THE BAVIECA OPEN-SOURCE SPEECH RECOGNITION TOOLKIT

Daniel Bolaños

Boulder Language Technologies (BLT), Boulder, CO, 80301 USA
dani@bltek.com

ABSTRACT

This article describes the design of Bavieca, an open-source speech recognition toolkit intended for speech research and system development. The toolkit supports lattice-based discriminative training, wide phonetic-context, efficient acoustic scoring, large n-gram language models, and the most common feature and model transformations. Bavieca is written entirely in C++ and presents a simple and modular design with an emphasis on scalability and reusability. Bavieca achieves competitive results in standard benchmarks.

The toolkit is distributed under the highly unrestricted Apache 2.0 license, and is freely available on SourceForge.

Index Terms**—** automatic speech recognition

1. INTRODUCTION

Bavieca is an open-source speech recognition toolkit intended for speech research and as a platform for rapid development of speech-enabled solutions by non-speech experts. It supports common acoustic modeling and adaptation techniques based on continuous density hidden Markov models (CD-HMMs), including discriminative training. Bavieca includes two efficient decoders based on dynamic and static expansion of the search space that can operate in batch and live recognition modes. Bavieca is written entirely in C++ and freely distributed under the Apache 2.0 license (https://sourceforge.net/projects/bavieca/).

Compared to existing open-source automatic speech recognition (ASR) toolkits such us HTK [1], CMU-Sphinx [2], RWTH [3] and the more recent Kaldi [4], Bavieca is characterized by a simple and modular design that favors scalability and reusability, a small code base, a focus on real-time performance and a highly unrestricted license.

Bavieca was developed at Boulder Language Technologies (BLT) during the last three years to fulfill the needs of research projects conducted within the company. Currently it is being used in a wide variety of projects including conversational dialog systems and assessment tools that are deployed in formal educational settings and other real-life scenarios.

This work was supported by U.S. Department of Education award number R305B070434, NSF award number 0733323, NIH award number R43 DC009926-01 and NSF award number IIS-1111953.

2. DESIGN

Bavieca is written entirely in C++ and makes extensive use of the Standard Template Library (STL) and the Linear Algebra PACKage (LAPACK) for linear algebra support, which is freely available (www.netlib.org/clapack). The code base is quite small, comprising about one hundred C++ classes and 30,000 lines of code.

The toolkit consists of about 25 command-line tools that serve very specific purposes such as accumulating sufficient statistics or estimating model parameters according to some estimation criterion. These tools present a simple interface in which many parameters concerning algorithmic details are optional and receive default values. Following the traditional design, these tools are intended to be invoked from scripting languages such as Perl or Python in order to build recognizers. This is a similar design to that of the HTK [1] or Kaldi toolkits [4], however the number of tools in Bavieca is much reduced. Tools are organized to make the toolkit scalable to compute clusters and decoding engines are available as executables and libraries.

Most C++ classes within the toolkit hide *unimportant* implementation details and expose relatively simple interfaces that favor reusability across different tools. For example, the same aligner objects are used in all the accumulation tools, and the acoustic and language model interfaces are shared by training, decoding and lattice edition tools. Execution errors and warnings are handled via exceptions, which increase the readability of the code.

Acoustic modeling is based on CD-HMMs and the HMM-topology, which cannot be modified outside the code, is fixed to three states with no transition probabilities.

3. FRONT END AND MODELING TOOLS

3.1. Front-end

Speech parametrization is carried out using Mel-Frequency Cepstral Coefficients (MFCC). A configuration file is used to specify feature extraction parameters such as the window size and width, the tapering function, filterbank characteristics, frequency cut-offs, number of cepstral parameters, energy, etc. Dynamic coefficients can be appended to the feature vectors in the form of n-order derivatives or by concatenating static coefficients from adjacent feature vectors (spliced feature vectors). Cepstral mean normalization (CMN) and cepstral mean variance normalization (CMVN) can be applied at both the utterance or session level. Decoders also support stream-mode for live recognition. Feature decorrelation can be carried out using Linear Discriminant Analysis (LDA) and Heteroscedastic LDA (HLDA).

