E_P[C(x_1^k)] \approx \min_{1 \leq i \leq k+1-r} E_P[C(x_i^{i+r-1})].$$

In addition to a filter transducer, pruning can be applied at different stages of the algorithm to reduce the size of the index. Pruning eliminates least likely paths in a weighted automaton or transducer. Applying pruning to A_i can be seen as part of the process that generates the uncertain input data. When pruning is applied to B_i, only the more likely alternatives will be indexed. If pruning is applied to T_i, or to T, pruning takes the expected counts into consideration and not the probabilities. Note that the threshold used for this type of pruning is directly comparable to the threshold used for pruning the search results in Section 4 since both are thresholds on expected counts.

6 Experimental Results

Our task is retrieving the utterances (or short audio segments) that a given query appears in. The experimental setup is identical to that of Saraclar and Sproat (2004). Since, we take the system described there as our baseline, we give a brief review of the basic indexation algorithm used there. The algorithm uses the same preprocessing step. For each label in Σ, an index file is constructed. For each arc a that appears in the preprocessed weighted automaton B_i, the following information is stored: $(i, p[a], n[a], d[p[a]], w[a])$. Since the preprocessing ensures that $f[q] = 0$ for all q in B_i, it is possible to compute $-\log(E_{P_i}[C_x])$ as in Equation 4 using the information stored in the index.

6.1 Evaluation Metrics

For evaluating retrieval performance we use precision and recall with respect to manual transcriptions. Let $\mathrm{Correct}(q)$ be the number of times the query q is found correctly, $\mathrm{Answer}(q)$ be the number of answers to the query q, and $\mathrm{Reference}(q)$ be the number of times q is found in the reference.

$$\mathrm{Precision}(q) = \frac{\mathrm{Correct}(q)}{\mathrm{Answer}(q)}$$

$$\mathrm{Recall}(q) = \frac{\mathrm{Correct}(q)}{\mathrm{Reference}(q)}$$

We compute precision and recall rates for each query and report the average over all queries. The set of queries Q includes all the words seen in the reference except for a stoplist of 100 most common words.

$$\mathrm{Precision} = \frac{1}{|Q|} \sum_{q \in Q} \mathrm{Precision}(q)$$

$$\mathrm{Recall} = \frac{1}{|Q|} \sum_{q \in Q} \mathrm{Recall}(q)$$

For lattice based retrieval methods, different operating points can be obtained by changing the threshold. The precision and recall at these operating points can be plotted as a curve.

In addition to individual precision-recall values we also compute the F-measure defined as

$$F = \frac{2 \times \mathrm{Precision} \times \mathrm{Recall}}{\mathrm{Precision} + \mathrm{Recall}}$$

and report the maximum F-measure (maxF) to summarize the information in a precision-recall curve.

6.2 Corpora

We use three different corpora to assess the effectiveness of different retrieval techniques.

The first corpus is the DARPA Broadcast News corpus consisting of excerpts from TV or radio programs including various acoustic conditions. The test set is the 1998 Hub-4 Broadcast News (hub4e98) evaluation test set (available from LDC, Catalog no. LDC2000S86) which is 3 hours long and was manually segmented into 940 segments. It contains 32411 word tokens and 4885 word types. For ASR we use a real-time system (Saraclar et al., 2002). Since the system was designed for SDR, the recognition vocabulary of the system has over 200K words.

The second corpus is the Switchboard corpus consisting of two party telephone conversations. The test set is the RT02 evaluation test set which is 5 hours long, has 120 conversation sides and was manually segmented into 6266 segments. It contains 65255 word tokens and 3788 word types. For ASR we use the first pass of the evaluation system (Ljolje et al., 2002). The recognition vocabulary of the system has over 45K words.

The third corpus is named *Teleconferences* since it consists of multi-party teleconferences on various topics. A test set of six teleconferences (about 3.5 hours) was transcribed. It contains 31106 word tokens and 2779 word types. Calls are automatically segmented into a total of 1157 segments prior to ASR. We again use the first pass of the Switchboard evaluation system for ASR.

