Lexical Prefix Tree and WFST: A Comparison of Two Dynamic Search Concepts for LVCSR

David Rybach, *Member, IEEE*, Hermann Ney, *Fellow, IEEE*, and Ralf Schlüter, *Member, IEEE*

***Abstract*—Dynamic network decoders have the advantage of significantly lower memory consumption compared to static network decoders, especially when huge vocabularies and complex language models are required. This paper compares the properties of two well-known search strategies for dynamic network decoding, namely history conditioned lexical tree search and weighted finite-state transducer-based search using on-the-fly transducer composition. The two search strategies share many common principles like the use of dynamic programming, beam search, and many more. We point out the similarities of both approaches and investigate the implications of their differing features, both formally and experimentally, with a focus on implementation independent properties. Therefore, experimental results are obtained with a single decoder by representing the history conditioned lexical tree search strategy in the transducer framework. The properties analyzed cover structure and size of the search space, differences in hypotheses recombination, language model look-ahead techniques, and lattice generation.**

***Index Terms*—Beam search, history conditioned lexical tree (HCLT) search, LVCSR, weighted finite-state transducer (WFST).**

I. Introduction

EFFICIENT search strategies for automatic speech recognition (ASR) have undergone intensive research since the early stages of automatic speech processing research. Various search techniques have been developed, investigated, and constantly enhanced. Nevertheless, efficient search (or decoding) for ASR continues to be a challenging research problem, in particular for huge vocabularies and complex language models, which are required for applications with unconstrained domain, morphologically rich languages, and spontaneous speech input. The term *efficiency* refers here to both computation time and memory usage.

The decoding process has to search an enormous set of hypotheses for the observed audio signal, incorporating the various types of knowledge sources. The involved statistical models, and hence the search space can be represented as finite-state networks.

This network can be pre-compiled in advance and used as a static structure by the decoder. Such static network decoders are known to be very efficient in terms of computation time, but have high memory requirements, especially if large language models (e.g., high-order n-gram models) and very large vocabularies are involved. Furthermore, the construction of these static networks is a time-consuming process, which is difficult to reconcile with frequently or dynamically changing models. In contrast, dynamic network decoders generate just the required parts of the search network on demand during the search [1]. The combination of models is performed on-the-fly as needed, which consumes significantly less memory [2].

In this paper, we study two state-of-the-art search strategies for dynamic network decoding, namely the history conditioned lexical tree (HCLT) strategy and a strategy based on weighted finite-state transducers (WFST). A general overview of search strategies for ASR can be found in [3].

Strictly speaking, the WFST framework does not define or dictate a certain search strategy. The transducer formalism allows to represent structures and algorithms occurring in many natural language-processing applications in a formal mathematical way. A search strategy is more or less independent of the network representation though. However, the term *WFST-based decoding* is used in numerous publications and is often associated with a particular search strategy (described in Section II).

In [4], Pereira, Riley, and Sproat proposed the use of WFST for speech recognition and described the construction of search networks by transducer composition. Following papers by the group at AT&T detailed the construction [5] and introduced further improvements for language model representation [6], context dependent modeling [7], network optimization [8], [9], and weight distribution [10]. A compendium can be found in [11]. The search strategy for these static network decoders is mostly defined by the way the search network is constructed.

Several approaches for the WFST-based dynamic construction of the search space—already mentioned in [5]—have been developed [2], [12]–[16]. Each of them implies a particular variant of the search strategy for dynamic network expansion. We will consider the most generic form as described in [2] in this paper and review the other methods briefly.

In order to avoid confusion of terms between the WFST framework and the search strategy of WFST-based dynamic network decoders, we denote the latter as *dynamic transducer composition* (DTC) search.

The HCLT search strategy, also known as word conditioned tree search, was designed for dynamic network decoding ab

Manuscript received September 13, 2012; revised December 20, 2012; accepted February 03, 2013. Date of publication February 25, 2013; date of current version March 13, 2013. This work was supported in part by the Google Research Awards Program and was partly realized as part of the Quaero Programme, funded by OSEO, French State agency for innovation. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Murat Saraclar.

D. Rybach is with Google Inc., New York, NY 10011 USA (e-mail: rybach@cs.rwth-aachen.de).

H. Ney and R. Schlüter are with the Human Language Technology and Pattern Recognition Group, Computer Science Department, RWTH Aachen University, 52056 Aachen, Germany (e-mail: ney@cs.rwth-aachen.d; schlueter@cs.rwth-aachen.de).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TASL.2013.2248723

initio [17], [18]. Not surprisingly, HCLT and DTC search strategies share many common principles like the use of dynamic programming, beam search, and many more. Differences have been analyzed in [19]–[21] for static WFST search networks and in [22] for dynamic WFST networks.

Most of the experimental comparisons of decoding strategies focus on runtime and memory efficiency, which obviously depend on a particular implementation. In contrast, this paper aims at comparing the search strategies as implementation-independent as possible. We focus therefore on the analysis of the search space, both theoretically and in experiments. Experimental results are obtained using a single implementation where possible. This is achieved by expressing the HCLT search in the WFST framework. The goal of this work is to reveal similarities of both search strategies and to analyze the implications of their differences.

In summary, the novel contributions of this paper are:

- A detailed comparative analysis of the HCLT and the DTC search strategy.
- The specification and implementation of the HCLT search in the WFST framework.
- An experimental analysis of the impact of those properties in which the two strategies differ.

The experimental work is performed with our large vocabulary speech recognition systems based on RASR [23] and OpenFst [24].

The remainder of this paper is organized as follows. Section II reviews the HCLT and DTC search strategies. The comparison is presented in Section III and fleshed out experimentally in Section V. The WFST-based HCLT search is described in Section IV. The findings are summarized in Section VI.

II. Search Strategies

Prior to describing the search strategies to be analyzed in detail, we briefly review the basic principles of the decoding process in order to introduce the notation and concepts used in this paper.

In the search process, the most likely word sequence $w_1^N = w_1 \dots w_N$ for the observed speech signal, represented by a sequence of acoustic feature vectors x_1^T, is computed:

$$x_1^T \mapsto \hat{w}_1^N(x_1^T) = \underset{w_1^N}{\arg\max} \left\{ p(w_1^N) \cdot p(x_1^T|w_1^N) \right\}. \qquad (1)$$

Two stochastic models appear in this equation: the language model (LM) assigns a prior probability $p(w_1^N)$ to the word sequence, whereas the acoustic model yields the conditional probability $p(x_1^T|w_1^N)$ of observing a sequence of features for the given word sequence.

The LM, assuming an n-gram LM, is factored into a product of conditional probabilities $p(w_i|h_i)$ with word history $h_i = w_{i-n+1} \dots w_{i-1} : p(w_1^N) = \prod_{i=1}^N p(w_i|h_i)$. In Sections III–VI we assume a backoff structured LM [25] with probabilities of the form

$$p(w|h) = \begin{cases} \alpha(w,h) & \text{if } N(w,h) > 0 \\ \gamma(h) \cdot p(w|\bar{h}) & \text{else} \end{cases} \qquad (2)$$

with $N(w,h)$ the count of n-gram hw in the LM training data, backoff weight $\gamma(h)$, and modified distribution $\alpha(w|h)$. α and γ are estimated on the training data by applying a smoothing technique. By $\bar{h}$ we denote a truncated history, obtained by dropping the leftmost word in h. Note that an interpolated LM can be represented in the same form.

The word models are assembled from sequences of phoneme models according to a pronunciation lexicon. Each phoneme is modeled (possibly depending on the phoneme context) by a hidden Markov model (HMM) with transition probabilities $p(s_t|s_{t-1}, w)$ and emission probabilities $p(x_t|s_t, w)$. By making use of the maximum (or Viterbi) approximation [26], the optimization problem (1) can be written as

$$\hat{w}_1^N = \underset{w_1^N}{\arg\max} \left\{ p(w_1^N) \cdot \max_{s_1^T} \prod_{t=1}^T p(x_t, s_t|s_{t-1}, w_1^N) \right\} \qquad (3)$$

with $p(x_t, s_t|s_{t-1}, w_1^N) = p(s_t|s_{t-1}, w_1^N) \cdot p(x_t|s_t, w_1^N)$. The optimization over the state sequence s_1^T is performed according to the HMM state network implied by the word sequence w_1^N.

The optimization problem is solved by applying a dynamic programming algorithm, which generates and evaluates partial hypotheses in a strictly left-to-right time-synchronous fashion [18]. The optimization over both word sequences and state sequences is performed simultaneously. The maximum approximation allows to recombine partial hypotheses at both state and word-level.

The knowledge sources, i.e., the LM, the pronunciation lexicon, and the context dependent phoneme models, can be jointly represented as a labeled weighted directed graph or finite-state network, denoted as *search network* in this paper. This network is independent of the acoustic observations. Due to the limited context dependency of the models, paths in the network can be merged at certain points. For example, for an n-gram LM, it suffices to separate paths by $(n-1)$ predecessor words. The actual structure of the search network depends on the applied search strategy. We will describe the structural differences in Sections III–VI.

In dynamic search methods, only parts of the search network are generated on demand as needed. A smaller network encoding solely the acoustic and lexical models is compiled in advance, while the LM is incorporated on-the-fly during search. This dynamic network approach has the advantage that the memory requirements of the decoder are feasible even with complex LMs and huge vocabularies. The low memory consumption comes along with decreased runtime performance compared to decoders using a pre-compiled static search network, due to the additional computational cost for integrating the LM. Furthermore, the construction of static networks can optimize the structure and weight distribution of the network to a larger extent.

