Lattice Indexing for Spoken Term Detection

Doğan Can, *Student Member, IEEE*, and Murat Saraçlar, *Member, IEEE*

***Abstract*—This paper considers the problem of constructing an efficient inverted index for the spoken term detection (STD) task. More specifically, we construct a deterministic weighted finite-state transducer storing soft-hits in the form of (utterance ID, start time, end time, posterior score) quadruplets. We propose a generalized factor transducer structure which retains the time information necessary for performing STD. The required information is embedded into the path weights of the factor transducer without disrupting the inherent optimality. We also describe how to index all substrings seen in a collection of raw automatic speech recognition lattices using the proposed structure. Our STD indexing/search implementation is built upon the OpenFst Library and is designed to scale well to large problems. Experiments on Turkish and English data sets corroborate our claims.**

***Index Terms*—Factor automata, lattice indexing, speech retrieval (SR), spoken term detection (STD), weighted finite-state transducers.**

I. Introduction

THE ever-increasing availability of vast multimedia archives calls for solutions to efficiently index and search them. Speech retrieval (SR) is a key information technology which integrates automatic speech recognition (ASR) and information retrieval to provide large scale access to spoken content. In an ideal SR setup, the ASR component would accurately convert speech to text and text retrieval methods would be applied on the recognition output. Unfortunately, state-of-the-art ASR systems are far from being reliable when it comes to transcribing unconstrained speech recorded in uncontrolled environments. Considering the heterogeneous nature of the large spoken databases, it is no surprise that SR research is mainly about compensating for ASR deficiencies.

In a realistic SR scenario, the end-user should be able to perform open-vocabulary search over a large collection of spoken documents in a matter of seconds. Therefore, the speech corpus must be indexed prior to search without the advance knowledge of the query terms. This is a challenging task. In text retrieval, the corpus is exact in the sense that it is known whether a particular word occupies a particular position in a document. In SR, however, the corpus is the output of the ASR component which is inherently inexact. As a consequence, speech indexers have to deal with the fact that any query word may occur anywhere in a given corpus of spoken documents.

Indexing ASR lattices, instead of the best ASR hypothesis, is a widely used SR method [1]–[3] for dealing with low quality ASR output. By assigning a probability to whether a string occupies a particular position in a spoken document (given the ASR lattice), we can significantly increase the recall rates for in-vocabulary (IV) query terms. Sub-word indexing, another well-studied SR method [1], [3]–[5], enables the retrieval of out-of-vocabulary (OOV) query terms by performing the search at the sub-word level. While being crucial for high-performing SR systems, these methods bring in significant processing and storage overhead, and hence necessitate space and search-time efficient implementations.

ASR lattices carry a large amount of connectivity information which is hard to capture with standard text retrieval systems. In [1], the authors propose an exact approach that constructs an inverted index from raw ASR lattices while storing the full connectivity information. They outline a method for exact calculation of n-gram expected counts from input lattices. In [2], this exact approach is employed in a more general indexing framework which offers optimal search complexity for the spoken utterance retrieval (SUR) task. Recent SR approaches [4], [6], [7] tend to discard most of the connectivity information present in the lattices and resort to approximate lattice representations, such as confusion networks (CNs) and position specific posterior lattices (PSPLs). These approaches argue that raw lattices contain a great deal of redundancy for SR applications and that exact inversion methods lack the proximity information required for relevance ranking.

In this paper, we consider the problem of constructing an exact inverted index for ASR lattices with time information, i.e., we index all substrings seen in the lattices along with their time alignments and posterior probabilities. Since the number of substrings is exponential in data size, in general it is infeasible to maintain an exact index with constant search complexity, e.g., a simple hash table keyed on substrings. Keeping a single word or n-gram index, on the other hand, is suboptimal since 1) such an index can not answer whether a substring actually matches a partial path in a lattice and 2) substring occurrence probabilities have to be approximated using the probabilities assigned to individual words or n-grams. Then, the challenge is to come up with an exact index structure which will efficiently store all substring occurrences while keeping the search complexity linear in the query length.

In the following sections, we generalize the index structure of [2] to accommodate the timing information and employ it in the spoken term detection (STD) task. The proposed structure is a general deterministic sequence index which retains auxil-

Manuscript received July 16, 2010; revised November 12, 2010 and March 02, 2011; accepted March 04, 2011. Date of publication April 21, 2011; date of current version August 19, 2011. This work was supported in part by The Scientific and Technological Research Council of Turkey (TÜBİTAK) under Project 105E102. The work of D. Can was supported in part by TÜBİTAK BİDEB. The work of M. Saraçlar was supported by the Turkish Academy of Sciences (TÜBA) GEBİP Award. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Gokhan Tur.

D. Can is with the University of Southern California, Los Angeles, CA 90089 USA (e-mail: dogancan@usc.edu).

M. Saraçlar is with the Electrical and Electronics Engineering Department, Boğaziçi University, Bebek, İstanbul 34342, Turkey (e-mail: murat.saraclar@boun.edu.tr).

Digital Object Identifier 10.1109/TASL.2011.2134087

iary weight information about lattice nodes, e.g., node timings in the case of STD. We provide retrieval experiments with IV query sets and show that the proposed method is effective for both word-based and phonetic indexing. Section II gives a brief review of relevant works that lead up to the current study. In Section III, we introduce the definitions and terminology used throughout the paper. Section IV describes the construction of the proposed inverted index structure from raw ASR lattices. Section V details our ASR architecture and the data used in experiments. Section VI provides the STD experiments evaluating the performance of the proposed index structure over a large data set. Finally, in Section VII, we summarize the advantages of the proposed structure and discuss future directions.

II. Related Work

ASR hypotheses are typically stored in the form of weighted directed acyclic graphs known as lattices. Using weighted finite-state transducers (WFSTs) [8] to represent ASR lattices has much appeal due to the general-purpose search, optimization and combination algorithms supplied by the WFST framework. The problem of indexing and searching a set of lattices in WFST form can be posed as an extension to the extensively studied problem of searching for patterns in a collection of text documents [9], [10]. An efficient solution [11] to the latter problem makes use of a structure known as the *factor transducer* [12]. A factor transducer (FT) is an inverted index of the set of substrings (factors) of the set of strings comprising a document. It is a very efficient sequence index and is suitable for SR applications where exact sequence matches are desired. SUR and STD are two such SR applications which aim to find respectively the utterances and the time intervals in those utterances that contain the exact sequence of query words.

In [2], the FT structure is extended to indexing weighted finite-state automata and employed in the SUR task. In this context, the FT stores soft-hit indices in the form of (utterance ID, expected count) pairs. Each successful FT path encodes a factor appearance. Input labels of each such path carry a factor, output labels carry the utterance ID and path weight gives the expected count of the factor in the corresponding utterance. Expected term counts, mere generalizations of the traditional term frequencies, provide a good relevance metric for the SUR task. Being a deterministic automaton (except final transitions), the FT offers a search complexity linear in the sum of the query length and the number of utterances in which the query term appears.

In the STD 2006 Evaluation Plan [13], NIST defines the STD task as finding all of the occurrences of each given *term*—a sequence of words spoken consecutively—in a large corpus of speech material. Since it is required to find the exact locations in time, ideally an inverted index for STD should provide soft-hits as (utterance ID, start time, end time, relevance score) quadruplets. In [14], the FT is utilized in a two-stage STD system which performs utterance retrieval followed by time alignment over the audio segments. This two-stage STD strategy is problematic: 1) it requires a costly alignment operation which is performed online and 2) the index stores within utterance expected term counts which are not direct relevance measures for STD since a query term may appear more than once in an utterance. In [15], we presented a method to obtain posterior probabilities over time intervals instead of expected counts over utterances along with a modified factor transducer (MFT) structure which stores (utterance ID, start time, end time, posterior probability) quadruplets. The MFT stores the timing information on the output labels and allows to perform the STD task in a single step greatly reducing the time spent for online retrieval. Furthermore, posterior probabilities provide a direct relevance metric for STD solving the second issue of the two-stage strategy. On the flip side, the MFT has its own deficiencies: 1) search complexity is suboptimal since the index is non-deterministic—timing information is on the arc labels—and 2) timing information has to be quantized to control the level of non-determinism.