3.2. Alignments

There are four C++ aligner classes implementing graph-based and linear versions of the Viterbi and Forward-Backward algorithms. These classes can handle phonetic-contexts of any length, and are reused across different components in the system such as the accumulation and adaptation tools.

The graph-based alignment is a data-driven method to find the sequence of word pronunciations and symbols that better matches the utterance while performing the actual alignment. The graph-based aligner performs over a directed acyclic graph (DAG) of HMM-states built from the reference string of words (typically the hand-made transcription) by considering alternative pronunciations and optional symbols like silence or fillers. The alignment is carried out against all the paths in the graph simultaneously, although optional and alternative symbols may not receive any significant occupation. The construction of this graph is similar to the procedure in the Attila toolkit [5]. It comprises the following steps, a) generation of an initial graph of words from the reference including alternative pronunciations found in the pronunciation lexicon (optionally) and optional symbols, b) transforming the word-graph into a phone-graph, c) expanding the phone-graph by propagating left and right phonetic context and attaching within-word position labels to each phone-arc using a queue, d) transforming the phone-graph into a graph of HMM-states using context-clustering decision trees and the phone labels from the previous step. Before performing the actual alignment the graph is minimized following a Forward-Backward edge merging process consisting of merging equivalent states and arcs (HMM-states). This process considerably reduces the graph size and thus the time and memory requirements for the alignment.

The linear aligner is the method of choice when the reference is completely known in advance, i.e., no optional or alternative symbols/words are allowed. This is the case when collecting Gaussian specific statistics for semi-supervised adaptation using the best-path from the decoder or when collecting Gaussian statistics at the phone level for discriminative training.

Alignments can be stored in disk in binary or text format. In binary format an array of pairs [HMM-state, occupation] is kept for each time frame. Additionally, for Viterbi alignments the word alignment information is kept, which might not be recoverable otherwise.

3.3. Accumulation and estimation

Accumulation of sufficient statistics is carried out using the accumulation tools: `mlaccumulator` for maximum likelihood (ML) and maximum a posteriori (MAP) estimation, and `dtaccumulator` for standard and boosted maximum mutual information (MMI) estimation. These tools align the data using the acoustic models and the aligner objects previously described and accumulate statistics on disk in a universal format suitable for physical and logical HMM-states and for numerator and denominator statistics. This strategy, although not as flexible as accumulating statistics at the utterance level through the use of alignment objects, is more compact.

Estimation of model parameters using sufficient statistics is carried out using the estimation tools, `mlestimator` for ML estimation, `mapestimator` for MAP estimation and `dtestimator` for standard and boosted MMI estimation.

3.4. Context modeling

Context clustering is carried out using logical accumulators obtained from single Gaussian HMMs and decision trees that are either state-specific or global. Decision trees are generated following a standard top-down procedure that iteratively splits the data by applying binary questions using a ML criterion. Questions are asked about the correspondence to phonetic groups (defined by hand-made phonetic rules) and the within-word position (initial, internal and final). The splitting process is governed by two parameters: a minimum occupation count for each leaf and a minimum likelihood increase for each split. Finally, a bottom-up merging process is applied to merge those leaves which, when merged, produce a likelihood decrease below the minimum value used to allow a split.

3.5. Mixture splitting and merging

Acoustic model refinement through mixture splitting and merging can be performed after each reestimation iteration using the `gmmeditor` tool, the original set of HMMs and the accumulated statistics. Gaussian splitting is performed iteratively until the desired number of components in the Gaussian mixture model (GMM) is reached. At each iteration the Gaussian to split is selected based on either the largest average covariance (default behavior) or the largest occupation.