We strictly distinguish here between the search network and the search space. The *search space* is the domain of the optimization problem for a certain acoustic observation, which comprises all time aligned HMM state sequences (including emission probabilities) and the corresponding word sequences. In other words, the search space is the network unfolded along the time axis ("trellis") with incorporated acoustic model probability. The search space can also be seen as the (re-weighted) set of all paths of a given length in the search network.

The dynamic programming approach for an abstract search network can be formulated using the quantity $Q(t, s)$, which is the score of the best partial path (in the search space) that ends at time t in state s of the search network. Independent of the search strategy, this quantity can be computed as

$$Q(t,s) = \max_{e=(s',l,s)\in E} \{\phi(x_t, e) \cdot Q(t-1, s')\} \tag{4}$$

where $E \subseteq S \times \Sigma \times S$ is the set of arcs in the search network with states S and labels Σ. The function $\phi(x_t, e)$ defines the score of an arc e for a given acoustic feature x_t. This score combines transition and emission probabilities from the HMM as well as LM probabilities. The definition of ϕ depends on the search strategy.

In order to reconstruct the best path after processing the complete utterance, each state hypothesis (t, s) is associated with a backpointer to keep track of local decisions. The actual definition and usage of backpointers depends on the search strategy.

For most tasks, an exhaustive search over the complete search space is prohibitive. Therefore, the application of beam search, i.e., a data-driven time-synchronous pruning of hypotheses is crucial for an efficient search. Beam search focusses the search on those hypotheses that are likely to result in the best path. The time-synchronous expansion of hypotheses allows to compare the scores $Q(t, \cdot)$ of all hypotheses at time frame t, since they cover the same part of the input. The part of the search space generated for a certain acoustic observation as result of the beam search is referred to as *actual search space* in contrast to the *potential search space* containing all possible hypotheses.

A. History Conditioned Lexical Tree (HCLT) Search

For the history conditioned lexical tree search strategy, the pronunciation lexicon is compiled into a lexical prefix tree [17]. The tree can be constructed at different levels of detail and size. If the arcs are labeled with context dependent phoneme models, the decoder has to expand the associated HMM structure on demand. The representation with generalized context dependent HMM state labels (leafs of the phonetic decision tree used to tie the HMM state parameters), yields a completely static structure. In addition, it allows to merge common prefixes of partially tied HMMs. A leaf node in the tree represents the end of one or more words (more specifically word pronunciations). A leaf node represents more than one word in the case of homophones, or more generally, if all associated words share the same sequence of tied HMM states. A small example of a lexical prefix tree is shown in Fig. 1.

In our decoder, the tied HMM state construction is used, resulting in the so called *state tree*. In addition, the HMM state labels and word-ends are associated with states instead of arcs, which is equivalent as long as the graph has a tree structure. We come back to this subtle detail later.

The tree structure imposes a special structure on the search space and thereby on the full search network. The identity of a partial hypothesis' most recent word becomes evident only at leaf nodes in the tree. Therefore, the integration of the LM is

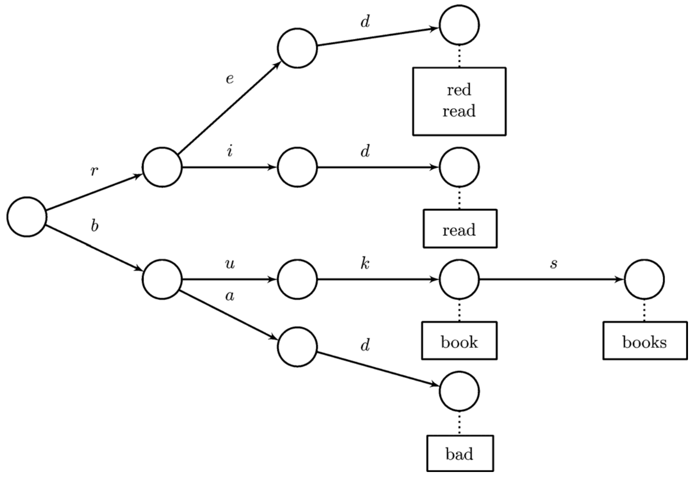

Fig. 1. Lexical prefix tree with context independent phoneme labels.

performed at word-end states only. In order to take account of the dependency of the LM probability on the word history, the hypotheses have to be separated by their respective $(n-1)$ predecessor words (for an LM of order n). For a general weighted grammar, although not considered in this work, the dependency of probabilities on a grammar state is handled analogously.

The partitioning of search hypotheses by word histories leads to the concept of history conditioned "tree copies." A physical copy of the tree is obviously not necessary, but rather an identification of hypotheses by a tuple (s, h) with s a state in the lexical prefix tree and h the word history.

Hypotheses are recombined at the state-level within the tree. At word end states the recombination is carried out on the word-level, merging hypotheses with equivalent histories. A detailed and formal description can be found in Section III-A.

Pruning is performed both on the state-level and the word-end-level. The acoustic pruning can be refined by incorporating the LM probabilities as early as possible using *language model look-ahead* [27]. We describe pruning and look-ahead methods in Sections III-D and III-E respectively.

For the use of across-word context dependent models, the tree structure has to be modified [28]. We only consider triphone models in this work. The triphone model of the last phoneme of a word depends on the first phoneme of the successor word. Just as the model of the first phoneme of a word depends on the last phoneme of the previous word.

For a word with its pronunciation ending with phonemes ab, the word end state in the tree and arcs corresponding to the last phoneme are split into a so-called fan-out for all possible right context phonemes c and resulting triphone models ${}_ab_c$. The root state of the tree is divided into root states for all phoneme pairs b, c with outgoing arcs for triphone models ${}_bc_d$. An example is shown in Fig. 2. Note that, because of the recombination after the fan-in (cf. [28]), the state "tree" may loose the tree property at this part of the network. The size of the fan-out depends linearly on the size of the vocabulary and can make up a major part of the total tree size.

B. Dynamic Transducer Composition (DTC) Search

The WFST formalism allows for a unified representation of the knowledge sources as well as operations for their combina-

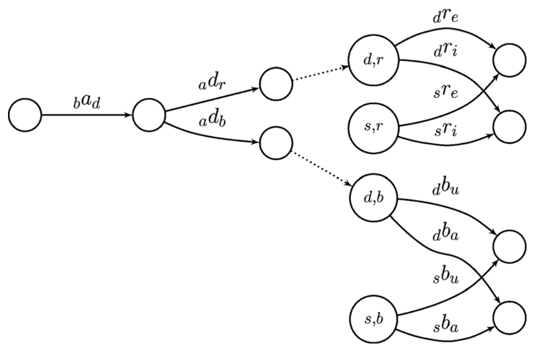

Fig. 2. Example of across-word context dependent modeling in the lexical prefix tree. Left: fan-out structure for the word "bad." Right: split root states. Cf. Fig. 1.

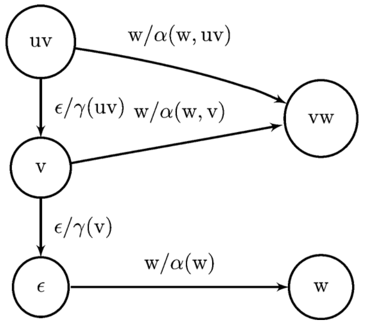

Fig. 3. Part of a 3-gram LM transducer. The state labels denote histories h.

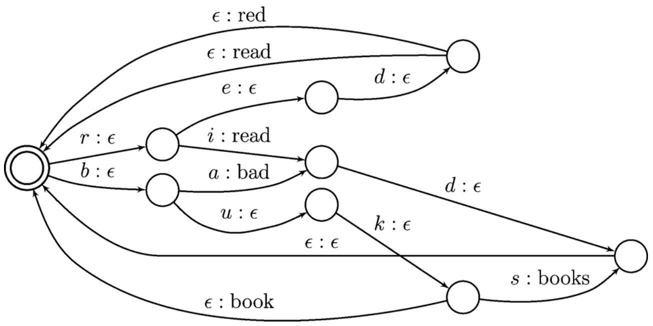

Fig. 4. Lexicon transducer: $\min(\det(L))$.

tion and optimization [11]. A *weighted finite-state transducer* $T = (\mathcal{A}, \mathcal{B}, Q, I, F, E, \rho)$ over a semiring $(\mathbb{K}, \oplus, \otimes, \bar{0}, \bar{1})$ is specified by finite alphabets $\mathcal{A}$, $\mathcal{B}$, a finite set of states Q, a set of initial states $I \subseteq Q$, a set of final states $F \subseteq Q$, a finite set of transitions (or *arcs*) $E \subseteq Q \times (\mathcal{A} \cup \{\epsilon\}) \times (\mathcal{B} \cup \{\epsilon\}) \times \mathbb{K} \times Q$ and final weights $\rho : Q \to \mathbb{K}$. ϵ is the empty word. Weighted automata (or acceptors) are defined similarly by omitting the output labels. $E[q] \subseteq E$ denotes the set of transitions leaving state $q \in Q$. Given a transition $e \in E$, $p[e]$ denotes its origin or previous state, $n[e]$ its destination or next state, $i[e]$ its input label, $o[e]$ its output label, and $w[e]$ is weight. A path $\pi = e_1^k$ is a sequence of consecutive transitions with $n[e_i] = p[e_{i+1}]$ $i = 1 \dots k - 1$. The notation for transitions is extended to $p[\pi] = p[e_1]$, $n[\pi] = n[e_k]$ and $w[\pi] = \bigotimes_{i=1}^k w[e_i]$. The string of input and output labels is denoted as $i[\pi]$ and $o[\pi]$ respectively. For a detailed description of the WFST framework, we refer to [11].