We use the AT&T DCD Library (Allauzen et al., 2003) as our ASR decoder and our implementation of the algorithm is based on the AT&T FSM Library (Mohri et al., 2000), both of which are available for download.

6.3 Results

We implemented some of the proposed techniques and made comparisons with the previous method used by Saraclar and Sproat (2004). The full indexing method consumed too much time while indexing Broadcast News lattices and used too much memory while indexing phone lattices for Teleconferences. In the other cases, we confirmed that the new method yields identical results. In Table 1 we compare the index sizes for full indexing and partial indexing with the previous method. In both cases, the input lattices are pruned so that the cost (negative log probability) difference between two paths is less than six. Although the new method results in much smaller index sizes for the string case (i.e. nbest=1), it can result in very large index sizes for full indexing of lattices (cost=6). However, partial indexing by length restriction solves this problem. For the results reported in Table 1, the length of the word strings to be indexed was restricted to be less than or equal to four, and the length of the phone strings to be indexed was restricted to be exactly four.

In Saraclar and Sproat (2004), it was shown that using word lattices yields a relative gain of 3-5% in maxF over using best word hypotheses. Furthermore, it was shown that a “search cascade” strategy for using both word and phone indices increases the relative gain over the baseline to 8-12%. In this strategy, we first search the word index for the given query, if no matches are found we search the phone index. Using the partial indices, we obtained a precision recall performance that is almost identical to the one obtained with the previous method. Comparison of the maximum F-measure for both methods is given in Table 2.

Task	Previous Method	Partial Index
Broadcast News	86.0	86.1
Switchboard	60.5	60.8
Teleconferences	52.8	52.7

Table 2: Comparison of maximum F-measure for three corpora.

As an example, we used a filter that indexes only consonants (i.e. maps the vowels to ϵ). The resulting index was used instead of the full phone index. The size of the consonants only index was 370MB whereas the size of the full index was 431MB. In Figure 5 we present the precision recall performance of this consonant only index.

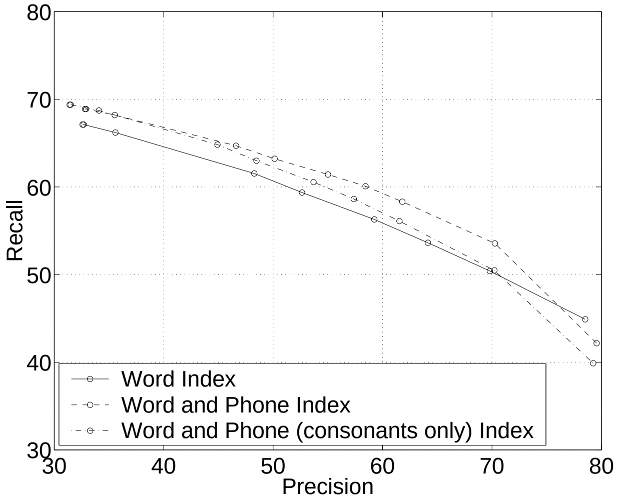


Figure 5: Comparison of Precision vs Recall Performance for Switchboard.

7 Conclusion

We described a general framework for indexing uncertain input data represented as weighted automata. The indexation algorithm utilizes weighted finite-state algorithms to obtain an index represented as a weighted finite-state transducer. We showed that many of the techniques used for speech indexing can be implemented within this framework. We gave comparative results to a previous method for lattice indexing.

The same idea and framework can be used for indexation in natural language processing or other areas where uncertain input data is given as weighted automata. The complexity of the index construction algorithm can be improved in some general cases using techniques similar to classical string matching ones (Blumer et al., 1985;

Task	Type	Pruning	Previous Method	Full Index	Partial Index
Broadcast News	word	nbest=1	29	2.7	–
Broadcast News	word	cost=6	91	–	25
Broadcast News	phone	cost=6	27	–	14
Switchboard	word	nbest=1	18	4.7	–
Switchboard	word	cost=6	90	99	88
Switchboard	phone	cost=6	97	431	41
Teleconferences	word	nbest=1	16	2.6	–
Teleconferences	word	cost=6	142	352	184
Teleconferences	phone	cost=6	146	–	69

Table 1: Comparison of Index Sizes in MegaBytes.