III. Preliminaries

In the following subsections, we first introduce the general algebraic notion of a semiring [16], [17] along with semirings widely used in text and speech processing [8]. Then we recap the relevant string and automata definitions and terminology [2], [8], [17] used throughout the paper.

A. Semirings

Definition 1: A *monoid* is a triple $(\mathbb{K}, \otimes, \overline{1})$, where $\otimes$ is a closed associative binary operator on the set $\mathbb{K}$, and $\overline{1}$ is the identity element for $\otimes$. A monoid is commutative if $\otimes$ is commutative.

Definition 2: A *semiring* is a 5-tuple $(\mathbb{K}, \oplus, \otimes, \overline{0}, \overline{1})$, where $(\mathbb{K}, \oplus, \overline{0})$ is a commutative monoid, $(\mathbb{K}, \otimes, \overline{1})$ is a monoid, $\otimes$ distributes over $\oplus$, $\overline{0}$ is an annihilator for $\otimes$. A semiring is *idempotent* if $\forall a \in \mathbb{K}, a \oplus a = a$.

Lemma 1: If $(\mathbb{K}, \oplus, \otimes, \overline{0}, \overline{1})$ is an idempotent semiring, then the relation $\leq$ defined by

$$\forall a, b \in \mathbb{K}: \quad (a \leq b) \iff (a \oplus b = a)$$

is a partial order over $\mathbb{K}$, called the *natural order* [18] over $\mathbb{K}$. It is a total order if and only if the semiring has the path property: $\forall a, b \in \mathbb{K}, a \oplus b = a$ or $a \oplus b = b$.

Definition 3: An idempotent semiring $(\mathbb{K}, \oplus, \otimes, \overline{0}, \overline{1})$ is *totally ordered* if its natural order is a total ordering.

In speech processing, two semirings are of particular importance. The *log semiring* is defined as

$$\mathcal{L} = (\mathbb{R} \cup \{-\infty, +\infty\}, \oplus_{\log}, +, +\infty, 0)$$

where $\forall a, b \in \mathbb{R} \cup \{-\infty, +\infty\} : a \oplus_{\log} b = -\log(e^{-a} + e^{-b})$, along with the conventions $e^{-\infty} = 0$ and $-\log(0) = \infty$. $\mathcal{L}$ is isomorphic to the familiar real or probability semiring

$$\mathcal{R} = (\mathbb{R}_+, +, \times, 0, 1)$$

via the negative-log morphism. The *tropical semiring* is defined as

$$\mathcal{T} = (\mathbb{R} \cup \{-\infty, +\infty\}, \min, +, +\infty, 0)$$

where the $\min$-convention is used for the tropical addition. Note that $\mathcal{T}$ is idempotent and the natural order over $\mathcal{T}$

$$\forall a, b \in \mathbb{R} \cup \{-\infty, +\infty\}: \quad (\min\{a, b\} = a) \iff (a \leq b)$$

is a total order, the usual order of real numbers [18]. Alternatively one can use the $\max$-convention to obtain another idempotent semiring

$$\mathcal{T}' = (\mathbb{R} \cup \{-\infty, +\infty\}, \max, +, -\infty, 0)$$

over which the natural order is another total order, this time the reverse order of real numbers.

Section IV frequently employs special semiring structures defined on the Cartesian product of ordered sets. As a matter of fact, our index structure itself is a weighted transducer over the lexicographic semiring of three tropical semirings.

Definition 4: The *product semiring* of two partially ordered semirings $\mathcal{A} = (\mathbb{A}, \oplus_{\mathbb{A}}, \otimes_{\mathbb{A}}, \overline{0}_{\mathbb{A}}, \overline{1}_{\mathbb{A}})$ and $\mathcal{B} = (\mathbb{B}, \oplus_{\mathbb{B}}, \otimes_{\mathbb{B}}, \overline{0}_{\mathbb{B}}, \overline{1}_{\mathbb{B}})$ is defined as

$$\mathcal{A} \times \mathcal{B} = (\mathbb{A} \times \mathbb{B}, \oplus_{\times}, \otimes_{\times}, \overline{0}_{\mathbb{A}} \times \overline{0}_{\mathbb{B}}, \overline{1}_{\mathbb{A}} \times \overline{1}_{\mathbb{B}})$$

where $\oplus_{\times}$ and $\otimes_{\times}$ are component-wise operators, e.g., $\forall a_1, a_2 \in \mathbb{A}, b_1, b_2 \in \mathbb{B}$: $(a_1, b_1) \oplus_{\times} (a_2, b_2) = (a_1 \oplus_{\mathbb{A}} a_2, b_1 \oplus_{\mathbb{B}} b_2)$. The natural order over $\mathcal{A} \times \mathcal{B}$, given by

$$((a_1, b_1) \leq_{\times} (a_2, b_2)) \iff (a_1 \oplus_{\mathbb{A}} a_2 = a_1, b_1 \oplus_{\mathbb{B}} b_2 = b_1)$$

defines a partial order, known as the *product order*, even if $\mathcal{A}$ and $\mathcal{B}$ are totally ordered.

Definition 5: The *lexicographic semiring* of two partially ordered semirings $\mathcal{A}$ and $\mathcal{B}$ is defined as

$$\mathcal{A} * \mathcal{B} = (\mathbb{A} \times \mathbb{B}, \oplus_{*}, \otimes_{*}, \overline{0}_{\mathbb{A}} \times \overline{0}_{\mathbb{B}}, \overline{1}_{\mathbb{A}} \times \overline{1}_{\mathbb{B}})$$

where $\otimes_{*}$ is a component-wise multiplication operator and $\oplus_{*}$ is a lexicographic priority operator, $\forall a_1, a_2 \in \mathbb{A}, b_1, b_2 \in \mathbb{B}$

$$(a_1, b_1) \oplus_{*} (a_2, b_2) = \begin{cases} (a_1, b_1 \oplus_{\mathbb{B}} b_2) & a_1 = a_2 \\ (a_1, b_1) & a_1 = a_1 \oplus_{\mathbb{A}} a_2 \neq a_2 \\ (a_2, b_2) & a_1 \neq a_1 \oplus_{\mathbb{A}} a_2 = a_2 \end{cases}.$$

Unlike $\mathcal{A} \times \mathcal{B}$, the natural order over $\mathcal{A} * \mathcal{B}$, given by

$$((a_1, b_1) \leq_{*} (a_2, b_2)) \iff \begin{array}{c} (a_1 = a_1 \oplus_{\mathbb{A}} a_2 \neq a_2) \\ \text{or} \\ (a_1 = a_2 \text{ and } b_1 = b_1 \oplus_{\mathbb{B}} b_2) \end{array}$$

defines a total order, known as the *lexicographic order*, when $\mathcal{A}$ and $\mathcal{B}$ are totally ordered.

More generally, one can define the product and lexicographic orders (or semirings) on the Cartesian product of n ordered sets. Suppose $\{\mathbb{A}_1, \mathbb{A}_2, \ldots, \mathbb{A}_n\}$ is an n-tuple of sets, with respective total orderings $\{\leq_1, \leq_2, \ldots, \leq_n\}$. The product order $\leq_{\times}$ of $\{\mathbb{A}_1, \mathbb{A}_2, \ldots, \mathbb{A}_n\}$ is defined as

$$(a_1, a_2, \ldots, a_n) \leq_{\times} (b_1, b_2, \ldots, b_n) \iff a_i \leq_i b_i \quad \forall i \leq n.$$

Similarly, the lexicographic order $\leq_{*}$ of $\{\mathbb{A}_1, \mathbb{A}_2, \ldots, \mathbb{A}_n\}$ is defined as

$$(a_1, a_2, \ldots, a_n) \leq_{*} (b_1, b_2, \ldots, b_n) \iff \exists m > 0, a_i = b_i \quad \forall i < m, a_m \leq_m b_m.$$

That is, for one of the terms $a_m \leq_m b_m$ and all the preceding terms are equal.