The LM is represented by a transducer denoted as G ("grammar"). The commonly used construction of G for an n-gram LM has a state q_h for every word history h with length $|h| < n$ which has been seen in the LM training data. Regular transitions in the transducer encoding an n-gram hw with $N(h, w) > 0$ (cf. (2)) have the form $(q_h, w, \alpha(w, h), q_{hw})$ or $(q_h, w, \alpha(w, h), q_{\bar{h}w})$ if $|hw| = n$. The backing-off structure is implemented using ϵ-transitions $(q_h, \epsilon, \gamma(h), q_{\bar{h}})$ [6], as illustrated in Fig. 3. This construction is an approximation since the transducer may contain multiple paths for a string whose weights do not correspond to the correct probability according to the model. In most cases a path through backoff-transitions has lower probability than the correct path though, which is why this approximation yields good results. An exact representation of the LM can be achieved by using failure transitions or a less compact transducer [29]. A failure transition is taken only if no other outgoing transition of the same state matches the input symbol.

The lexicon transducer L is derived from the pronunciation dictionary. It has phonemes as input and words as output labels, that is it accepts the Kleene closure of the set of pronunciations and maps them to the individual words. The transduction may be weighted with pronunciation probabilities. An example of an L transducer is depicted in Fig. 4.

The transducer C is used to map sequences of context independent phonemes to sequences of context dependent units. A straight-forward construction of C maps to triphones or context dependent phones of higher order. A more compact representation is achieved by exploiting the classes of equivalent context as defined by the decision tree used to cluster the HMM state models [30]. In this case the mapping yields tied HMM models.

The structure of the HMMs and their transition probabilities can be compiled into a transducer H, which maps from context dependent units to sequences of HMM states. In our default decoder configuration, H is not explicitly constructed, but the HMM states and transitions are expanded dynamically during search by the decoder.

The individual transducers are combined by the finite-state operation of transducer composition as $H \circ C \circ L \circ G$ yielding a weighted mapping from sequences of HMM states to sequences of words. The composed transducer can be optimized in terms of size and distribution of weights using weighted transducer determinization, minimization, and weight pushing [9].

The composition of two transducers can be performed using so-called lazy evaluation, by computing the arcs of a state only when it is discovered [5]. In a dynamic network decoder, the composition of $(C \circ L)$ with G is computed on-the-fly [2]. Already computed arcs can be cached for increased runtime efficiency, but higher memory consumption.

For simplicity, we consider the composed transducer $(C \circ L)$ and the dynamically expanded HMM states as single transducer A, which is the acoustic model part of the search network. The states in the dynamically composed transducer $(A \circ G)$, which is the search network for this search strategy, are basically tuples (s_A, s_G) of acoustic model states s_A and LM states s_G. These tuples are augmented with additional information by the composition filters described below. At this point, the similarity to the HCLT search strategy becomes apparent. The remaining differences between the search networks are analyzed in Section III-A.

The transducers L or $C \circ L$ or both are usually determinized to merge common (triphone model) prefixes and minimized to share common futures. The L transducer may be ambiguous and thus non-determinizable, which can be resolved by adding transitions with disambiguation symbols [11]. Because of the cyclic structure of the transducer, also the transitions between words and therefore the structures for the across-word context dependency of the models is optimized. These optimizations remove as much redundancy from the network as possible. The output labels (words) are shifted by transducer determinization and minimization, such that their positions do not necessarily correspond to word-ends.

The minimization of L has only limited effect on the transducer resulting from the composition with G though. Paths merged in L having different output labels (appearing on arcs before the consolidation) result in separate paths in the composed transducer, except for paths through the unigram state of G. Merged suffix paths in the resulting search network, which do not pass the unigram state in G, are therefore pronunciation variants of the same word with common suffixes and shared models for different across-word contexts.

Several issues have to be considered for the composition $A \circ G$ to be efficient. Most states in G have outgoing arcs labeled with words of only a relatively small subset of the vocabulary, as a result of the sparseness of the LM. The unigram state, that is the state representing the empty history, is a notable exception. Thus, many paths from a certain state in the composed transducer with ϵ output labels have to be traversed until an output label is reached, which decides whether the arc can be matched with an arc in G. This delayed matching generates many dead-end paths, enlarging the search network and hence the search space.

In the static network construction, the word labels in L are placed at the first arc and L is not determinized before building the composition with G, which is then followed by applying determinization of $(L \circ G)$. The generation of unsuccessful paths in $\det(L) \circ G$ can be prevented by pre-computing the set of reachable output labels for each state in L. The label-reachability information can be used to prevent matching of output epsilon arcs in $\det(L)$ resulting in dead-end states [14]. This label look-ahead is used in generalized label-reachability composition filters [2], [31]. The composition filter concept can also be used to implement an on-the-fly redistribution of weights (weight pushing) and labels (label pushing). Composition with a label-reachability composition filter results, using a graphic description, in A transducers reduced to the pronunciations of those words available for a particular LM context. The on-the-fly weight pushing enables integrating the weights of G as soon as possible (cf. Section III-E).

The label-reachability filter with label pushing allows to merge common suffixes across different LM histories (also called tail-sharing), which results in a significantly smaller search network (if constructed fully) [31]–[33]. The effect on the actual search space is very limited though, because only a small fraction of the search network is visited.

Another approach to on-the-fly integration of LM scores was proposed in [15]. The described method generates hypotheses using a static search network with unigram LM scores. These hypotheses are re-scored on-the-fly using a higher order LM. A comparison between the on-the-fly rescoring approach and look-ahead composition along with a comprehensive survey of other online composition techniques is given in [34]. In this work, we use look-ahead composition.

III. Comparison of Search Strategies

A. Search Network

We use the notation from (4) to describe the search networks for both search strategies in a unified way. In the following we assume an LM of order n for a vocabulary W and a set of (tied) HMM state labels Σ.

The lexical prefix tree of the HCLT search is specified by a set of states S_L, a labeled edge relation $E_L \subseteq S_L \times \Sigma \times S_L$, and word-end labels $o[s] \subseteq W$.

The HCLT search network can then be described as follows. The set of states $S \subset S_L \times W^*, \forall\ (s,h) \in S : |h| < n$ consists of tuples of prefix tree states and word histories. The transitions $E = E_I \cup E_W$ are divided into within-word arcs $E_I := \{((s'_L,h),\sigma,(s_L,h)) : (s'_L,\sigma,s_L) \in E_L\}$ and across-word arcs $E_W := \{((s'_L,h),\sigma,(s_0(s_L),\bar{h}w)) : (s'_L,\sigma,s_L) \in E_L, w \in o[s_L]\}$ where $s_0(s)$ is the fan-in root state for a word-end fan-out state s. The score function integrates emission and transition probabilities for within-word transitions $e = ((s'_L,h),\sigma,(s_L,h))$: $\phi(x_t,e) = p(x_t|\sigma) \cdot p(s_L|s'_L)$. Across-word transitions $((s'_L,h),\sigma,(s_L,\bar{h}w))$ incorporate in addition the LM probability $p(w|h)$. In the decoder, the transitions E_W are computed on-the-fly, while E_I is defined by the static tree.

The states of the search network for the DTC search strategy are defined by the state tuples (s_A, s_G) of the composed transducer $A \circ G$. For simplicity, we assume that A has no input ϵ labels. The transitions depend on the output label $o[e_A]$ of transition e_A in transducer A and can be separated into two sets $E = E_\epsilon \cup E_W$.

For transitions $e_A \in E_A$ in A with output label $o[e] = \epsilon$, the search network has transitions $E_\epsilon := \{((s'_A,s_G),\sigma,(s_A,s_G)) : (s'_A,\sigma,\epsilon,q,s_A) \in E_A\}$ with corresponding score $\phi(x_t,e) = p(x_t|\sigma) \cdot q$.

Transitions with word output label change the G-part of the state tuple. For a transition $e_A = (s'_A,\sigma,o[e_A],w[e_A],s_A)$ in A and a state $s' = (s'_A,s'_G)$, the search network has transitions $(s',\sigma,(s_L,s_G)) \in E_W$ for every path π in G from $p[\pi] = s'_G$ to $n[\pi] = s_G$, with $o[\pi] = o[e_A]$. The corresponding arc score is $\phi(x_t,e) = p(x_t|\sigma) \cdot w[e_A] \cdot w[\pi]$. Let h' be the word history associated with state s'_G, $v = o[e_A]$ a word label, and $h = \bar{h}'v$, then the set of paths to be considered includes not only the transition from h' to h with weight $p(v|h)$ but also to all truncated history states. The weights of paths to truncated history states include the back-off weights.