After the reestimation iteration(s) that typically follow mixture splitting, the occupation of some Gaussian components may fall below the minimum occupation needed to reliably estimate their parameters (typically 100 feature vectors). Thus, aiming for a robust parameter estimation, components with an occupation falling below a given threshold are merged to the nearest Gaussian in the mixture, which is found using a covariance metric. Each GMM after applying these tools will typically have a variable number of components depending on the data aligned to the HMM-state, which varies across reestimation iterations.

3.6. Speaker adaptation

Vocal Tract Length Normalization (VTLN) [6] is implemented using a piece-wise linear function with one breakpoint to scale the filterbank frequencies. Warp factors are estimated under the ML criterion using conventional acoustic models, while unvoiced phones and symbols are excluded from the estimation.

Maximum Likelihood Linear Regression (MLLR) is implemented for both feature-space (fMLLR) [7] and model-space adaptation [8]. For fMLLR a single transform is used. In model-space, mean and diagonal variance adaptation is performed using a regression tree. The regression tree is generated by clustering Gaussian means using either K-means or expectation–maximization (EM) clustering. The clustering process is governed by two parameters: the maximum number of base-classes and the minimum number of Gaussian components per base-class. Adaptation statistics are accumulated in the regression tree and transforms are estimated conditioned on a minimum number of frames and observed Gaussian distributions per transform.

3.7. Language Modeling

The toolkit supports language models (LMs) in the ARPA n-gram format. Internally a n-gram LM is represented as a Finite State Machine (FSM) in which each state represents a word-history and each transition has a word label (or back-off symbol) and a log-likelihood attached. Given a new word, the current LM-state is updated by performing a binary search on the sorted array of transitions, which includes a transition to the back-off state. This representation makes the use of different n-gram sizes transparent to the decoder.

3.8. Lattice Processing

Lattice operations are performed using the `latticeeditor` tool. Operations include: word error rate (WER) computation (oracle), HMM-state and LM marking, HMM-state alignment, path-insertion, determinization (merging of equivalent states and arcs), posterior probability and confidence measure computation, etc. This tool only operates on lattices in binary format, although lattices can be converted to text format for visualization purposes.

3.9. Discriminative training

The toolkit supports model-based discriminative training under the MMI and boosted MMI estimation criteria with cancellation of statistics and I-smoothing to the previous iteration [9]. Discriminative training is carried out using the tools `dtaccumulator` and `dtestimator` to accumulate numerator and denominator statistics and to estimate model parameters from the accumulators respectively. In addition, the lattice edition tool is used to convert the lattices to a format suitable for the accumulation of statistics, which includes: lattice determinization, HMM-state marking, path insertion, etc.

Both numerator and denominator statistics are accumulated from a single set of lattices resulting from decoding the training data. To enable the accumulation of numerator statistics the orthographic transcription of each utterance (with the corresponding time alignment information) is added to each lattice if not present, and the path marked.

The accumulation of denominator statistics may imply a significant number of redundant likelihood computations across overlapping phones in the lattice. For this reason this process is carried out by an optimized Forward-Backward aligner object that utilizes a hash table to cache and reuse likelihood evaluations. This simple strategy can speed-up the accumulation of statistics by over a factor of 2 depending on the lattice depth.

4. DECODERS

The toolkit comprises two large vocabulary decoders, a dynamic decoder and a WFSA (Weighted Finite State Acceptor) based decoder. The main difference between these systems is how the LM is applied during the search. The dynamic decoder uses a decoding network compiled and optimized from the set of physical HMM-states and the pronunciation lexicon, while the LM is applied dynamically at the token level. On the other hand, the WFSA decoder performs recognition on a decoding network in which all sources of information including the LM are statically compiled and optimized.

These two decoders serve complementary purposes. In the context of large vocabulary dialog systems we need a large vocabulary decoder that can handle large and potentially dynamic LMs with a small memory footprint for live applications in resource-constrained scenarios. In turn, the WFSA decoder exhibits excellent performance for experimentation at the expense of a large memory footprint on some tasks.