We should also note that product (or lexicographic) semiring on $\{\mathbb{A}_1, \mathbb{A}_2, \ldots, \mathbb{A}_n\}$ can be recursively defined using the associativity of $\times$ (or $*$) operator

$$\mathcal{A}_1 \times \mathcal{A}_2 \times \cdots \times \mathcal{A}_n = ((\cdots(\mathcal{A}_1 \times \mathcal{A}_2) \times \cdots) \times \mathcal{A}_n).$$

B. Weighted Finite-State Automata

Definition 6: 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; Δ is the finite output alphabet; Q is a finite set of states; $I \subseteq Q$ is the set of initial states; $F \subseteq Q$ is the set of final states; $E \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times (\Delta \cup \{\varepsilon\}) \times \mathbb{K} \times Q$ is a finite set of arcs; $\lambda : I \to \mathbb{K}$ is the initial weight function; and $\rho : F \to \mathbb{K}$ is the final weight function.

A *weighted finite-state acceptor* $A = (\Sigma, Q, I, F, E, \lambda, \rho)$ is defined in a similar way by simply omitting the output labels. The size of an automaton[1] M is defined as $|M| = |Q| + |E|$.

Given an arc $e \in E$, we denote by $i[e]$ its input label, $o[e]$ its output label, $w[e]$ its weight, $p[e]$ its origin or previous state, and $n[e]$ its destination or next state. A path $\pi = e_1 \ldots e_k$ is an element of E^* with consecutive arcs satisfying $n[e_{i-1}] = p[e_i], i = 2, \ldots, k$. We extend n and p to paths by setting $n[\pi] = n[e_k]$ and $p[\pi] = p[e_1]$. The labeling and the weight functions can also be extended to paths by defining $i[\pi] = i[e_1] \ldots i[e_k]$, $o[\pi] = o[e_1] \ldots o[e_k]$ and $w[\pi] = w[e_1] \otimes \ldots \otimes w[e_k]$. We also extend w to any finite set of paths Π by setting

$$w[\Pi] = \bigoplus_{\pi \in \Pi} w[\pi].$$

We denote by $\Pi(q, q')$ the set of paths from q to q', by $\Pi(q, x, q')$ the set of paths from q to q' with input label $x \in \Sigma^*$ and by $\Pi(q, x, y, q')$ the set of paths from q to q' with input label $x \in \Sigma^*$ and output label $y \in \Delta^*$. These definitions can be extended to subsets $S, S' \subseteq Q$, by

$$\Pi(S, x, S') = \bigcup_{q \in S, q' \in S'} \Pi(q, x, q').$$

A *successful path* in an automaton M is a path from an initial state to a final state. A symbol sequence x is recognized by M if there exists a successful path π labeled with x on the input side. M is *unambiguous* if for any string $x \in \Sigma^*$ there is at most one successful path labeled with x on the input side. Thus, an unambiguous transducer defines a function.

[1]In finite automata literature, the terms *automaton* and *acceptor* are often used interchangeably. Throughout this paper, we use the term *automaton* when we do not differentiate between a transducer and an acceptor.

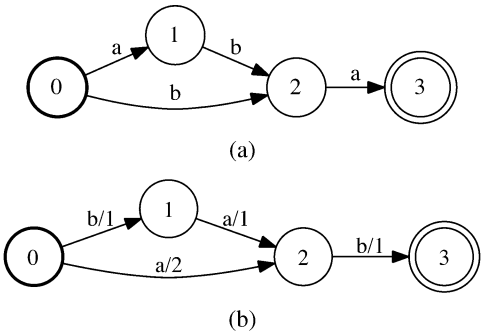

Fig. 1. Weighted automata (a) A_1 and (b) A_2 over the real semiring $\mathcal{R}$ along with the state timing lists $t_1 = [0, 1, 2, 3]$ and $t_2 = [0, 1, 2, 3]$.

The weight associated by a transducer T to any input-output string pair $(x, y) \in \Sigma^* \times \Delta^*$ is given by

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

and $[\![T]\!](x, y)$ is defined to be $\overline{0}$ when $\Pi(I, x, y, F) = \emptyset$.

C. Factor Automata

Definition 7: Given two strings u and v in Σ^*, v is a *factor (substring)* of u if $u = xvy$ for some x and y in Σ^*. More generally, v is a factor of a language $L \subseteq \Sigma^*$ if v is a factor of some string $u \in L$. The *factor automaton* $F(u)$ of a string u is the minimal deterministic finite-state acceptor recognizing exactly the set of factors of u.

$F(u)$ can be built in linear time and its size is linear in the size of the input string $|u|$ [19], [20]. We denote by $F(A)$ the minimal deterministic acceptor recognizing the set of factors of a finite acceptor A, that is the set of factors of the strings recognized by A. A recent work [12] showed that the size of the factor automaton $F(A)$ is linear in $|A|$ and provided an algorithm for the construction of $F(A)$ in linear time.

IV. Timed Factor Transducer of Weighted Automata

This section presents an algorithm for the construction of an efficient timed index for a large set of speech utterances. We propose a new factor transducer structure, timed factor transducer (TFT), which stores the timing information on the arc weights, thereby solving the issues associated with the non-deterministic factor transducer of [15]. For easy comparison, we follow the development in [2].

We assume that for each speech utterance u_i of the data-set in consideration $\{u_i \mid i = 1, \ldots, n\}$, a weighted automaton A_i over the log semiring with alphabet Σ (e.g., phone or word lattice output by ASR), and a list t_i of state timings are given. Fig. 1 gives examples of automata over the real semiring $\mathcal{R}$. The problem consists of creating a timed index that can be used for the direct search of any factor of any string accepted by these automata. Note that this problem crucially differs from the classical text indexing problems in that the input data is uncertain.

Our index construction algorithm is based on general weighted automata and transducer algorithms. The main idea is that the timed index can be represented by a weighted finite-state transducer T mapping each factor x to 1) the set of automata in which x appears, 2) start-end times of the intervals where x appears in each automaton, and 3) the posterior probabilities of x actually occurring in each automaton in the corresponding time interval. We start with preprocessing each input automaton to obtain a posterior lattice in which non-overlapping arc clusters are separately labeled. Then from each processed input automaton we construct an intermediate factor transducer which recognizes exactly the set of factors of the input. We convert these intermediate structures into deterministic transducers by augmenting each factor with a disambiguation symbol and then applying weighted automata optimization. Finally, we take the union of these deterministic transducers and further optimize the result to obtain a deterministic inverted index of the entire data-set. The following sections detail the consecutive stages of the algorithm.

A. Preprocessing

When the automata A_i are word/phone lattices output by an ASR system, the path weights correspond to joint probabilities assigned by the language and acoustic models. We can apply to A_i a general weight-pushing algorithm in the log semiring [21] which converts these weights into the desired $(-\log)$ posterior probabilities given the lattice. Since each input automaton A_i is acyclic, i.e., a lattice, the complexity of the weight-pushing algorithm is linear in the size of the input $(O(|A_i|))$.

The algorithm given by [2] generates a single index entry for all the occurrences of a factor in an utterance. This is the desired behavior for the SUR problem. In the case of STD, we would like to keep separate index entries for non-overlapping occurrences in an utterance since we no longer search for the utterances but the exact time intervals containing the query term. This separation can be achieved by clustering the arcs with the same input label and overlapping time-spans. The clustering algorithm is as follows. For each input label: 1) sort the collected (start time, end time) pairs with respect to end times; 2) identify the largest set of non-overlapping (start time, end time) pairs and assign them as cluster heads; and 3) classify the rest of the arcs according to maximal overlap. We effectively convert the input automaton to a transducer where each arc carries a cluster identifier on the output label. In other words, each (input label, output label) pair of the transducer designates an arc cluster. Note that the clustering operation does not introduce additional paths, i.e., it simply assigns each arc to an arc cluster. Fig. 2 illustrates the application of the preprocessing algorithm to the automata of Fig. 1.