Using the label-reachability composition filter, only a subset of the transitions E_ϵ is generated. Given a label reachability $r[s] \subseteq W$ defining the set of reachable output labels for every state s in A, only transitions $((s'_A,s_G),\sigma,(s_A,s_G))$ with $r[s_A] \cap O[s_G] \neq \emptyset$ and $O[s_G] = \{o[e] : e \in E[s_G]\}$, are required (cf. [31]).

In addition, the label-reachability composition filter with label pushing updates the G-part of the state tuple as soon as an output label can be uniquely determined, i.e., as soon as $|r[s_A] \cap O[s_G]| = 1$. This earlier combination of paths results in a smaller search network. The application of label pushing is not reflected in the definition of E_ϵ and E_W though, which are rather intended to illustrate the general search network construction.

Backpointers for the HCLT search are defined on the word-level, pointing to the best word start hypothesis. Hence, backpointers are propagated within words and updated at word-boundary transitions. The DTC decoder generates backpointers at the arc-level giving the best start hypothesis of a triphone.

B. Static Network Part

The static acoustic model part of the search networks, namely the state tree and the A transducer, differ in structure and size even though they represent the same models and the same mapping from HMM states to words.

The state tree is nearly the same as a deterministic $H \circ C \circ L$ transducer. In contrast to the WFST construction, the state tree is acyclic, the transitions from word-end states to one of the root states are computed at runtime by the decoder. In the straight-forward construction, the root states of the tree do not exploit the HMM state tying. Separate root states are generated for every pair of phonemes to account for the across-word context dependency. This structure introduces non-determinism at the word-end states.

The construction of the A transducer starts with a deterministic minimal cyclic L transducer. The context dependency of the models—both within and across words—is introduced by applying transducer composition with the C transducer. Across-word structures as in the HCLT state tree are not explicitly built but rather generated implicitly and in an optimized way.

However, the across-word structure described here is not an inherent property of the HCLT search strategy but an implementation specific design decision. In [35], the state tree size was significantly reduced by introducing merged fan-out structures. This construction moves the word labels before the last phoneme of a pronunciation and therefore requires modifications of the search strategy. A minimized network is also used in [21].

We limit the discussion to triphones in this work. However, models accounting for larger phonetic context are relatively easy to integrate in the DTC search by using a different C transducer. For the HCLT search, incorporating larger contexts, especially across words or across multiple words, requires significantly more effort.

C. Hypotheses Recombination/Dynamic Programming

Recombination of hypotheses occurs due to the maximization in (4) at the state-level for hypotheses entering the same search network state s at time t independent of the predecessor state. The two search strategies recombine hypotheses at the HMM state-level for network states (s, h) with the same history h or G-state respectively.

In the HCLT search strategy, recombination at the word-level is performed at word-ends only. Hypotheses are merged by the across-word transitions if they have an equivalent history (according to the LM order) and the same fan-in root state in the prefix tree. The sparsity of the LM is not exploited.

The DTC search strategy recombines hypotheses at the word-level as soon as a word label can be uniquely determined from the set of reachable output labels in A and the input labels of corresponding arcs in G. The target states in the search network of such transitions with non-empty output label have an updated G state according to the new word history. As a result of the LM sparsity, the new G state may correspond to a shortened history. In addition, because of the epsilon back-off transitions in G, the hypothesis is also expanded to all history states with lower order history down to the unigram state. For example, a hypothesis for network state (s'_A, uv) and a transition in A to state s_A with output label w is expanded to network states (s, vw), (s, w) and (s, ϵ) if the LM contains respective histories. Therefore, at network states (s, ϵ) hypotheses are recombined independently of the predecessor word. As output labels are not necessarily associated with word-ends, the word-level recombination may occur as soon as the word identity can be determined.

D. Pruning

The potential search space of the HCLT strategy has a size of $O(|W|^{n-1} \cdot |S_L|)$. Whereas the size of the potential search space for the DTC strategy is $O(|G| \cdot |A|)$. The number of states in G is proportional to the number of n-grams in the LM, more specifically to the number of distinct histories. However, only a small fraction of the search space needs to be generated during decoding. The size of the actual search space depends on the degree of redundancy in the search network and—more importantly—on an effective pruning.

The beam search keeps at each time-frame only state hypotheses with scores within a certain range relative to the currently best hypotheses. Hypotheses with

$$Q(t, s) < f \cdot \max_{s'} Q(t, s')$$

are pruned, where f is a pruning threshold controlling the beam width. In addition, the absolute number of active state hypotheses can be limited by keeping only the m best hypotheses, usually implemented using histogram pruning [36].

The HCLT search strategy applies another pruning step in addition. The word-end hypotheses are pruned, after applying the LM score, relative to the currently best word-end hypothesis. In addition, the absolute number of word-end hypotheses can be limited as well. This word-end pruning (also called LM pruning [18]) limits the number of active word histories (tree copies) and thus reduces LM look-ahead computations (described in Section IV). Usually a pruning threshold smaller than the general threshold can be used at this point. A detailed analysis and improvements of pruning strategies for the HCLT search can be found in [37].

DTC (as well as WFST-based static network) decoders commonly use a single pruning threshold without special treatment of word-end hypotheses. Due to the exploitation of LM sparsity and lower additional costs for look-ahead computations, a word-end pruning is assumed to be less important for this strategy. Furthermore, if word-ends are not evident in the DTC search network, it is difficult to separate the corresponding hypotheses. Advanced pruning methods, as for example described in [38], have not been considered in this work.

E. Look-Ahead Methods

The early incorporation of the LM in the search process is crucial for distinguishing promising hypotheses from unlikely ones and thus allowing for effective pruning.

The HCLT decoder calculates for each state hypothesis an approximated LM score which is, combined with the hypothesis score, used for the pruning process. This so-called LM look-ahead computes for each state in the prefix tree the set of reachable word-ends [27]. For a state hypothesis of network state (s, h) the best LM score of the reachable word-ends given history h is used. Calculating these scores for every single hypothesis is neither feasible nor reasonable. Instead, the score for each state in the prefix tree with a given history can be pre-computed and stored in a table. Computing all look-ahead tables in advance would in many cases exceed memory constraints. Hence, a table has to be calculated each time a state hypothesis (s, h) with a previously inactive history h is generated. Tree compression (removing linear paths), tree pruning (limiting the tree depth), and caching of score tables can reduce the computational costs. However, with high-order language models the runtime cost for look-ahead score computations outweighs benefits of improved pruning. The computation depends on the number of nodes in the lookahead tree obtained from the prefix tree, whose size depends on the vocabulary size. Therefore, a simplified LM can be used to compute the look-ahead scores, which allows to share look-ahead tables among different tree copies, but results in less effective pruning.

Recently, more efficient strategies for computing the LM look-ahead scores have been described, allowing to incorporate the full LM history [21], [39], [40]. By exploiting the back-off structure of the LM, a sparse look-ahead table for history h is computed, which contains only scores for those words w with $N(w, h) > 0$ [40]. Thereby, the efficiency of the LM look-ahead computation is significantly increased in terms of both time and memory.

In a fully static WFST search network, the reachable LM probabilities are known in advance and can be propagated to predecessor arcs using weight pushing [10]. When the composition is performed on-the-fly this kind of up-front optimization is not possible, because the LM context is not known in advance. The usage of a label-reachability composition filter with weight pushing [2] integrates the weights of the G transducer as early as possible and is comparable to the LM look-ahead technique.

We omit the formal definition of the weight-pushing composition filter here and refer to the description in [2]. The idea is to integrate the weights from transitions in G already on output ϵ arcs in A before a matching output label is reached. The information about reachable output labels, as used by the label-reachability composition filter, is used to compute this look-ahead. From the set of prospective matching transitions in G either the best LM probability (using the tropical semiring) or the sum of probabilities (log semiring) is used. The already pushed weight (simply put) is used as a potential of the target state and weights on successor transitions integrate only differences to this potential. In contrast to the HCLT LM look-ahead, the look-ahead scores do not need to be handled separately, but are used as regular transition weights. The sum of probabilities can be computed efficiently using pre-computed running sums of transition weights.

Even with the sparse lookahead tables, the computation of LM look-ahead scores needs to distinguish reachable word-ends with a corresponding event included in the LM from those requiring a back-off score, due to the back-off-independent search space structure of the HCLT search strategy. In contrast, the search space structure of the DTC search allows to access the corresponding LM scores directly (as arcs in G) within the weight pushing computation.

The LM look-ahead scores can be used to perform an additional pruning of hypotheses already before the calculation of acoustic model scores. This additional anticipated pruning does not impact the actual search space but it reduces acoustic likelihood computations (the usage of LM look-ahead scores for the usual pruning certainly affects the search space greatly).

F. Word Lattice Construction

The lattice generated by the decoder is a compact representation of the actual search space, i.e., the competing hypotheses. It can be generated on various levels of detail, depending on downstream applications. The most frequently used form is a word graph, which is an acyclic weighted finite-state automaton with word labels. Other variants have HMM state or triphone labels. Many applications require also time information (word start/end time), usually associated with the states. Nevertheless, there is no generally accepted single definition of the term lattice [41].

The HCLT search requires only slight modifications to generate compact word lattices. We can assume that the start time of a word-end hypothesis of a word w depends only on w and its $(n-1)$ predecessor words. For a bigram LM used for lattice construction, this assumption is known as "word pair approximation" [42] and was reported to hold in most cases [43], [44]. In general, the optimal word boundary may depend on the whole utterance though.