The design of both decoders presents a number of similarities: a) Decoding networks are FSMs in which arcs and nodes are stored in two contiguous arrays of memory, arcs keep the offset of the destination node, and nodes keep the offset of the first outgoing arc (a node's outgoing arcs are stored contiguously). This compact layout saves memory and favors locality, which reduces cache misses. FSMs representing LMs are stored in the same fashion. b) HMM self-transitions are not explicit in the decoding network but simulated during the search. c) Data structures frequently accessed during decoding (such as Viterbi tokens on the dynamic decoder, active nodes, word-history items, lattice tokens, etc.) are preallocated during initialization as contiguous arrays of memory to preserve locality and speed up the search. This enables the utilization of offsets instead of pointers, which considerably reduces memory requirements in architectures with 64-bit memory addresses. d) Word-history items are organized as a tree; every time a word-arc is visited a new item is appended to the tree. During decoding a garbage collection mechanism is periodically invoked to reuse word-history items and lattice tokens that become inactive. e) During decoding, network nodes (or tokens) are activated only when their updated path score is within the pruning beam with respect to the score of the best partial path.

The implementation of the lattice generation in both decoders is identical and very similar to that of the Attila toolkit [5]. In addition to pruning thresholds, lattice depth is governed by the maximum number of distinct word sequences kept at each active node (token in the dynamic decoder), which are managed by a specialized hash-table.

4.1. Dynamic decoder

This decoder is based on the token-passing paradigm, it supports triphone and pentaphone phonetic contexts and n-gram LMs of any size. The decoding network consists of a word loop built from the set of physical HMM-states and the pronunciation lexicon, while the LM is applied dynamically at the token level.

The decoding network comprises initial, internal and final parts, which contain HMM-states for the first phone, internal phones and final phone of each word respectively. In order to simplify the network building process for wide phonetic contexts, cross-word context spans only adjacent words and only the initial and final parts are sensitive to it while the internal part remains sensitive to intra-word phonetic context only. Initial and final parts are built by enumerating state sequences corresponding to all possible contexts, and are minimized incrementally for efficient support of wide phonetic contexts. Hash-tables are used to keep auxiliary nodes that facilitate the connections to the internal part, which is built one word at a time. Once the network is built, word arcs are pushed to the initial part of the network in order to allow for early token recombination. Finally the network is globally minimized using a forward-backward merging process in order to combine equivalent arcs and nodes.

LM look-ahead is implemented generating a tree from the internal part of the network up to the word arcs. This tree is minimized by merging linear sequences of arcs and pruned in order to speed-up the computation of the look-ahead scores. Histogram and beam pruning are carried out globally, at word ends, and within each active node.

4.2. WFSA decoder

This decoder performs over a statically optimized WFSA that is composed incrementally following a procedure similar to the one described by Novak et al. [10]. The network composition is carried out by the `wfsabuilder` tool and is limited to cross-word triphones and small to medium size LMs, given that the memory requirements would otherwise exceed the available physical memory. Each transition in the acceptor keeps an integer encoding the symbol type (HMM-state index, word-id or epsilon) and the actual symbol, a floating point value with the weight, and an integer with the offset of the destination state within the array of states (12 bytes total). States in the WFSA can only be reached by transitions with the same input symbol, which facilitates the recombination.

The decoder utilizes a specialized hash table to facilitate the activation of states from emitting and epsilon transitions. At each time frame the active states are processed, path scores are updated with transition weights and acoustic scores, and destination states of emitting transitions are activated. Epsilon transitions are processed in topological order until an emitting transition is found and its destination state activated.