B. Construction of the Timed Factor Transducer

Let $B_i = (\Sigma, \Delta, Q_i, I_i, F_i, E_i, \lambda_i, \rho_i)$ denote an ε-free transducer over the log semiring $\mathcal{L}$ obtained by applying the weight pushing and clustering algorithms (of the previous section) to the automaton A_i. The output string associated by B_i to each input string it accepts gives the string of cluster identifiers. The weight associated by B_i to each input–output string pair can be interpreted as the posterior probability of that pair for the utterance u_i given the models used to generate the automata. More generally, B_i defines an occurrence probability $P_i(x, y)$ for each string pair $(x, y) \in \Sigma^* \times \Delta^*$, where $P_i(x, y)$ is the probability of

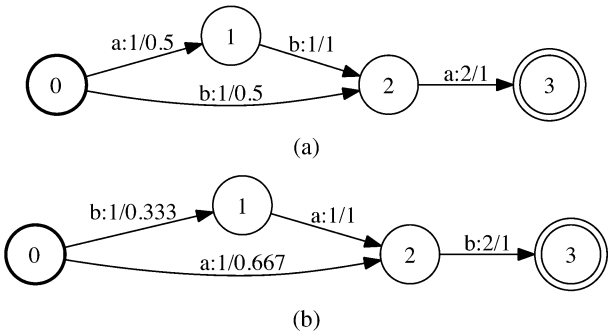

Fig. 2. (a) B_1 and (b) B_2 over the real semiring obtained by applying the pre-processing algorithm to the automata A_1 and A_2 in Fig. 1.

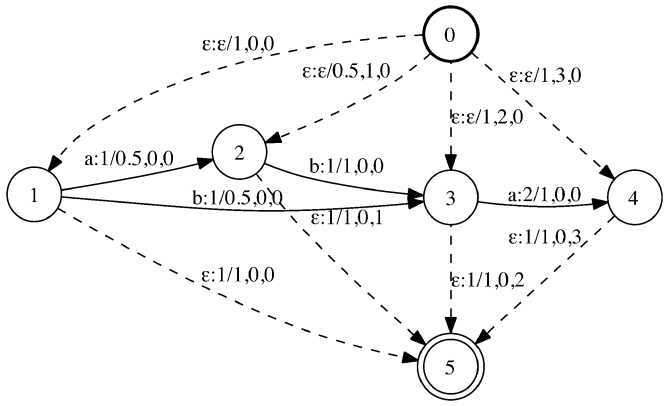

Fig. 3. Construction of T_1 from the weighted automaton B_1 in Fig. 2(a) and the state timing list $t_1 = [0, 1, 2, 3]$: after factor generation.

factor (x, y) given B_i. $P_i(x, y)$ is simply the sum of the probabilities of all successful paths in B_i that contain (x, y) as a factor. For each state $q \in Q_i$, we denote by $\alpha_i[q]$ the shortest distance from the initial states I_i to q ($= -\log$ of the forward probability) and by $\beta_i[q]$ the shortest distance from q to the final states F_i ($= -\log$ of the backward probability)

$$\alpha_i[q] = \bigoplus_{\pi \in \Pi(I_i, q)}^{\log} (\lambda_i(p[\pi]) + w[\pi]) \tag{1}$$

$$\beta_i[q] = \bigoplus_{\pi \in \Pi(q, F_i)}^{\log} (w[\pi] + \rho_i(n[\pi])). \tag{2}$$

The shortest distances $\alpha_i[q]$ and $\beta_i[q]$ can be computed for all states $q \in Q_i$ in linear time ($O(|B_i|)$) since B_i is acyclic [18]. Let Π_i denote the set of all paths (including partials) in B_i. Then, $P_i(x, y)$ is given by

$$-\log P_i(x, y) = \bigoplus_{\substack{i[\pi]=x, o[\pi]=y \\ \pi \in \Pi_i}}^{\log} \alpha_i[p[\pi]] + w[\pi] + \beta_i[n[\pi]]. \tag{3}$$

Note that since there is a unique occurrence of the factor pair (x, y) in the utterance u_i, $P_i(x, y)$ is a proper posterior probability even when there are multiple occurrences of x in u_i. Without the output symbols y, (3) would yield the expected count of x in u_i.

Let $t_i[q]$ denote the timing of state $q \in Q_i$, and $t_i^{s/e}(x, y)$ denote the start/end time of the factor (x, y) in B_i. Then,

$$t_i^s(x, y) = \min_{\substack{i[\pi]=x, o[\pi]=y \\ \pi \in \Pi_i}} t_i[p[\pi]] \tag{4}$$

$$t_i^e(x, y) = \max_{\substack{i[\pi]=x, o[\pi]=y \\ \pi \in \Pi_i}} t_i[n[\pi]]. \tag{5}$$

Equations (3)–(5) define the quantities we need to store for each factor. Now, we want to construct an index transducer which will map each factor (x, y) to a $(-\log P_i(x, y), t_i^s(x, y), t_i^e(x, y))$ triplet. We do this by first constructing a transducer which indexes each factor occurrence separately, i.e., each path corresponds to a factor occurrence and the weight of this path gives the corresponding (posterior probability, start time, end time) triplet. To obtain the mapping we are after, we optimize this transducer on the $\mathcal{L} \times \mathcal{T} \times \mathcal{T}'$ semiring so that overlapping factor occurrences are merged by adding their posterior probabilities in $\mathcal{L}$ with $\oplus_{\log}$ operation, start times in $\mathcal{T}$ with $\min$ operation and end times in $\mathcal{T}'$ with $\max$ operation. After overlapping factors are merged, we no longer need to work over $\mathcal{L} \times \mathcal{T} \times \mathcal{T}'$, so we switch to the more familiar $\mathcal{T} * \mathcal{T} * \mathcal{T}$ semiring (equivalent of tropical semiring on $\mathbb{R}^3$) which allows pruning and shortest path operations.

From the weighted transducer B_i over $\mathcal{L}$, and the state timing list t_i, one can derive a timed factor transducer T_i over $\mathcal{T} * \mathcal{T} * \mathcal{T}$ in four steps.

1) *Factor Generation*. In the general case we index all of the factors in the following way:
   - Map each arc weight (see Section III-A):
   $$w \in \mathcal{L} \to (w, \overline{1}, \overline{1}) \in \mathcal{L} \times \mathcal{T} \times \mathcal{T}'.$$
   - Create a unique initial state $q_I \notin Q_i$.
   - Create a unique final state $q_F \notin Q_i$.
   - $\forall q \in Q_i$, create two new arcs:
     — an initial arc: $(q_I, \varepsilon, \varepsilon, (\alpha_i[q], t_i[q], \overline{1}), q)$;
     — and a final arc: $(q, \varepsilon, i, (\beta_i[q], \overline{1}, t_i[q]), q_F)$.
2) *Factor Merging*. We merge the paths carrying the same factor-pair by viewing the result of factor generation as an acceptor, i.e., encoding input–output labels as a single label, and applying weighted ε-removal, determinization and minimization over the $\mathcal{L} \times \mathcal{T} \times \mathcal{T}'$ semiring. After the overlapping factors are merged, we map the arc weights
   $$(w_1, w_2, w_3) \in \mathcal{L} \times \mathcal{T} \times \mathcal{T}' \to (w_1, w_2, w_3) \in \mathcal{T} * \mathcal{T} * \mathcal{T}.$$
3) *Factor Disambiguation*. We remove the cluster identifiers on the non-final arcs and insert disambiguation symbols δ into the final arcs. For each edge $e \in E_i$:
   - if $n[e] \notin F_i$, then assign $o[e] = \varepsilon$;
   - if $n[e] \in F_i$, then assign $i[e] = \delta[e]$.
4) *Optimization*. The result of factor disambiguation can be optimized by viewing it as an acceptor and applying weighted determinization and minimization over the $\mathcal{T} * \mathcal{T} * \mathcal{T}$ semiring.