The HCLT decoder records the competing word-end hypotheses (with different LM context) before the recombination is performed by collecting the backpointers of the recombined hypotheses [44]. Each state in such a lattice corresponds to a hypotheses (s, t), with search network state $s = (s_L, \bar{h}w)$ and s_L a root state of the prefix tree. It has incoming arcs for all word-end hypotheses of word w at time t with some LM context $h = u\bar{h}$. For across-word context dependent models, the states depend on the fan-in state of the prefix tree, i.e., the hypothesized left and right phoneme context [28].

The lattices are pruned implicitly during construction by the word-end pruning (cf. Section III-D). This kind of lattice pruning uses forward scores only. Additional pruning using forward-backward scores, i.e., complete path scores, can be carried out in an optional post-processing step.

Efficient lattice construction using the DTC search strategy (or other search strategies exploiting the LM backoff structure and recombining common lexical suffixes) is not as straight-forward as with the HCLT decoder. The assumption, required by the word pair approximation, that each word-end hypothesis has a unique predecessor word history (or at least a unique predecessor word) does not hold, because of the back-off structure of G, in particular the unigram state. The recombination of paths before the word-end results in a loss of information about competing LM contexts. Several methods have been described in the literature to still generate lattices efficiently.

The method described in [45] first generates lattices on the tied HMM-level ($C \circ L$ input labels) by collecting backpointers of recombined hypotheses at $C \circ L$ states. The resulting triphone lattice transducer has HMM input labels and word output labels, which can be converted to a word lattice by projection on the output labels followed by ϵ-removal. The lattice is pruned using complete path scores. Redundant paths may be removed by subsequent determinization, which adds additional runtime costs. Lattice determinization is not mentioned in [45]. Determinization and ϵ-removel result in a loss of time information.

The approach presented in [41] is similar, but generates an intermediate lattice on the HMM state-level. In a post-processing step, this lattice is converted to a word lattice using determinization with a specialized semiring. The resulting lattice contains, in addition to the words, the corresponding HMM state-level alignment and thereby the correct time information. The size of state-level lattices may become unwieldy and requires careful pruning.

The lattice generation described in [46] keeps a list of word-level backpointers for the n-best distinct word histories for each hypothesis. During state-level recombination, these sorted lists are merged. At word-ends, only the best scoring backpointer is updated and propagated.

Compared to the DTC methods, the HCLT lattice generation requires a low amount of computations in addition to the first-best search and yields word graphs of a well-defined structure.

G. Subtleties in Practice

Some subtle details of the search strategies are often omitted in the literature even though they are relevant in practice.

Word boundaries are obvious in the HCLT search space, which facilitates the assignment of word start and end times as well as the generation of word lattices. The DTC decoder needs to apply some tricks, like special markers in the search network, if the exact word boundaries are required in the decoder output. Our DTC decoder exploits the word boundary information of the context dependent phone models to detect transitions between words.

Non-speech events, like silence and noise, may cause problems, because they do not perfectly fit in the transducer-based framework, if they are not part of the LM [47]. If noise and silence tokens are required in the decoder output, they have to occur in the G transducer. Furthermore, if weight pushing using the log semiring is used, non-speech labeled transitions in G require a weight [48]. If non-speech events shall preserve the language model context, loop transitions have to be added for every state in G and each non-speech event, increasing the size of G [49]. The HCLT decoder, though, does not require LM scores for all word-end states and handles non-speech events independently of the LM.

WFST-based decoders often use generic toolkits offering common data structures and algorithms, which simplify the development of search network construction tools and the decoder itself. New modeling approaches, for example larger phonetic context, class LMs, or dynamic expansion of the LM, require in many cases only modifications of the search network, leaving the decoder untouched. The decoder can be implemented in a generic way, even independent of using static or dynamic search networks. On the other hand, the generality of the implementation may impede some runtime optimizations and approaches not fitting into the transducer framework would be difficult to integrate.

The HCLT search strategy is usually implemented using specialized operations and specific data structures, which can be fine-tuned and well optimized. The non-generic approach, however, complicates the integration of other models, requiring to modify the decoder and the data structures used.

However, as mentioned in the introduction, the search strategies per se are not tied to either implementation method. The HCLT search strategy can be implemented using finite-state transducers, as the DTC search strategy can use specialized hand-crafted data structures and algorithms.

IV. WFST-Based HCLT Search

One of the goals of this paper is to examine the individual differences between the search strategies, pointed out in the previous section, not only theoretically but also in practice. For a clean experimental setup, we use one decoder for all experiments which allows us to analyze individual characteristics separately and independently of the decoder implementation specifics. Because of its flexibility, we chose the WFST-based decoder for this purpose.

The HCLT search strategy can be simulated by an adequately structured WFST-based search network. The definition of the transducers follows from the definition of the search network in Section III-A. The conversion of the lexical prefix tree, as generated by the HCLT decoder, into a finite-state transducer with HMM state input labels is straightforward. A word-end state is modelled by transitions with ϵ input label and word output labels, connecting the state with one of the root states, as depicted in Fig. 5. A word-end state representing several words implies several word-end ϵ-transitions, e.g., the words "red" and "read" in the example. Due to the non-optimized fan-in structure, some word-end states have transitions for the same word to several fan-in root states.

The HMM state labels are associated with the states in the HCLT prefix tree in our decoder. Therefore, all incoming arcs of a state in the lexical prefix tree transducer have the same input label, which is inconsiderable as each state in a tree has exactly one predecessor. The fan-in structure introduces some states with higher in-degree though, resulting in redundant search hypotheses which do not match the HCLT search space. In order to simulate the HCLT search exactly, we replace those transitions by ϵ-transitions and insert an adjacent transition carrying the redundant input label.

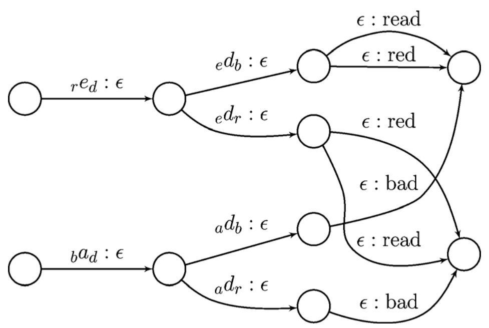

Fig. 5. Example of word-end transitions in the prefix tree transducer (using ϵ-transitions). Cf. Figs. 1 and 2.

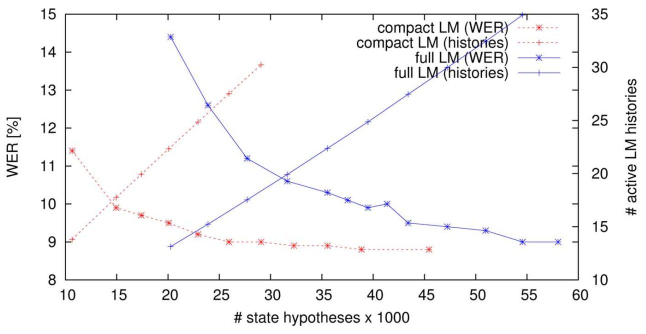

Fig. 6. Active hypotheses vs. WER and active LM histories for full LM and compact backoff LM without LM look-ahead, both using a HCLT prefix tree transducer, without LM look-ahead (WSJ 65 k System).

The incorporation of the LM happens as in the HCLT search at word ends by traversing the ϵ-transition with an output label. The transducer composition then generates as sucessor a search network state for the prefix tree root state and the updated LM history. The backoff-independent word histories of the HCLT search can be achieved by using an LM transducer with failure transitions for the backoff-structure. These backoff-transitions with a dedicated input label are only traversed if no other transition has a matching label.

For technical reasons, we chose another implementation for the LM transducer. Instead of using a static transducer with failure transitions, we generate an ϵ-free transducer for the full LM dynamically as needed during decoding. A state in this transducer has an outgoing transition for every word in the vocabulary. The transition weights are computed on-the-fly (directly from the LM).

The LM look-ahead is simulated by using the label-reachability composition filter with weight pushing. In this case, the LM is not simplified, i.e., the full LM context is used. Weight pushing using the tropical semiring will yield the best reachable LM score, as in the HCLT decoder.

Our recognition systems use state-independent HMM transition probabilities, which are applied by the decoder during runtime. Loop transitions on HMM states are not encoded in the lexical prefix tree transducer, but generated dynamically.

Optional word-end pruning had to be added to the decoder, which is usually not used for the DTC search strategy.

Note that the decoder, especially the computation of LM look-ahead scores, is clearly not optimized for this kind of search network, resulting in poor runtime performance. Consequently, the simulated search can be used solely to analyze the actual search space and not to measure the runtime and memory efficiency.

We can then apply transducer operations to optimize the prefix tree, use a G transducer with backoff-structure, or use the log semiring for weight pushing.

V. Experimental Results

A. Experimental Setup

For the simulated HCLT search, we performed experiments on a moderate size task with our Wall Street Journal (WSJ) read speech transcription system. Due to the computationally expensive HCLT search simulation, we chose a system with lower complexity compared to state-of-the-art ASR systems. We use a vocabulary of 65 k words and a lexicon containing 70 k pronunciations. The acoustic model consists of 450 k densities in 7000 Gaussian mixtures with a single globally pooled variance vector, modeling generalized triphone states of 3-state HMMs with across-word context dependency for an MFCC acoustic front-end. The 3-gram LM used contains 32.1 M n-grams. All experiments are performed on the November ’94 North American Business (NAB) Hub-1 development corpus (49 minutes, 7577 words).

