A Tree-Trellis N-best Decoder for Stochastic Context-Free Grammars

Alexander Seward
Centre for Speech Technology, KTH
Stockholm, Sweden

Abstract
In this paper a decoder for continuous speech recognition using stochastic context-free grammars is described. It forms the backbone of the ACE recognizer, which is a modular system for real-time speech recognition. A new rationale for automata is introduced, as well as a new model for pruning the search space.

1. Introduction
The primary goal of this work is to create a decoder for high-accuracy recognition in real-time. Another purpose is to enable language models to be dynamically adapted to semantic context.

The language model we have chosen is based on stochastic context-free grammars defined in Extended Backus-Naur-Form with an additional option of adding transition probabilities to symbols; terminals and non-terminals. A terminal is a string token associated with a Hidden Markov Model (HMM) such as a word or a phone. A non-terminal symbol is defined by its production that consists of a left-hand side declaring the symbol name and a right-hand side with valid derivation alternatives. Each alternative is a symbol sequence where the first symbol denotes a terminal, a non-terminal, a pre-terminal, or a pre-pre-terminal. Transition weights are declared before the head symbol. Transition weights are declared before the head symbol.

A stochastic transition set. A stochastic transition rule set and a stack alphabet of all symbols,

\[ S = \{ a_i \mid a_i = P (t_i, s) \} \]

is the alphabet of all symbols, \( V \) is the class transition probability vector \( \{ a_i \mid a_i = P (t_i, s) \} \).

By applying this formalism, we may represent a stochastic context-free grammar \( G = (V, \Sigma, \Lambda, S) \) as a Stochastic Pushdown Automaton (SPDA). An SPDA (see Figure 3), according to our definition, differs from a conventional pushdown automaton in several aspects. Firstly, and this also holds for Stochastic Finite-State Acceptors; nodes can be of different types and contain other structures, such as HMMs. Secondly, we use pushdown stores to stack node references instead of symbols.

The general purpose of these discrepancies is to produce models that are more adapted to decoding, rather than lexical parsing. An SPDA is defined as \( \Phi = (K, D, \Sigma, I, S, F, \Delta) \) is a set of terminal and non-terminal nodes, \( D \) is a set of distributor nodes, formed by any set of nodes sharing some or all constituents, with the sole purpose of reducing the memory footprint of the automaton. \( \Sigma \) is a lexicon of labeled word-HMMs, \( \Gamma = \Theta^0 \) is a stack alphabet, \( S \subseteq K \) is a set of initial nodes. \( F \subseteq K \) is a set of final nodes and \( \Lambda \) is a stochastic transition set. A grammar \( G \) is defined to be equivalent to a model \( \Phi \) iff 

\[ L(G) = L(\Phi) \]

is the language produced by \( G \), \( \Phi \) accepts all strings in \( L(\Phi) \) and rejects all strings not in \( L(\Phi) \).

SPDAs are very memory-efficient computing devices with powerful modeling capabilities, including Chomsky type II grammars (see e.g. [1]), i.e. any context-free grammar including irregular ones. However, the requirement of a stack memory for each concurrent hypothesis trace over the model reduces the performance of applicable algorithms.

If \( 0 \) is optional and \( succ(0) \) exists (black arrow):

- initiate \( \tau (succ(0), \) copy of \( s) \).

If \( 0 \) is a non-terminal node:

- push reference to \( succ(0) \) onto the stack \( s \).

- for each alternative \( 0 \) of \( 0 \) (white arrows):

  - initiate \( \tau (0, \) copy of \( s) \).

- else if \( 0 \) is a terminal node:

  - initiate processing of HMMs associated with \( 0 \).

  - if \( 0 \) is final: path grammatically acceptable.

  - if \( succ(0) \) exists:

    - continue trace \( \tau (succ(0), \) s) .

    - else if \( s \) is not empty:

      - pop node reference \( 0_{pop} \) from stack \( s \).

      - continue trace \( \tau (0_{pop}, \) s) .

\[ \text{Figure 1. A trace routine } \tau (0, \) s for an SPDA, where } 0 \text{ is a node and } s \text{ is a stack (See Figure 3).} \]