Figs. 3 and 4 illustrate the TFT construction from a weighted automaton and a list of state timings. Factor generation step creates an intermediate factor transducer which maps exactly the set of factors of B_i to the utterance ID i (possibly many

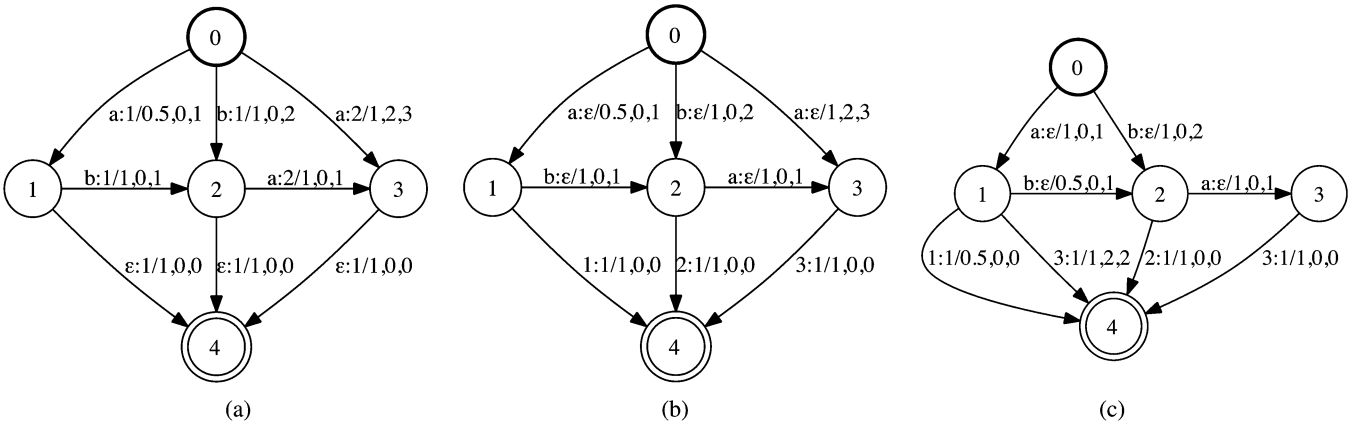

Fig. 4. Construction of T_1 from the weighted automaton B_1 in Fig. 2(a) and the state timing list $t_1 = [0, 1, 2, 3]$: (a) after factor merging over $\mathcal{R} \times \mathcal{T} \times \mathcal{T}'$; (b) after factor disambiguation; (c) after optimization over $\mathcal{T} * \mathcal{T} * \mathcal{T}$.

times with different weights). During factor merging, overlapping factor occurrences are reduced to a single path. Let $\tilde{T}_i$ denote the result of factor merging. It is clear from (3)–(5) that for any factor $(x, y) \in \Sigma^* \times \Delta^*$

$$[\![\tilde{T}_i]\!](x, yi) = (-\log P_i(x, y), t_i^s(x, y), t_i^e(x, y)) \qquad (6)$$

where yi denotes the concatenation of the string of cluster identifiers y and the automaton identifier i.

The intermediate transducer $\tilde{T}_i$ [Fig. 4(a)] is not deterministic over the input labels. To make it so, we remove the cluster identifiers and augment each path with a disambiguation symbol $\delta[e]$ by modifying the final transitions $e \in E_i, n[e] \in F_i$. It is convenient to use the previous state $p[e]$ (more precisely a symbol derived from it) as this auxiliary disambiguation symbol in a practical implementation, i.e., $\delta[e] = p[e]$. After this operation [Fig. 4(b)] each final transition carries the symbol of its origin state as an input label. These symbols make sure that non-overlapping factors labeled with the same input string are kept separate during optimization. The resulting transducer T_i [Fig. 4(c)] is deterministic over the input labels and includes the augmented paths, i.e., each path in T_i corresponds to an input–output factor pair in $\tilde{T}_i$.

The timed factor transducer T [Fig. 5(a)] of the entire data-set is constructed by:

- taking the union U of individual TFTs:

$$U = \bigcup_i T_i, \quad i = 1, \ldots, n$$

- encoding the input-output labels of U as a single label and applying weighted ε-removal, determinization, and minimization over the $\mathcal{T} * \mathcal{T} * \mathcal{T}$ semiring;
- and finally defining T as the transducer obtained after decoding the labels of U and removing the disambiguation symbols on the final transitions.

The final optimization step merges only the partial paths since there is no successful path shared between $T_i, i = 1, \ldots, n$—each successful path has the unique automaton identifier on its final output label. The natural order of $\mathcal{T} * \mathcal{T} * \mathcal{T}$, being a total order, allows pruning before or during the final optimization if needed. Fig. 5(a) illustrates the fully optimized TFT of the entire data-set. Even though this picture suggests that the TFT is nothing more than a prefix tree, this is not true in general.[2] As a matter of fact, this is exactly why this structure is a feasible index for a large collection of automata. Unlike a prefix tree, by allowing multiple incoming arcs, we are able to index exponentially many factors in a structure linear in the size of the input lattices (see Sections III-C and VI-A).

C. Factor Selection

Instead of the above given method of indexing each and every factor along with its time span and posterior probability, we can utilize factor selection filters in WFST form to restrict, transform or re-weight the index entries. [2] introduces various filters that are applied at various stages of the algorithm. Each filter is composed with some automaton, obtained in the course of the algorithm, to achieve a specific filtering operation.

One such filter is a pronunciation dictionary which maps words to phone sequences. This filter is applied to the word lattices to obtain phonetic lattices. In our case, applying such a filter warrants an update of the state timings accordingly.

Another example is a simple grammar which restricts the factors. This filter is applied after the factor generation step and removes the factors that are not accepted by the grammar. We utilize such a grammar to reject the silence symbol, i.e., factors including the silence symbol are not indexed.

D. Search Over the Timed Factor Transducer

The user query is typically an unweighted string, but it can also 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 R to a query X is another automaton obtained by:

- composing X with T on the input side [22] and projecting the resulting transducer onto its output labels;
- removing the ε transitions and finally sorting with the shortest-path algorithm.

R is a simple acceptor. Each successful path π in R is a single arc (from the initial state to one of the final states) which carries an automaton identifier on its label $i[\pi]$, and a $(-\log$

[2]Consider replacing B_2 with a simple transducer which has two states $\{0, 1\}$, a single arc $(0, \mathrm{a}, 1, 1, 1)$ and the state timing list $[0, 1]$. The resulting TFT (before removing the disambiguation symbols) would be the same as the transducer in Fig. 4(c), except for an additional arc $(1, 1, 2, (1, 0, 0), 4)$.