Figure 1 shows the real time factor (RTF) of both decoders on the Wall Street Journal (WSJ) Nov'92 task using a bigram LM. Decoders were run on a laptop computer equipped with a Core 2 Duo P8600 processor (2.4 GHz and 3MB cache) and 3GB of main memory (only one core was used). Acoustic models consisted of 112k Gaussian components and were trained under the MLE criterion. The WFSA decoding network had 7M transitions and 3.1M states for a total of 92 MB. When the SSE instruction set was used for acoustic scoring the RTF of the WFSA decoder improved by 20 to 30%. Results in figure 1 improve those reported by Novak [11] for two WFST-based decoders, Sphinx and HTK on the same task using a faster machine (comparing comparable systems).

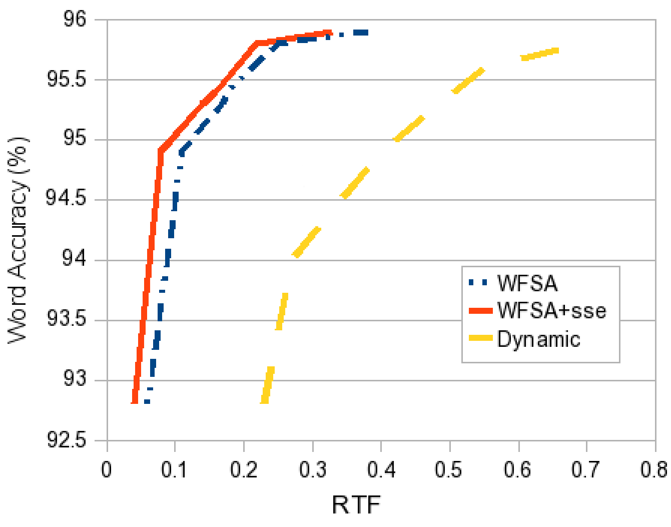

Fig. 1. *RTF vs Accuracy on the WSJ Nov'92 task (bigram)*

4.3. Acoustic scoring

The computation of Gaussian log-likelihoods is usually the most CPU-intensive task within an ASR system. Different techniques have been implemented to speed up this task.

The first optimization consists of using the nearest neighbor approximation [12] in conjunction with partial distance elimination [13] in order to get rid of the exponential and terminating the computation as early as possible. Additionally, the support for Single Instruction Multiple Data (SIMD) parallel computation present in the x86 architecture is utilized to perform floating-point arithmetic operations in up to four vector dimensions simultaneously. In particular, the SSE2 (Streaming SIMD Extensions) Intrinsics were used. In order to take full advantage of this instruction set the data (features and model parameters) were 16-byte aligned, because otherwise a significant portion of the gains resulting from the parallel computation were lost when loading and storing the data into the 128-bit registers. SIMD support can be disabled using a compilation flag if the SSE instruction set is not supported by the architecture. Cholesky decomposition is used to speed up the evaluation of full-covariance Gaussian distributions.

Finally, during decoding multiple network nodes attached to the same physical HMM-state can be active simultaneously, thus acoustic likelihoods for each HMM-state are cached and reused within each time-frame.

5. EVALUATION

The system was evaluated on two tasks: the WSJ Nov'92 task (5k and 20k vocabulary sizes) and the MyST task.

5.1. WSJ

Recognition results are reported on the WSJ Nov'92 test sets. Two evaluation conditions were examined, the 5k closed vocabulary and the 20k open vocabulary conditions. In both cases non-verbalized pronunciations and standard bigram and trigram LMs were used. Acoustic models were trained on 80 hours of data from the SI-284 dataset. For every speech utterance, 39-dimensional feature vectors, consisting of 12 MFCCs and energy plus first and second order derivatives, were extracted and CMN was applied. Word pronunciations were extracted from the CMU dictionary (cmudict.0.7a) using the standard CMU phonetic symbol set without stress markers (39 phonetic classes plus silence). Acoustic models consisted of 4100 clustered HMM-states and about 112k Gaussian distributions with diagonal covariance. No feature transformation or speaker/gender adaptation was carried out.