For performance measurements on optimized systems we use a more challenging task, namely transcription of broadcast news and general broadcast conversions using our Quaero ASR system for English [50]. The vocabulary comprises 150 K words with 180 K pronunciations. The speaker independent acoustic model consists of 1 M densities for 4500 Gaussian mixture models. The 3-state HMMs have, in contrast to the WSJ system, an additional skip transition. The VTLN-warped MFCC features are augmented with neural network-based phone posterior features. The 4-gram LM contains 47.7 M n-grams. The test set used consists of 1482 segments with a total duration of 3.3 h and contains about 40 K words.

Runtime measurements were performed on a 2.3 GHz AMD Opteron 6176 CPU using a random sample of the test set containing 20% of the audio data.

B. Search Space Structure

The first experiments compare the impact of the different search space structures. We compare the WFST-based HCLT search with a decoder where we use a backoff LM transducer instead of the full LM. The search network integrating the backoff LM uses the label-reachability composition filter. Both systems do not integrate LM look-ahead scores. The results shown in Fig. 6 reveal a significantly smaller search space for the backoff LM. The number of active LM histories (distinct G states) is also slightly higher when the full LM is used. By exploiting the sparsity of the LM, the number of search hypotheses per LM history is considerably reduced.

When we integrate the LM look-ahead, in form of on-the-fly weight pushing, the actual search space is reduced by an order of magnitude. The huge difference in search space size between the two structure variants vanishes when LM look-ahead is used for pruning, as can be seen in Fig. 7. Here, the backoff LM

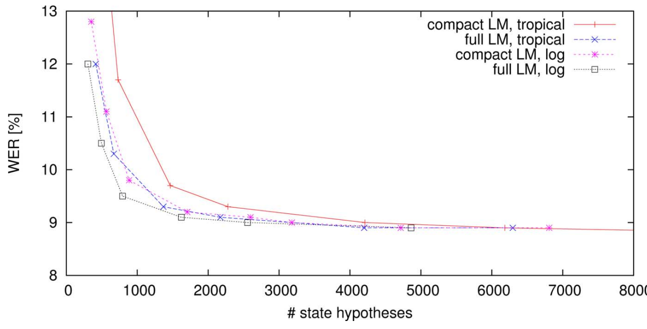

Fig. 7. Active hypotheses vs. WER for full LM and compact backoff LM using a HCLT prefix tree transducer. The LM look-ahead is computed using either the tropical or the log semiring (WSJ 65 k System).

structure generates a slightly larger search space. The differences between different kinds of look-ahead are discussed in Section V-D.

C. Search Network

After converting the HCLT prefix tree to a transducer, we can apply WFST algorithms to optimize its size. We do not use weighted pronunciation variants for the WSJ system. The prefix tree transducer is therefore unweighted and all optimizations affect only size and structure of the transducer.

Because the output labels are shifted away from the word-end by network optimizations, we introduce special input labels for transitions corresponding to the last HMM state of a word, in order to record the word-end time. For each word-end of a prefix tree state, we add such a tagged transition from the predecessor state, carrying also the output label. The ϵ-transitions are not required anymore and hence removed. The actual search space is slightly enlarged by this construction, as a result of the redundant hypotheses for the last HMM state of homophones.

The effect of the search network optimizations is shown in Table I. First, we determinized the prefix tree transducer S_ϵ (having ϵ transitions) for demonstration purposes as an acceptor, i.e., input and output labels of each arc are combined in a single label, denoted as $\hat{S}_\epsilon$. The number of arcs is reduced by about 12%. The position of output labels is fixed in this optimization, thus only the redundant transitions in the fan-in structure are merged. Additional minimization reduces the size only slightly.

The ϵ-free representation, denoted S in Table I, is of course significantly smaller, because we model the junction between words by one instead of two transitions. The insertion of disambiguation symbols ($\tilde{S}$ in Table I), required for transducer determinization, enlarges the transducer temporarily. The subsequent determinization already reduces the number of states slightly, but minimization has the largest effect. During minimization, the output labels can be moved before the fan-out structures and common fan-outs can be shared.

The transducer $\min(\tilde{S})$ has more arcs than the $C \circ L$ transducer expanded to the HMM state level, which corresponds to $H \circ C \circ L$ without HMM loop transitions. In the minimized prefix tree transducer, pronunciations being a prefix of another pronunciation have the output label still on the fan-out transitions. Hence, the fan-out structure can not be shared with other pronunciations. This issue could be resolved by explicitly inserting ϵ-transitions as placeholder just before the fan-out. The deterministic L transducer has the output label as close to the word start as possible and introduces the context dependency according to the state tying afterwards, which prevents the generation of redundant across-word transitions.

TABLE I
SIZE OF THE STATIC NETWORK PART, I.E., THE HCLT PREFIX TREE TRANSDUCER (DENOTED AS S) AND THE $C \circ L$ TRANSDUCER EXPANDED TO THE HMM STATE LEVEL (WSJ 65 K SYSTEM). HMM STATE LOOP TRANSITIONS ARE NOT INCLUDED. S_ϵ AND S ARE THE PREFIX TREE TRANSDUCERS WITH AND WITHOUT INPUT-ϵ TRANSITIONS RESPECTIVELY. $\hat{S}$ IS AN ACCEPTOR OBTAINED FROM S BY COMBINING INPUT AND OUTPUT LABELS OF EACH ARC IN A SINGLE LABEL. $\tilde{S}$ HAS ADDITIONAL DISAMBIGUATION SYMBOLS

transducer	states	transitions
S_ϵ	3.956M	7.076M
$\det(\hat{S}_\epsilon)$	3.941M	6.210M
$\min(\hat{S}_\epsilon)$	3.889M	6.127M
S	1.969M	5.109M
$\tilde{S}$	3.571M	6.711M
$\det(\tilde{S})$	2.493M	4.824M
$\min(\tilde{S})$	0.533M	1.179M
$C \circ L$	0.447M	0.594M

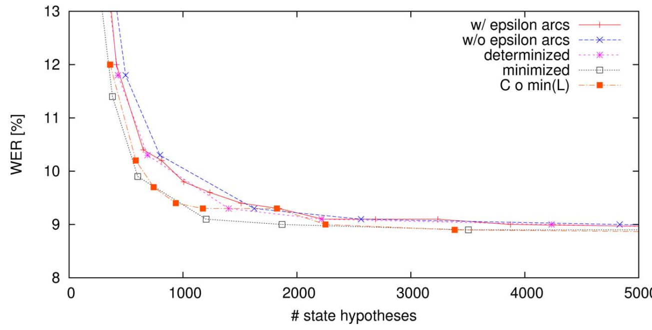

Fig. 8. Active hypotheses vs. WER for various optimizations of the prefix tree transducer. The full LM and LM look-ahead are used (WSJ 65 k System).

The size of the static part of the search network has only small effects on the actual search space, as can be seen in Fig. 8. The smallest search space is achieved with both the minimized prefix tree transducer $\min(\tilde{S})$ and the $C \circ L$ transducer, because of the shared across-word structures. The full sharing of HMM states has no measurable effect on the search space in this experiment. Note that, due to the small test set used for these experiments, small variations in WER relate to only a few word errors.

The effect of minimization of the $C \circ L$ transducer in the DTC decoder on the actual search space is rather small, as we discussed in Section II-B. In an experiment, we study the impact of a minimization of the L transducer on the actual search space. We compared the effect of using either a deterministic L transducer or a minimal L transducer for the Quaero System. No further optimization was applied to the composed $C \circ L$ transducer. The size of the search network is shown in Table II. Even though the number of transitions in L is reduced by 34% and in $C \circ L$ by 55%, the effect on both the actual search space size and

TABLE II
SIZE OF VARIOUS TRANSDUCERS WITH AND WITHOUT MINIMIZATION OF L (QUAERO 150 K SYSTEM). $(C \circ L) \circ G$ IS NOT FULLY CONSTRUCTED FOR DECODING, THE SIZE IS GIVEN FOR DEMONSTRATION ONLY

transducer	states	transitions
$\det(L)$	543,536	902,374
$\min(L)$	130,792	310,213
$C \circ \det(L)$	548,020	1,018,739
$C \circ \min(L)$	166,485	457,815
G	7,348,742	55,044,063
$(C \circ \det(L)) \circ G$	86,926,430	192,643,625
$(C \circ \min(L)) \circ G$	86,080,420	190,869,153

the runtime performance is barely measurable (less than 1% difference for both number of active states and processing time), which is consistent with the size of $(C \circ L) \circ G$ shown for demonstration purposes at the bottom of Table II.

D. LM Look-Ahead

Fig. 7 also compares the impact of using the tropical or the log semiring for weight pushing. Using the log semiring, i.e., using the sum of probabilities of reachable words, results in a smaller search space. These results are consistent with the findings in [10] and [22].

In the case where weight pushing with the tropical semiring is used for a backoff LM-structured search graph, the pruning is less effective compared to the search graph using the full LM. In the backoff LM-structured search graph, the look-ahead scores do not incorporate backoff-weights for hypotheses of lower order history states, as these weights are already included in the hypotheses score. While searching for the highest reachable LM probability, the lower order history might distort the look-ahead. When summed probabilities are used, the effect of imprecise LM probabilities is attenuated.