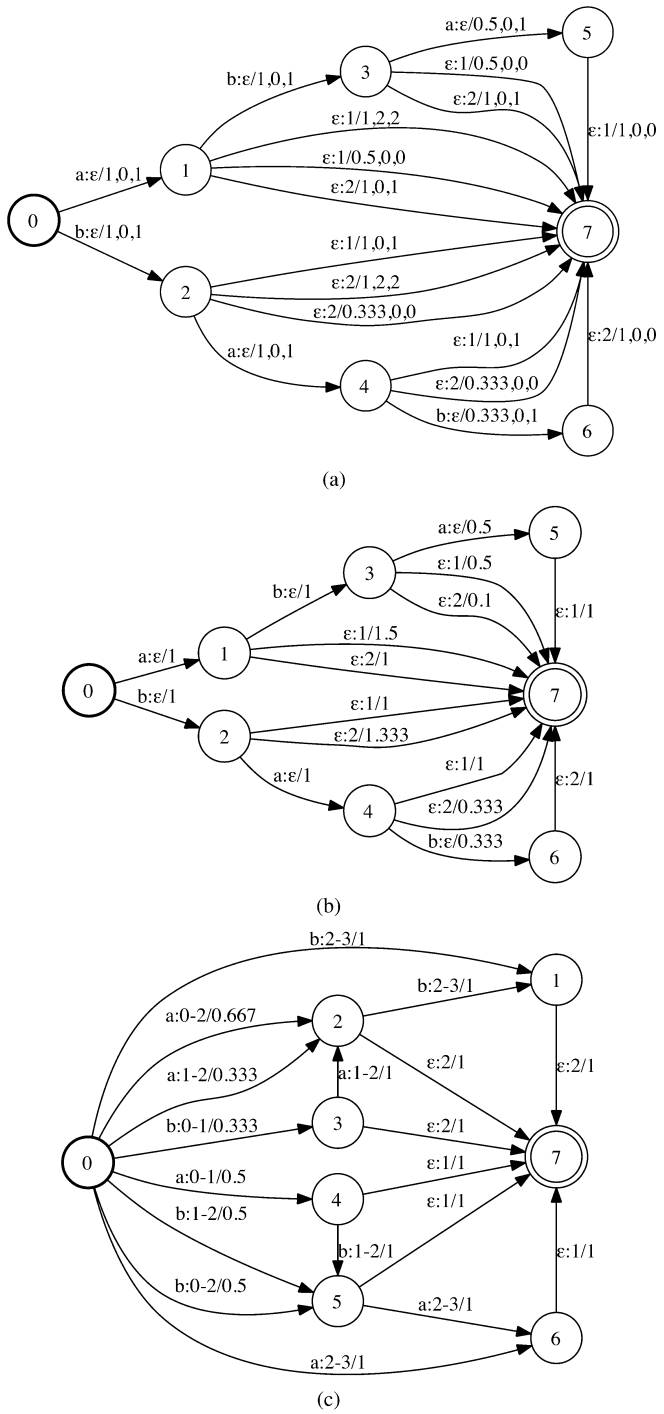

Fig. 5. (a) TFT T over $\mathcal{R} \times \mathcal{T} \times \mathcal{T}'$, (b) FT over $\mathcal{R}$, and (c) MFT over $\mathcal{R}$ obtained from the weighted automata and the state timing lists in Fig. 1. Output labels on the non-final arcs of the MFT represent the associated time intervals, i.e., "a:0-2" means there is an "a" from time 0 to 2.

posterior probability, start time, end time) triplet on its weight $w[\pi] \in \mathcal{T} * \mathcal{T} * \mathcal{T}$. A simple traversal over R in the arc order gives results sorted with the natural order of $\mathcal{T} * \mathcal{T} * \mathcal{T}$. Note that R can be pruned before traversal to retain only the most likely responses. The pruning threshold may be varied to achieve different operating points.

The full inverted index T is search-time optimal since it is a deterministic transducer except for the final ε transitions which have automaton identifiers on the output. Assuming we can access any arc of T (that originates from a given state and matches a given input label) in constant time, the search complexity for a string query x is given as $O(|x| + r)$, where r is the number of results, i.e., the number of arcs in R.

TABLE I
BREAKDOWN OF TBN DATABASE (IN HOURS)

T (Training)	H (Held-out)	R (Retrieval)	All
184.0	3.1	163.3	350.4

E. Comparison With the Factor Transducer and the Modified Factor Transducer

For easy comparison, Fig. 5(b) and (c) gives the FT [2] and the MFT [15] obtained from the automata of Fig. 1. Structurally, the FT is very similar to the TFT. The major difference is that the FT does not store any timing information. The MFT, on the other hand, is quite different from both the FT and the TFT. Timing information is encoded in the output labels, i.e., each output label on a non-final arc represents a time interval. In Section II, we pointed out the issues related to both structures. The proposed method alleviates the issues of the FT by indexing the timing information and keeping separate entries for non-overlapping factors—note the extra final transitions of the TFT. Issues of the MFT, on the other hand, are resolved by embedding the timing information into the arc weights. Once the cluster identifiers are removed, the final TFT can be made fully deterministic except for the final transitions. Also note that we no longer have the quantization problem which was a by-product of keeping timing labels.

V. EXPERIMENTAL SETUP

In this study, we present results on two different STD systems, one of them in Turkish and the other in English. Both systems utilize IBM's Attila Speech Recognition Toolkit [23] and our OpenFst [17] based STD tools. Following sections detail the ASR training and STD experimentation data used in each system.

A. Turkish Broadcast News (TBN) STD System

Boğaziçi University Speech Processing Group has been collecting a large database of Turkish Broadcast News since 2006. Currently, TBN database includes 350 hours of manually transcribed speech data collected from one radio (VoA) and four TV channels (CNN Türk, NTV, TRT1, TRT2). In this study, we used non-overlapping subsets of the TBN database (given in Table I) for building ASR systems and performing STD experiments.

Our STD system utilizes the T, H, and R subsets of the TBN database for ASR training, ASR optimization and STD experiments, respectively. This system is meant to mimic a realistic scenario where a large database of spoken documents is indexed and searched. R subset, which includes 1.2 M words over 163 hours of speech, constitutes a fairly large evaluation set for speech retrieval experiments.

The ASR engine was built with the IBM Attila toolkit using the T subset. It is a word-based system with a vocabulary of

TABLE II
R-IV QUERY SET DECOMPOSITION

1-word	2-word	3-word	4-word	Total
2312	1725	256	115	4408

TABLE III
DRYRUN06-IV QUERY SET DECOMPOSITION (W.R.T. PHONETIC LENGTH)

1	2	3	4	5	6	7	8	9	10	11+
3	36	107	125	115	110	79	83	83	51	266

200 K words. The language model (200 K word-based model presented in [24]) was derived from the manual transcripts of the T subset and a large text corpus of size 184 M words [25]. The WER of the ASR system on the H and R subsets are 25.9% and 29.9%, respectively.

In STD experiments, we used the R-IV query set which was selected from the reference transcriptions of the R subset. R-IV query terms are confined to the ASR vocabulary. Table II gives the decomposition of R-IV query set with respect to query length.

B. English Broadcast News (EBN) STD System

This system uses standard data sets from NIST's 2006 Spoken Term Detection Evaluation [13]. Experiments utilize the Broadcast News subset of the STDDEV06 data set, which includes 25 K words over 3 hours of speech, and the IV (in-vocabulary) subset of the DRYRUN06 query set which includes 1058 terms. Table III gives the decomposition of the DRYRUN06-IV query set with respect to the phonetic query length. The ASR engine is the one used by IBM during the 2006 NIST STD evaluations [4]. Architectural details of the IBM research prototype ASR system can be found in [23]. The WER of the ASR system on STDDEV06 data set is 12.7%.

VI. EXPERIMENTS

In this section, we provide experiments comparing three STD schemes: Two-Stage Retrieval with FT [15], Retrieval with MFT [15], and Retrieval with TFT. Our comparisons are in terms of index size and average search time. We also analyze the change of average search time w.r.t. query length (only for the last two schemes).

For the experiments of this section, we first extracted a fairly large word lattice (five back-pointers per word trace) for each utterance. Then, we pruned the raw lattices with different logarithmic beam widths and conducted the same set of experiments for each beam width. We observed that for all STD schemes in consideration, a beam width of 4 is ideal for actual system operation, i.e., minimal index size and search time, without incurring a significant loss in retrieval performance. We should note that there is no significant difference between the three STD schemes as far as the term detection performance is concerned (Actual Term Weighted Value [13] 0.81 for the word-based TBN system and 0.80 for the word-based/phonetic EBN STD systems).

NIST STD 2006 Evaluation Plan requires the results to contain no more than 0.5 s gap between the adjacent words of a query term. We exploit this requirement by indexing only the factors that do not contain such gaps. We process input lattices to identify long gaps (> 0.5 s) and use a silence symbol to mark them. Then after the factor generation step, we employ a simple restriction grammar to filter out the factors including the silence symbol. Shorter gaps are mapped to ε symbols and removed prior to index construction.