1 In order to simplify selection of subsets, we use the notion of a set of initial nodes rather than a single initial node and a set of extra rules. This notion allows forests of disjoint lexical tree structures.
There are SCFGs, or subsets of such grammars (an initial node and all its descendants), that are regular (Chomsky type III, see e.g. [1]). The regular languages generated by these non-recursive rules are not only accepted by SPDA. Kleene’s theorem states that regular languages are exactly those recognized by finite-state acceptors (FSA). Many recognizers (e.g. HTK [5]) are designed to only accept context-free grammars of regular type. FSAs have the property of reduced decoding complexity without a performance trade-off compared to pushdown automata, which are probable motives for such design decisions. However, the SPDA principle and its implementation have some advantages over a stochastic FSA. They can handle any context-free grammar and enable a very memory-efficient representation of language models. And more importantly, some advantages over a stochastic FSA. They can handle any context-free grammar and enable a very memory-efficient representation of language models. And more importantly, for regular grammar segments, SPDA can be easily transformed to Stochastic Finite-State Acceptors (SFSA). An SFSA (see Figure 3) is defined as $(G, \Sigma, A, S, F, T)$, where $G = (V, \Sigma, \Delta, S)$, where $V = \{A \ldots n\}$, $\Sigma = \{the \ vocabulary \ of \ e, f, g, h, i, j, l, m, n\}$, $A = \{A \rightarrow B\ K\}$, $B \rightarrow 0.6 C D ? \ | \ 0.4 D C ?$, $C \rightarrow 0.5 e f \ | \ 0.4 g ? \ | \ 0.1 h$, $D \rightarrow 0.7 h \ | \ 0.3 i j$, $K \rightarrow 0.8 l \ | \ 0.2 m n \}$.

**Figure 3.** Lexical-acoustic automata generated from a very simple grammar. Given the grammar definition $G$, we first build an SPDA $\Phi(G)$. In the transformation to an SFSA $\Phi(G)$, nodes $e, f, g, h, i, j$ are cloned to 7 new nodes: $e', f', g', h', h'', i', j'$. and the number of arcs is increased from 20 to 32. Although this small example does not show actual size relations, it gives a general idea of the underlying principles. Note that successor arcs from optional nodes are preserved in the expansion. When reaching an optional node, it is processed and alternative arcs (white arrows) are traced, and in parallel, a trace is initiated in the successor arc (black arrow), rendering a skip over the optional node. Paths to final nodes are grammatically acceptable.
3. Decoding

The input to the decoder consists of a lexical-acoustic automaton, parameterized speech data, and model parameters supplied by a grammar node controller. The purpose of this controller is to enable dynamic adaptation of recognizer behavior. By assembling knowledge given by the current dialogue state and semantic context, accuracy can be improved. This is realized by modifications of node transition probabilities, as well as restrictions to an initially very general language model. For example, from a series of earlier utterances, predictions can be made of what the user might say next. The decoder can then promote search in a specific subspace and hereby intensify the generation of hypotheses in the selected domain. A hypothesis is a sequence of words, with an estimated probability. A hypothesis is partial, as opposed to full, if the intra-HMM search associated with the last word has not yet reached a final state. Hypotheses are definable by a specific path in the lexical-acoustic model, where word \( w_n \) is a member of the \( n \)-th terminal node on this path. Although hypotheses ending in the same node in the lexical-acoustic model may be completely disjoint, even regarding the last word, they share the ways of how further expansion can be applied. This decisive fact enables us to group hypotheses sharing the same expansion criteria at a specific time frame. A hypothesis is assigned to an additional group for every word added, i.e. for every terminal node visited during generation. Such a group, called a hypothesis record, consists of a set of entries. Each entry is a triple \( \langle w, p, l \rangle \), where \( w \) is a word label reference, \( p \) is an accumulated probability and \( l \) is a backward-link to a predecessor triple. The result is a trellis of interlinked hypothesis records, from which N-best lists, or word graphs, can be generated through a rapid backtracking process. Due to the properties of HMMs, a hypothesis will be expanded lexically indifferently over several time frames before it is pruned. A possible method to cope with this time-alignment issue would be to bookmark the accumulated probability for each start time in each entry. This approach would enable generation of denser N-best lists, i.e. a shorter average distance between scored hypotheses, but has a trade-off of increased memory and processing time. An alternative is to increase the number of simultaneous hypotheses, through less restrictive pruning. This methodology, according to our experience, enables high accuracy, still generating N-best hypotheses of acceptable density.

3.1 Decoding algorithm

An outline of the decoding process is presented in pseudo code (an SFSA is used in this example). Furthermore, application of a Word-Model Pool (WMP) is mentioned, which is a concept of pruning is described in the next chapter.