LM look-ahead using the sum of probabilities requires an efficient implementation in order to reduce not only the size of the search space but also the runtime. The LM scores, negative log probabilities, have to be summed in probability space. This summation requires more expensive computations than determining the lowest LM score.

E. Word Lattice Construction

We compare the lattices generated by our standard HCLT decoder and a DTC decoder. The DTC decoder uses the triphone based lattice construction with subsequent conversion to a word lattice as described in [45].

The generated lattices are not directly comparable. The HCLT decoder performs lattice pruning based only on forward scores. Lattices generated by the DTC decoder are pruned using forward-backward scores. In our implementation, the lattice pruning of the HCLT decoder is coupled with the word-end pruning. Hence, the lattice pruning threshold affects the search space, especially for tight lattice pruning beams. In the DTC decoder, the lattice pruning does not affect the search space.

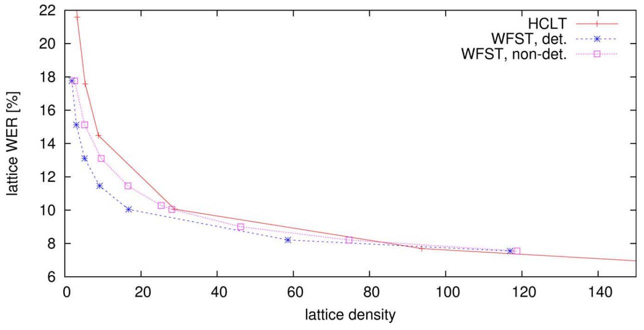

Fig. 9. Lattice density vs. lattice WER for various lattice generation methods (Quaero 150 K System). The first best WER is 20%.

TABLE III
RUNTIME PERFORMANCE OF THE HCLT AND THE DTC DECODER WITH AND WITHOUT LATTICE GENERATION (QUAERO 150 K SYSTEM). THE LATTICES GENERATED BY THE DTC DECODER ARE ON THE WORD LEVEL AND ARE NOT DETERMINIZED

	lattice		RTF	
Decoder	density	WER	first best	lattice
HCLT	28.9	10.0	1.52	1.53
DTC	28.1	10.0	1.17	1.28

The results in Fig. 9 have to be interpreted with these considerations in mind. We generated lattices with the DTC decoder with and without subsequent determinization. We measure the lattice density, i.e., the ratio between number of arcs in the lattice and the number of reference words, and the lattice WER which is the lowest WER of the word sequences contained in the lattice. For small (w.r.t. density) lattices, the lattice WER differs not much among the various lattices. At a reasonable lattice WER of 10%—the first best WER is 20%—the most compact lattice is the deterministic one generated by the DTC decoder. Note that the lattice WER of the HCLT can be higher than the first best results because of the interdependence of lattice pruning and search space size.

We also compared the overhead in computation time for lattice generation over the first best search. This measurement is of course implementation specific and depends on pruning thresholds. Therefore, it can not be generalized, but it gives an impression of the additional computations required. In Table III, we compare the RTF for decoding parameters yielding the same lattice density and equal lattice WER. The lattice generation using the HCLT decoder requires virtually no additional runtime, as the existing book-keeping data is used. The computational overhead for lattice generation in our DTC decoder is about 9% in RTF (processing time divided by audio duration) at the chosen lattice density.

VI. CONCLUSION

The main difference between the HCLT and the DTC search strategy for dynamic search networks is the structure of the search space. The DTC variant exploits the sparsity of the LM using an approximate representation of the LM, whereas the HCLT search always uses full LM histories. Furthermore, the

DTC search allows an early recombination of partial search hypotheses at the word level in contrast to the word-end recombination of the HCLT search. However, the impact of these differences on the actual search space is rather small if effective pruning using LM look-ahead methods is applied.

The more compact representation of the static part of the search network obtained by applying transducer optimization methods slightly reduces the memory consumption of the decoder. The compact representation of across-word phone context-dependency results in a reduced search space. The effect of further minimization of the static search network part on the actual search space proved to be negligible.

We discussed the similarity between LM look-ahead in the HCLT decoder and on-the-fly weight pushing for WFST composition. The usage of look-ahead composition filters allows for the transparent integration of look-ahead scores in the hypotheses scores, without the need for simplified LMs or complex management of look-ahead tables. The search space structure of the DTC search strategy enables a very efficient computation of required LM scores.

The generation of compact word lattices using the HCLT search strategy is straightforward compared to the approaches for the DTC search. The lattices generated by the DTC decoder are comparable in density and quality, but require higher computational overhead.

Given that both strategies operate on search spaces of comparable size, the speed of the decoders depends mainly on an efficient implementation. The actual implementation is however not tied to a WFST-based or specialized representation of the network. The HCLT decoder benefits from low computational costs for state expansions inside the prefix tree, but has higher computational costs for look-ahead score computation and word-end handling. The DTC decoder has somewhat higher costs for within-word state expansions due to the computation of the composition, but advantages in look-ahead score computation and hypotheses combination.

Another important aspect is the scalability of the approaches with respect to the size of both vocabulary and LM. The efficiency of the DTC strategy depends mainly on the size of the LM (namely on the number of arcs per state in the LM transducer). Whereas the efficiency of the HCLT strategy depends mostly on the vocabulary size. The size of non-optimized across-word structures and thereby the impact on the search space increases with the vocabulary size. The computation of LM look-ahead tables depends on the vocabulary size, too. If sparse tables are used, the dependence is rather on the size of LM though. However, the issue of scalability is not covered in this paper and should be addressed in future work.

References

[1] H. Ney, D. Mergel, A. Noll, and A. Paeseler, "A data-driven organization of the dynamic programming beam search for continuous speech recognition," in *Proc. ICASSP*, Dallas, TX, USA, Apr. 1987, pp. 833–836.

[2] C. Allauzen, M. Riley, and J. Schalkwyk, "A generalized composition algorithm for weighted finite-state transducers," in *Proc. INTERSPEECH*, Brighton, U.K., Sep. 2009, pp. 1203–1206.

[3] X. L. Aubert, "An overview of decoding techniques for large vocabulary continuous speech recognition," *Comput. Speech Lang.*, vol. 16, no. 1, pp. 89–114, Jan. 2002.

[4] F. Pereira, M. Riley, and R. Sproat, "Weighted rational transductions and their application to human language processing," in *ARPA Human Lang. Technol.*, Plainsboro, NJ, USA, Mar. 1994, pp. 262–267.

[5] M. Mohri, F. Pereira, and M. Riley, "Weighted automata in text and speech processing," in *Proc. Eur. Conf. Artif. Intell., Workshop Extended Finite State Models of Lang.*, Budapest, Hungary, Aug. 1996.

[6] G. Riccardi, E. Bocchieri, and R. Pieraccini, "Non-deterministic stochastic language models for speech recognition," in *Proc. ICASSP*, Detroit, MI, USA, May 1995, pp. 237–240.

[7] M. Riley, F. Pereira, and M. Mohri, "Transducer composition for context-dependent network expansion," in *Proc. EUROSPEECH*, Rhodes, Greece, Sep. 1997, pp. 1427–1430.

[8] M. Mohri and M. Riley, "Weighted determinization and minimization for large vocabulary speech recognition," in *Proc. EUROSPEECH*, Rhodes, Greece, Sep. 1997, pp. 131–134.

[9] M. Mohri and M. Riley, "Network optimizations for large-vocabulary speech recognition," *Speech Commun.*, vol. 28, no. 1, pp. 1–12, May 1999.

[10] M. Mohri and M. Riley, "A weight pushing algorithm for large vocabulary speech recognition," in *Proc. EUROSPEECH*, Aalborg, Denmark, Sep. 2001, pp. 1603–1606.

[11] M. Mohri, F. Pereira, and M. Riley, "Speech recognition with weighted finite-state transducers," in *Handbook of Speech Processing*, J. Benesty, M. Sondhi, and Y. Huang, Eds. New York, NY, USA: Springer, 2008, ch. 28, pp. 559–582.

[12] H. Dolfing and I. L. Hetherington, "Incremental language models for speech recognition using finite-state transducers," in *Proc. ASRU*, Madonna di Campiglio, Italy, Dec. 2001, pp. 194–197.

[13] D. Willett and S. Katagiri, "Recent advances in efficient decoding combining on-line transducer composition and smoothed language model incorporation," in *Proc. ICASSP*, Orlando, FL, USA, May 2002, pp. 713–716.

[14] D. Caseiro and I. Trancoso, "A specialized on-the-fly algorithm for lexicon and language model composition," *IEEE Trans. Audio, Speech, Lang. Process.*, vol. 14, no. 4, pp. 1281–1291, Jul. 2006.

[15] T. Hori, C. Hori, Y. Minami, and A. Nakamura, "Efficient WFST-based one-pass decoding with on-the-fly hypothesis rescoring in extremely large vocabulary continuous speech recognition," *IEEE Trans. Audio, Speech, Lang. Process.*, vol. 15, no. 4, pp. 1352–1365, 2007.

[16] T. Oonishi, P. R. Dixon, K. Iwano, and S. Furui, "Implementation and evaluation of fast on-the-fly WFST composition algorithms," in *Proc. INTERSPEECH*, Brisbane, Australia, Sep. 2008, pp. 2110–2113.