Table 1 shows the performance of the system in comparison with published results on the WSJ Nov'92 evaluation. WER is given for 5k and 20k vocabulary sizes using bigram (bg) and trigram (tg) LMs, first pass decoding only. The table summarizes WERs from the HTK system described in [14], the LIMSI dictation system [15], and the more recent Kaldi toolkit [4]. WERs from Bavieca are given for ML estimation (which are directly comparable to results from the other systems) and after four iterations of bMMI training in rows six and seven respectively. The last column of the table shows whether gender dependent modeling (GD) was applied. It can be seen that Bavieca produces comparable results to other ASR systems.

	5k		20k		
system	bg	tg	bg	tg	GD
HTK	5.1	3.2	11.1	9.5	yes
LIMSI	4.8	3.1	11.0	9.1	yes
Kaldi			11.8		no
Bavieca	4.7	3.1	10.6	8.7	no
Bavieca+bMMI[1]		2.8		8.2	no

Table 1. *WER on the WSJ Nov'92 evaluation (%).*

5.2. My Science Tutor

My Science Tutor (MyST) [16] is an intelligent tutoring system designed to improve science learning in elementary school students through conversational dialogs with a virtual science tutor. The MyST corpus comprises about 160 hours of conversational speech from 3rd, 4th, and 5th graders. The corpus was collected using close-talk microphones and is divided into 140 hours for training, 10 hours for development and 10 hours for testing.

5.2.1. Recognition setup

The 140 hours of training data were parameterized using 12 MFCC coefficients plus energy and the first and second derivatives. CMVN was applied for each session. The phonetic symbol set comprised 39 phone classes plus 12 symbols to model non-speech events including filled pauses and other events such as silence and breath noise. The vocabulary size was about 7k words including a small number of alternative pronunciations. A LM for each of the four modules (measurement, magnetism and electricity, variables and water) was trained by interpolation from a general LM built on the training transcriptions only. The CMU LM Toolkit [17] was used for training and interpolation purposes. The training process started with a set of single-Gaussian context-independent (CI) HMMs initialized to the global distribution of the data, which were reestimated for five iterations. Then a set of 4600 single-Gaussian context-dependent (CD) clustered triphones was generated using context decision trees. Pentaphone modeling did not reduce the WER and resulted in slower decoding due to the larger decoding network. After three reestimation iterations the CD HMMs were refined through 22 iterations

[1]Results after four iterations of discriminative training, not comparable to results from the other systems, which are based on MLE solely.

of Gaussian splitting and merging, resulting in about 140k Gaussian distributions (diagonal covariance).

VTLN was applied according to the standard iterative procedure until the warp-factors converged (5 iterations). HLDA was applied by performing single-pass retraining on a set of extended features (third derivatives were added) and estimating a full covariance Gaussian distribution for each clustered HMM-state. Once the transform was estimated, features were decorrelated and projected down to the original dimensionality and the HMMs were settled into the new feature space.

Model-based discriminative training was conducted using the bMMI objective function. The training data were decoded using a bigram LM and lattices were generated. The lattice WER (oracle) was 3.06%, and the lattice depth (average number of lattice arcs containing words that cross every time frame) was 383 after making the lattices deterministic. The size of the uncompressed lattices in disk once processed to be used for discriminative training was 323GB. Acoustic scores were scaled down with the inverse of the language-model scaling factor used for decoding and lattices were marked with an unigram LM. The Gaussian specific learning rate was computed setting the constant E to 3. I-smoothing to the previous iteration was carried out using $\tau = 100$. The boosting factor b was set to 0.5. A comprehensive parameter optimization was not carried out.

Table 2 shows the WER on the development set after the different training stages for both speaker independent (SI) and speaker dependent (SD) systems.