TABLE IV
NUMBER OF FACTORS AND INDEX SIZE VERSUS TOTAL LATTICE SIZE (STDDEV06 DATA SET, BOLD COLUMN INDICATES BEAM WIDTH 4)

$\log_{10}$(Total Lattice Size)	5.64	**6.22**	6.61	6.85	7.01
$\log_{10}$(TFT Size)	5.58	**6.40**	7.16	7.66	7.87
$\log_{10}$(Number of Factors)	12.67	**20.06**	25.24	29.34	32.38

A. Index Size

In retrieval applications, index size is an important application concern. Preferably, it should be as small as possible but maybe even more importantly it should not grow exponentially as the data size increases. In our case, the data size depends on the total amount of speech and the beam width of ASR lattices used in index construction. Table IV demonstrates how fast the number of factors increases as we increase the beam width even with a small data set like STDDEV06.

Fig. 6 plots the increase of index size w.r.t. the size of input lattices—lattice beam increases from 1 to 10. As pointed out in Section III-C, the size of factor automata are linear in the size of input lattices. Fig. 6 demonstrates this linear dependence for all structures in consideration. Note that MFT has the smallest size at all beam widths. (For our data set it is even smaller than the total size of input lattices!) While this may appear rather unexpected—since FT carries much less information compared to MFT—, we should not forget that MFT is not a deterministic machine. Before the final optimization step of the index construction algorithm, there are much less common paths to merge in the case of MFT due to the time labels. This leads to a smaller final index since the lattice structure of the index is largely preserved after determinization. The size difference between FT and TFT, on the other hand, can be attributed to two key features of TFT: 1) storage of the time alignment information on the path weights and 2) separation of non-overlapping factors via clustering.

B. Search Time

The most important concern in a retrieval application is the search time since search is usually performed online unlike index construction. Table V gives the per query average search times for a lattice beam of 4.

Due to the costly second stage, employing the FT in the STD task results in two to three orders of magnitude slower search. The MFT and the TFT give similar search time performances as far as the whole query sets are concerned. Since most of the query terms comprise of a single word, this behavior is not surprising. Even though the MFT is not a deterministic machine, it has a much smaller memory footprint compared to the TFT,

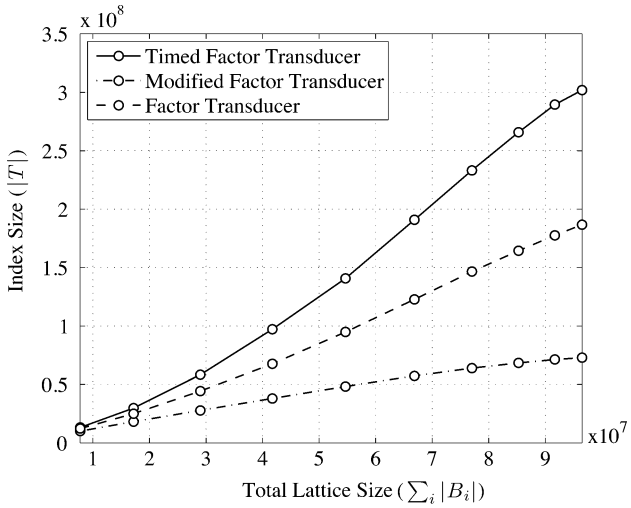

Fig. 6. Index size versus total lattice size (TBN-R Data Set).

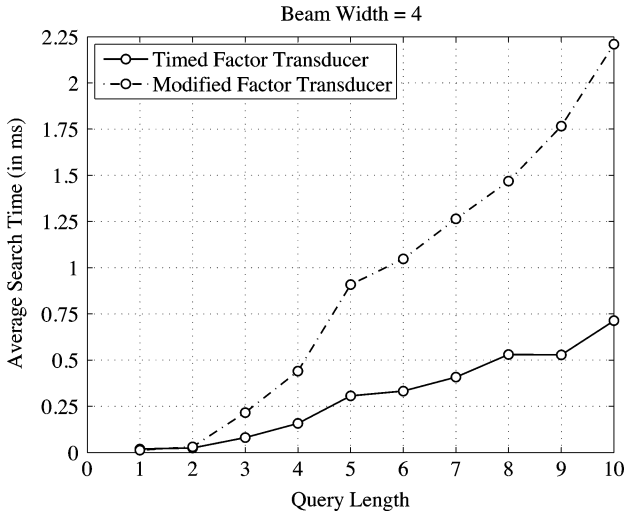

Fig. 7. Per result average search times versus phonetic query length (Phonetic STDDEV06 Data Set, Phonetic DRYRUN06-IV Query Set, Beam Width 4).

TABLE V
PER QUERY AVERAGE SEARCH TIMES (IN MILLISECONDS) (BEAM WIDTH 4)

	TBN System	EBN System
FT (2-Stage)	202.71	54.67
Modified FT	4.01	0.24
Timed FT	4.53	0.22

TABLE VI
PER QUERY AVERAGE SEARCH TIMES (IN MILLISECONDS) W.R.T. QUERY LENGTH (TBN-R DATA SET, R-IV QUERY SET, BEAM WIDTH 4)

	1-word	2-word	3-word	4-word	All
Modified FT	6.45	1.34	1.21	1.22	4.01
Timed FT	8.16	0.58	0.27	0.24	4.53

TABLE VII
PER RESULT AVERAGE SEARCH TIMES (IN MILLISECONDS) W.R.T. QUERY LENGTH (TBN-R DATA SET, R-IV QUERY SET, BEAM WIDTH 4)

	1-word	2-word	3-word	4-word
Modified FT	0.02	0.34	0.66	0.87
Timed FT	0.02	0.09	0.15	0.19

and hence we obtain comparable search times. While non-determinism does not really matter when the query is a single word, it becomes more and more important as the queries get longer. Recall that the TFT has a search time complexity linear in the sum of the query length and the number of results. The MFT, on the other hand, has an average search time complexity linear in the product of the query length and the number of results—worst case complexity is exponential. Table VI gives per query average search times over the TBN-R data set w.r.t. query length.

First thing to notice about Table VI is that the MFT is faster than the TFT only when the query is a single word. As the query gets longer, the TFT becomes much faster due to its deterministic structure. In the MFT, every distinct time alignment of a word leads to a new path to be traversed. Thus, each path matching the beginning of the query has to be traversed until a mismatch is found to determine the successful paths matching the whole query. On the other hand, in the TFT, only one arc—hence only one path—needs to be traversed at each state until the single partial path matching the query is found. After that, results are read from the final transitions leaving the single destination state of that partial path.

Table VII gives per result average search times w.r.t. query length over the TBN-R data set. Our search time analysis in Section IV-D assumed that we could access any arc (given a state and a label) at constant time. However, our OpenFst based implementation keeps a list of arcs for each state rather then a hash map, i.e., access time is $O(\log D)$, where D represents the average out-degree. Even though it is not possible to observe an exact linear dependence on the query length, we can clearly observe the difference between the two methods. Fig. 7 is particularly interesting since it compares the two methods in a phonetic STD setting. To obtain these graphs, we converted the EBN word lattices into phone lattices using the pronunciation dictionary of the ASR system and constructed phonetic indexes. Once mapped to their phonetic counterparts, we obtained longer query strings for search. As demonstrated by the graphs of Fig. 7, in a sub-word scenario a deterministic index is crucial for high performance.

VII. CONCLUSION

Efficient indexing of ASR lattices (word or sub-word level) for STD is not a straightforward task. In this paper, we generalized the SUR indexing method of [2] by augmenting the index with timing information and used the resulting structure to perform single-stage STD. Proposed index structure is deterministic; hence, the search complexity is linear in the query length. As demonstrated by the comparisons given in Section VI-B, single-stage STD schemes significantly improve the search time over the two-stage scheme. We further analyzed the differences between the single-stage methods and demonstrated that the TFT significantly outperforms the MFT as the query length increases. This fact becomes even more valuable in the case of sub-word indexing due to longer query strings.