[17] H. Ney, R. Haeb-Umbach, B. Tran, and M. Oerder, "Improvements in beam search for 10000-word continuous speech recognition," in *Proc. ICASSP*, San Francisco, CA, USA, Mar. 1992, pp. 9–12.

[18] H. Ney and S. Ortmanns, "Progress in dynamic programming search for LVCSR," *Proc. IEEE*, vol. 88, no. 8, pp. 1224–1240, Aug. 2000.

[19] S. Kanthak, H. Ney, M. Riley, and M. Mohri, "A comparison of two LVR search optimization techniques," in *Proc. ICSLP*, Denver, CO, USA, Sep. 2002, pp. 1309–1312.

[20] H. Dolfing, "A comparison of prefix tree and finite-state transducer search space modelings for large-vocabulary speech recognition," in *Proc. ICSLP*, Denver, CO, USA, Sep. 2002, pp. 1305–1308.

[21] H. Soltau and G. Saon, "Dynamic network decoding revisited," in *Proc. ASRU*, Merano, Italy, Dec. 2009, pp. 276–281.

[22] D. Rybach, R. Schüter, and H. Ney, "A comparative analysis of dynamic network decoding," in *Proc. ICASSP*, Prague, Czech Republic, May 2011, pp. 5184–5187.

[23] D. Rybach, C. Gollan, G. Heigold, B. Hoffmeister, J. Lööf, R. Schlüter, and H. Ney, "The RWTH Aachen University open source speech recognition system," in *Proc. INTERSPEECH*, Brighton, U.K., Sep. 2009, pp. 2111–2114.

[24] C. Allauzen, M. Riley, J. Schalkwyk, W. Skut, and M. Mohri, "OpenFst: A general and efficient weighted finite-state transducer library," in *Proc. CIAA*, Prague, Czech Republic, Jul. 2007, pp. 11–23.

[25] S. M. Katz, "Estimation of probabilities from sparse data for the language model component of a speech recognizer," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. 35, no. 3, pp. 400–401, Mar. 1987.

[26] F. Jelinek, "Continuous speech recognition by statistical methods," *Proc. IEEE*, vol. 64, no. 4, pp. 532–556, Apr. 1976.

[27] S. Ortmanns and H. Ney, "Look-ahead techniques for fast beam search," *Comput. Speech Lang.*, vol. 14, no. 1, pp. 15–32, Jan. 2000.

[28] A. Sixtus and H. Ney, "From within-word model search to across-word model search in large vocabulary continuous speech recognition," *Comput. Speech Lang.*, vol. 16, no. 2, pp. 245–271, May 2002.

[29] C. Allauzen, M. Mohri, and B. Roark, "Generalized algorithms for constructing statistical language models," in *Proc. ACL*, Sapporo, Japan, Jul. 2003, pp. 40–47.

[30] R. Sproat and M. Riley, "Compilation of weighted finite-state transducers from decision trees," in *Proc. ACL*, Santa Cruz, CA, USA, Jun. 1996, pp. 215–222.

[31] C. Allauzen, M. Riley, and J. Schalkwyk, "A filter-based algorithm for efficient composition of finite-state transducers," *Int. J. Foundat. Comput. Sci.*, vol. 22, no. 8, pp. 1781–1795, Jun. 2011.

[32] M. Federico, M. Cettolo, F. Brugnara, and G. Antoniol, "Language modelling for efficient beam-search," *Comput. Speech Lang.*, vol. 9, no. 4, pp. 353–379, Oct. 1995.

[33] D. Caseiro and I. Trancoso, "A tail-sharing WFST composition algorithm for large vocabulary speech recognition," in *Proc. ICASSP*, Hong Kong, Hong Kong, Apr. 2003, pp. 356–359.

[34] P. R. Dixon, C. Hori, and H. Kashioka, "A comparison of dynamic WFST decoding approaches," in *Proc. ICASSP*, Kyoto, Japan, Mar. 2012, pp. 4209–4212.

[35] D. Nolden, D. Rybach, R. Schlüter, and H. Ney, "Joining advantages of word-conditioned and token-passing decoding," in *Proc. ICASSP*, Kyoto, Japan, Mar. 2012, pp. 4425–4428.

[36] V. Steinbiss, B.-H. Tran, and H. Ney, "Improvements in beam search," in *Proc. ICSLP*, Yokohama, Japan, Sep. 1994, pp. 2143–2146.

[37] D. Nolden, R. Schlüter, and H. Ney, "Extended search space pruning in LVCSR," in *Proc. ICASSP*, Kyoto, Japan, Mar. 2012, pp. 4429–4432.

[38] T. Hori, S. Watanabe, and A. Nakamura, "Search error risk minimization in Viterbi beam search for speech recognition," in *Proc. ICASSP*, Dallas, TX, USA, Mar. 2010, pp. 4934–4937.

[39] L. Chen, "Efficient language model look-ahead probabilities generation using lower order LM look-ahead information," in *Proc. ICASSP*, Las Vegas, NV, USA, Apr. 2008, pp. 4925–4928.

[40] D. Nolden, H. Ney, and R. Schüter, "Exploiting sparseness of backing-off language models for efficient look-ahead in LVCSR," in *Proc. ICASSP*, Prague, Czech Republic, May 2011, pp. 4684–4687.

[41] D. Povey, M. Hannemann, G. Boulianne, L. Burget, A. Ghoshal, M. Janda, M. Karafiát, S. Kombrink, P. Motlíček, Y. Qian, K. Riedhammer, K. Veselý, and N. T. Vu, "Generating exact lattices in the WFST framework," in *Proc. ICASSP*, Kyoto, Japan, Mar. 2012, pp. 4213–4216.

[42] R. Schwartz and S. Austin, "A comparison of several approximate algorithms for finding multiple (N-best) sentence hypotheses," in *Proc. ICASSP*, Toronto, ON, Canada, Apr. 1991, pp. 701–704.

[43] X. Aubert and H. Ney, "Large vocabulary continuous speech recognition using word graphs," in *Proc. ICASSP*, Detroit, MI, USA, May 1995, pp. 49–52.

[44] S. Ortmanns, H. Ney, and X. Aubert, "A word graph algorithm for large vocabulary continuous speech recognition," *Comput. Speech Lang.*, vol. 11, no. 1, pp. 43–72, Jan. 1997.

[45] A. Ljolje, F. Pereira, and M. Riley, "Efficient general lattice generation and rescoring," in *Proc. EUROSPEECH*, Budapest, Hungary, Sep. 1999, pp. 1251–1254.

[46] G. Saon, D. Povey, and G. Zweig, "Anatomy of an extremely fast LVCSR decoder," in *Proc. EUROSPEECH*, Lisbon, Portugal, Sep. 2005, pp. 549–552.

[47] P. Garner, "Silence models in weighted finite-state transducers," in *Proc. INTERSPEECH*, Brisbane, Australia, Sep. 2008, pp. 1817–1820.

[48] C. Allauzen, M. Mohri, B. Roark, and M. Riley, "A generalized construction of integrated speech recognition transducers," in *Proc. ICASSP*, Montreal, QC, Canada, May 2004, pp. 761–764.

[49] D. Rybach, R. Schlüter, and H. Ney, "Silence is golden: Modeling non-speech events in WFST-based dynamic network decoders," in *Proc. ICASSP*, Kyoto, Japan, Mar. 2012, pp. 4205–4208.

[50] M. Sundermeyer, M. Nußbaum-Thom, S. Wiesler, C. Plahl, A. E.-D. Mousa, S. Hahn, D. Nolden, R. Schlüter, and H. Ney, "The RWTH 2010 Quaero ASR evaluation system for English, French, and German," in *Proc. ICASSP*, Prague, Czech Republic, May 2011, pp. 2212–2215.

David Rybach (S'07) studied computer science at RWTH Aachen University, Germany. In 2005 he joined the Human Language Technology and Pattern Recognition Group and received his Diplom degree in 2006. From 2006 to 2012, he worked at the Computer Science Department of RWTH Aachen University as Research Assistant. He has been a Software Engineer at Google Inc., New York, NY since 2012. His research interests include automatic speech recognition with focus on efficient search methods and weighted finite-state transducers.

Hermann Ney (M'86–SM'07–F'11) is a full professor of computer science at RWTH Aachen University in Aachen, Germany. His research interests lie in the area of statistical methods for pattern recognition and human language technology and their specific applications to speech recognition, machine translation and image object recognition. In particular, he has worked on dynamic programming and discriminative training for speech recognition, on language modelling and on phrase-based approaches to machine translation. His work has resulted in more than 600 conference and journal papers (H-index 70, estimated using Google scholar). He is a fellow of both the IEEE and of the International Speech Communication Association. In 2005, he was the recipient of the Technical Achievement Award of the IEEE Signal Processing Society. For the years 2010-2013, he was awarded a senior DIGITEO chair at LIMSI/CNRS in Paris, France.

Ralf Schlüter (M'11) studied physics at RWTH Aachen University, Germany, and Edinburgh University, UK. He received the Diplom degree with honors in physics from RWTH Aachen and the Dr. rer. nat. degree with honors in computer science from RWTH Aachen in 1995 and 2000, respectively. He is currently a Senior Researcher at the Computer Science Department, RWTH Aachen University. His research interests cover automatic speech recognition, acoustic and stochastic modeling, discriminative training, confidence measures, and signal analysis.