Crochemore, 1986; Blumer et al., 1987). Various pruning techniques can be applied to reduce the size of the index without significantly degrading performance. Finally, other types of filters that make use of the general framework can be investigated.

Acknowledgments

We wish to thank our colleague Richard Sproat for useful discussions and the use of the lattice indexing software (`lctools`) used in our baseline experiments.

References

Cyril Allauzen, Mehryar Mohri, and Michael Riley. 2003. DCD Library - Decoder Library. http://www.research.att.com/sw/tools/dcd.

Arnon Amir, Alon Efrat, and Savitha Srinivasan. 2001. Advances in phonetic word spotting. In *Proceedings of the Tenth International Conference on Information and Knowledge Management*, pages 580–582, Atlanta, Georgia, USA.

Anselm Blumer, Janet Blumer, Andrzej Ehrenfeucht, David Haussler, and Joel Seiferas. 1985. The smallest automaton recognizing the subwords of a text. *Theoretical Computer Science*, 40(1):31–55.

Anselm Blumer, Janet Blumer, David Haussler, Ross McConnel, and Andrzej Ehrenfeucht. 1987. Complete inverted files for efficient text retrieval and analysis. *Journal of the ACM*, 34(3):578–595.

Maxime Crochemore. 1986. Transducers and repetitions. *Theoretical Computer Science*, 45(1):63–86.

Werner Kuich and Arto Salomaa. 1986. *Semirings, Automata, Languages*. Number 5 in EATCS Monographs on Theoretical Computer Science. Springer-Verlag, Berlin, Germany.

Andrej Ljolje, Murat Saraclar, Michiel Bacchiani, Michael Collins, and Brian Roark. 2002. The AT&T RT-02 STT system. In *Proc. RT02 Workshop*, Vienna, Virginia.

Beth Logan, Pedro Moreno, and Om Deshmukh. 2002. Word and sub-word indexing approaches for reducing the effects of OOV queries on spoken audio. In *Proc. HLT*.

Mehryar Mohri, Fernando C. N. Pereira, and Michael Riley. 1996. Weighted Automata in Text and Speech Processing. In *Proceedings of the 12th biennial European Conference on Artificial Intelligence (ECAI-96), Workshop on Extended finite state models of language, Budapest, Hungary*.

Mehryar Mohri, Fernando C. N. Pereira, and Michael Riley. 2000. The Design Principles of a Weighted Finite-State Transducer Library. *Theoretical Computer Science*, 231:17–32, January. http://www.research.att.com/sw/tools/fsm.

Mehryar Mohri. 1997. Finite-State Transducers in Language and Speech Processing. *Computational Linguistics*, 23:2.

Mehryar Mohri. 2002. Semiring Frameworks and Algorithms for Shortest-Distance Problems. *Journal of Automata, Languages and Combinatorics*, 7(3):321–350.

Murat Saraclar and Richard Sproat. 2004. Lattice-based search for spoken utterance retrieval. In *Proc. HLT-NAACL*.

Murat Saraclar, Michael Riley, Enrico Bocchieri, and Vincent Goffin. 2002. Towards automatic closed captioning: Low latency real-time broadcast news transcription. In *Proceedings of the International Conference on Spoken Language Processing (ICSLP)*, Denver, Colorado, USA.

Michael Witbrock and Alexander Hauptmann. 1997. Using words and phonetic strings for efficient information retrieval from imperfectly transcribed spoken documents. In *2nd ACM International Conference on Digital Libraries (DL'97)*, pages 30–35, Philadelphia, PA, July.