	SI	SD
initial CD models	23.92	23.92
+VTLN		23.33
+HLDA	23.43	22.98
+bMMI	20.57	20.32
+fMLLR		19.28
+MLLR		18.25

Table 2. *WER on the MyST task (%).*

6. CONCLUSIONS

Bavieca is an open-source ASR toolkit intended for speech research and system development. The toolkit is based on CD-HMMs and offers a simple and modular design with an emphasis on efficiency, scalability and reusability. Bavieca exhibits competitive results on standard benchmarks and is being successfully used at BLT on a number of research projects addressing both read and conversational children's speech as well as conversational adult's speech. Nonetheless, the development of Bavieca is still a work in progress. Future plans include exploring generic recipes for building ASR systems, further code refactoring and testing, and implementation of new features, such as feature-space discriminative training.

7. REFERENCES

[1] S. Young, G. Evermann, M. Gales, T. Hain, D. Kershaw, X. Liu, G. Moore, J. Odell, D. Ollason, D. Povey, V. Valtchev, and P. Woodland, The HTK Book (for version 3.4). Cambridge University Engineering Department, 2009.

[2] W. Walker, P. Lamere, P. Kwok, B. Raj, R. Singh, E. Gouvea, P. Wolf, and J. Woelfel, "Sphinx-4: A flexible Open Source Framework for Speech Recognition," Sun Microsystems Inc., Technical Report SML1 TR2004-0811, 2004.

[3] 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. of Interspeech, 2009*, pp. 2111–2114.

[4] D. Povey, A. Ghoshal et al., "The Kaldi Speech Recognition Toolkit," in *Proc. of ASRU*, 2011.

[5] H. Soltau, G. Saon, and B. Kingsbury, The IBM Attila Speech Recognition Toolkit, in Proc. *IEEE SLT*, 2010.

[6] L. Lee & R. C. Rose. A frequency warping approach to speaker normalization, 1998. *IEEE TSAP*. 6, 1, 49–60.

[7] M. J. F. Gales, "Maximum-likelihood linear transforms for HMM-based speech recognition," *Computer Speech and Language*, vol. 12, no. 2, pp. 75–98, 1998.

[8] C. J. Leggetter, & P. C. Woodland, 1995. Maximum likelihood linear regression for speaker adaptation of continuous density Hidden Markov Models. *Comput. Speech Langu.* 9, 171–185.

[9] D. Povey, D. Kanevsky, B. Kingsbury, B. Ramabhadran, G. Saon, and K. Visweswariah, "Boosted MMI for model and feature space discriminative training," in *Proc. ICASSP, 2008*, pp. 4057–4060.

[10] M. Novak, "Incremental composition of static decoding graphs," in *Proc. of Interspeech*, 2009, pp. 1175–1178.

[11] J. Novak et al., An Empirical Comparison of the T 3 , Juicer, HDecode and Sphinx3 Decoders, In *Proc Interspeech 2010*.

[12] F. Seide, "Fast likelihood computation for continuous-mixture densities using a tree-based nearest neighbor search," in *Proc. Eurospeech*, 1995, pp. 1079–1082.

[13] C. D. Bei and R. M. Gray, "An improvement of the minimum distortion encoding algorithm for vector quantization," *IEEE Trans. Commun.*, vol. COM-33, pp. 1132–1133, Oct. 1985.

[14] P. C. Woodland, J. J. Odell, V. Valtchev, and S. J. Young, "Large vocabulary continuous speech recognition using HTK," in *Proc. ICASSP*, vol. 2, 1994, pp. II/125–II/128.

[15] J. L. Gauvain et al., "The LIMSI speech dictation system: evaluation on the ARPA Wall Street Journal task," in *Proc. ICASSP 1994*, Vol. I, pp. 557–560.

[16] W. Ward, R. Cole, D. Bolaños et al., My Science Tutor: A conversational multimedia virtual tutor for elementary school science. *ACM Trans. Speech Lang. Process.*, 7(4), 2011.

[17] R. Rosenfeld, 1994. The CMU Statistical Language Modeling Toolkit.