We presented the core index construction algorithm for the general problem of indexing lattices, but it can also be used

to index approximate structures like CNs or PSPLs. Since approximate structures include the posterior scores and the clustering information by construction, we no longer need the preprocessing step. The resulting index stores the timings/positions of the nodes in the case of CNs/PSPLs.

Since the TFT inherently stores the proximity information (by means of time alignments), it can be utilized in other SR applications like spoken document retrieval. Furthermore, since the query can be any weighted automaton, we can search for complex relations between query words without changing the index. Any finite-state relation, e.g., a regular expression, can be compiled into a query automaton and retrieved from the index. We gave an example to this type of search in [15] where we compiled weighted pronunciation alternatives into a query automaton to search the index for the OOV term occurrences. Another possibility is to relax the exact string match objective and allow for gaps between query terms. Although this is not expected in STD, it might be useful in other speech retrieval applications. Implementing such a search is trivial in our framework since we can easily modify the query automaton in such a way that an arbitrary number of words can be inserted between actual query terms. Searching for arbitrary permutations of query words, even allowing the insertion of other words in between these permutations, is yet another trivial extension which can be achieved without changing the index.

ACKNOWLEDGMENT

The authors would like to thank B. Ramabhadran and A. Sethy for providing the ASR lattices used in the EBN system, IBM for the Attila ASR toolkit [23], the developers of the OpenFst Library [17] for making it publicly available, and the reviewers of the first draft of this paper for their invaluable input. The software used for obtaining the results in this paper can be found at: http://busim.ee.boun.edu.tr/~speech/stdtools.html

REFERENCES

[1] M. Saraçlar and R. Sproat, "Lattice-based search for spoken utterance retrieval," in *Proc. HLT-NAACL*, Boston, MA, 2004, pp. 129–136.
[2] C. Allauzen, M. Mohri, and M. Saraçlar, "General indexation of weighted automata: Application to spoken utterance retrieval," in *Proc. HLT-NAACL Workshop Interdisciplinary Approaches Speech Indexing Retrieval*, Boston, MA, 2004, pp. 33–40.
[3] O. Siohan and M. Bacchiani, "Fast vocabulary independent audio search using path based graph indexing," in *Proc. Interspeech/Eurospeech*, 2005, pp. 53–56.
[4] J. Mamou, B. Ramabhadran, and O. Siohan, "Vocabulary independent spoken term detection," in *Proc. ACM SIGIR*, 2007, pp. 615–622.
[5] U. Chaudhari and M. Picheny, "Improvements in phone based audio search via constrained match with high order confusion estimates," in *Proc. ASRU*, 2007, pp. 665–670.
[6] C. Chelba and A. Acero, "Position specific posterior lattices for indexing speech," in *Proc. ACL*, 2005, pp. 443–450.
[7] Z. Y. Zhou, P. Yu, C. Chelba, and F. Seide, "Towards spoken-document retrieval for the internet: Lattice indexing for large-scale web-search architectures," in *Proc. HLT-NAACL*, 2006, pp. 417–422.
[8] M. Mohri, F. Pereira, and M. Riley, "Weighted finite-state transducers in speech recognition," *Comput. Speech Lang.*, vol. 16, no. 1, pp. 69–88, 2002.
[9] D. Gusfield, *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology*. Cambridge, U.K.: Cambridge Univ. Press, 1997.
[10] M. Crochemore and W. Rytter, *Jewels of Stringology*. River Edge, NJ: World Scientific, 2003.
[11] A. Blumer, J. Blumer, D. Haussler, R. McConnell, and A. Ehrenfeucht, "Complete inverted files for efficient text retrieval and analysis," *J. ACM*, vol. 34, no. 3, pp. 578–595, Jul. 1987.
[12] M. Mohri, P. Moreno, and E. Weinstein, "General suffix automaton construction algorithm and space bounds," *Theoret. Comput. Sci.*, vol. 410, no. 37, pp. 3553–3562, 2009.
[13] NIST, "The spoken term detection (STD) 2006 evaluation plan," 2006. [Online]. Available: http://www.itl.nist.gov/iad/mig/tests/std/
[14] S. Parlak and M. Saraçlar, "Spoken term detection for Turkish broadcast news," in *Proc. ICASSP*, Las Vegas, NV, 2008, pp. 5244–5247.
[15] D. Can, E. Cooper, A. Sethy, C. White, B. Ramabhadran, and M. Saraçlar, "Effect of pronunciations on OOV queries in spoken term detection," in *Proc. ICASSP*, Taipei, Taiwan, 2009, pp. 3957–3960.
[16] W. Kuich and A. Salomaa, *Semirings, Automata, Languages*. New York: Springer-Verlag, 1986.
[17] C. Allauzen, M. Riley, J. Schalkwyk, W. Skut, and M. Mohri, "OpenFst: A general and efficient weighted finite-state transducer library," in *Proc. 9th Int. Conf. Implementat. Applicat. Automata, (CIAA'07)*, 2007, vol. 4783, pp. 11–23. [Online]. Available: http://www.openfst.org, ser. LNCS
[18] M. Mohri, "Semiring frameworks and algorithms for shortest-distance problems," *J. Automata, Lang. Combinatorics*, vol. 7, no. 3, pp. 321–350, 2002.
[19] A. Blumer, J. Blumer, A. Ehrenfeucht, D. Haussler, M. T. Chen, and J. Seiferas, "The smallest automaton recognizing the subwords of a text," *Theoret. Comput. Sci.*, vol. 40, pp. 31–55, 1985.
[20] M. Crochemore, "Transducers and repetitions," *Theoret. Comput. Sci.*, vol. 45, no. 1, pp. 63–86, 1986.
[21] M. Mohri, "Finite-state transducers in language and speech processing," *Comput. Linguist.*, vol. 23, no. 2, pp. 269–311, 1997.
[22] M. Mohri, F. C. N. Pereira, and M. Riley, "Weighted automata in text and speech processing," in *Proc. ECAI, Workshop Extended Finite State Models of Lang.*, 1996.
[23] H. Soltau, B. Kingsbury, L. Mangu, D. Povey, G. Saon, and G. Zweig, "The IBM 2004 conversational telephony system for rich transcription," in *Proc. ICASSP*, 2005, pp. 205–208.
[24] E. Arısoy, D. Can, S. Parlak, H. Sak, and M. Saraçlar, "Turkish broadcast news transcription and retrieval," *IEEE Trans. Audio, Speech, Lang. Process.*, vol. 17, no. 5, pp. 874–883, Jul. 2009.
[25] H. Sak, T. Güngör, and M. Saraçlar, "Turkish language resources: Morphological parser, morphological disambiguator and web corpus," in *Proc. GoTAL*, 2008, vol. 5221, pp. 417–427, ser. LNCS, Springer.

Doğan Can received the B.S. and M.S. degrees from the Electrical and Electronics Engineering Department, Boğaziçi University, Istanbul, Turkey, in 2006 and 2010, respectively. He is currently pursuing the Ph.D. degree in computer science at the University of Southern California, Los Angeles.

His research interests fall under the domain of speech and language processing, particularly speech recognition, finite-state transducers, and spoken term detection.

Murat Saraçlar received the B.S. degree from the Electrical and Electronics Engineering Department, Bilkent University, Ankara, Turkey, in 1994 and the M.S.E. and Ph.D. degrees from the Electrical and Computer Engineering Department, Johns Hopkins University, Baltimore, MD, in 1997 and 2001, respectively.

He is currently an Associate Professor at the Electrical and Electronic Engineering Department, Boğaziçi University, Istanbul, Turkey. From 2000 to 2005, he was with the Multimedia Services Department at the AT&T Labs—Research. His main research interests include all aspects of speech recognition, its applications, as well as related fields such as speech and language processing, human–computer interaction and machine learning.

Dr. Saraçlar was a member of the IEEE Signal Processing Society Speech and Language Technical Committee (2007–2009). He is currently serving as an Associate Editor for the IEEE SIGNAL PROCESSING LETTERS and he is on the editorial board of *Computer Speech and Language*.