Algorithm:

- Initially assign word models of \( \psi(S) \) (see Figure 2) to the WMP, where \( S \) is a set of initial nodes in the lexical-acoustic automaton.
- for each time frame:
  - for each active word model \( w_i \) in WMP:
    - Perform a Viterbi iteration over \( w_i \) and let \( h = \) partial hypothesis record associated with the HMM-state with the highest accumulated exit probability of \( w_i \).
    - Reactivate HMM-states above the pruning level.
    - if \( \theta^i \) is final: mark \( h \) as a full hypothesis.
    - if \( \theta^i = \psi(\theta) \neq \emptyset \):
      - for each word \( w_j \in \theta^i \subseteq \Theta^i \):
        - let \( s = \) initial state of \( w_j \).
        - let accumulative probability of entering \( w_j \):
          \[
          p_{ij} = P(h)P(w_j \in \theta^i | w_i \in \theta^i) \n          \]
        - if a hypothesis record of \( s \) does not exist:
          - Create a new hypothesis record in the hypothesis trellis.
          - Add the entry \( \langle w_1, p_1 \rangle \) to the partial hypothesis record of \( s \) and set a possible new maximum probability.
          - Add \( w_j \) to the to WMP.
- Execute a pruning stage in WMP.

Figure 4. Trellis of hypothesis records, with a top-scored path in bold.

Figure 5. An HMM with associated hypothesis records.

3.2 Generating result paths

The best-scored hypothesis is obtained by the highest ranked entry of the set of trellis records associated with final nodes. By traversing backwards arcs and choosing the word of the top-scored entry of each record, we will construct the best sentence path in reverse. A trace from another end-record will generate other sets of paths with lower probabilities. In each record a lower ranked entry can be selected, generating even more hypotheses. The difference to the best-scored entry of the record is then subtracted from the total accumulated probability of the calculated path. This is a very fast procedure, which in our implementation allows an updated N-best list of moderate length to be displayed, as the user speaks, without affecting performance significantly.
4. Word-Model Pool

When a search space of active word models grows too large, it must be reduced to a limited set for computational reasons. Typically, pruning methodologies apply two forms of threshold criteria, used individually or combined [3, 4]. The relative-likelihood approach, also referred to as beam pruning, is to process partial hypotheses that differ by no more than a relative distance from the best-scored partial hypothesis. Although this method is computationally efficient, it will result in an uneven time-distribution of the number of active hypotheses, making it less feasible for real-time tasks. Another approach is to use a maximum active model threshold that sets an upper limit on the number of active models per frame. If the number of active models would exceed this limit, only the N-best hypotheses are kept while the rest are pruned off. However, this latter approach usually requires the sets of hypotheses to be sorted during the process. Maintaining a sorted set (e.g. heap data structure) between time frames is computationally expensive. Our approach is to introduce the concept of a Word-Model Pool (WMP), which applies a more efficient sort. Basically, this model consists of a large range of pre-allocated HMM search structures. This range is partitioned each time frame, using a fast n\textsuperscript{th}-element partial sort. The result is two sub-ranges, divided by a pivot element. These sub-ranges satisfy the properties that no maximum probability in the right sub-range (see Figure 6) is greater than any maximum probability in the left sub-range. The pivot element is at the same position as it would be if the entire range were completely sorted. After applying this partial sort, models in the right sub-range are pruned. Since every model requires a significant chunk of memory for processing (Viterbi, hypothesis handling etc.), in addition to the HMM structure, it is advantageous to reuse pruned models, instead of executing a deallocation-allocation cycle.

The following pruning procedure is applied (See Figure 6):

1. Set the pivot at the number of word models we are able to process at current time frame with respect to available CPU time and memory.
2. Partition the entire word model range using an n\textsuperscript{th}-element sort algorithm.
3. Set the pruning depth at the maximum probability of the pivot-model.
4. The pivot model and all models on the right side of it are pruned off. New model resources are reclaimed starting from the pivot to the right.

Individual states in active HMMs are deactivated (flooded) whenever current probabilities come below the threshold.

5. Final remarks

The principles described in this paper were implemented for the ACE platform. ACE is a modular, Java-based framework for speech recognizers and auxiliary tools, developed at the Centre for Speech technology at KTH. The implemented decoder was evaluated against the HTK recognizer, developed by Entropic. In order to focus the evaluation on the decoding process we used the same input-set for both applications. The acoustic model set consisted of HMM-based phone models using MFCC-parameters. All models were trained using HTK. The grammar model was a regular context-free grammar, since HTK lacks support for irregular CFGs. In this preliminary evaluation we used a vocabulary of 1000 words and a test set of 195 utterances from 10 different speakers. The results indicate no significant difference in accuracy between the systems; the Word-Error-Rate was 4.5% in both cases. Processing time was however reduced by approx 75% using our approach, compared to HTK. We are continuing the development of this decoder and aim to conduct more extensive tests.

6. References