# DynaSpeak: SRI's Scalable Speech Recognizer for Embedded and Mobile Systems

Horacio Franco, Jing Zheng, John Butzberger, Federico Cesari, Michael Frandsen, Jim Arnold, Venkata Ramana Rao Gadde, Andreas Stolcke, Victor Abrash

SRI International, 333 Ravenswood Ave., Menlo Park, CA 94025, USA

Phone: +1 650 859 3284

Email: hef@speech.sri.com

## ABSTRACT

We introduce SRI's new speech recognition engine, DynaSpeak$^{TM}$, which is characterized by its scalability and flexibility, high recognition accuracy, memory and speed efficiency, adaptation capability, efficient grammar optimization, support for natural language parsing functionality, and operation based on integer arithmetic. These features are designed to address the needs of the fast-developing and changing domain of embedded and mobile computing platforms.

## Keywords

Speech recognition, scalability, embedded and mobile computing.

## 1. INTRODUCTION

With the advent of more powerful processors, portable computing devices are gaining widespread use. Some of them, such as the new generation of mobile phones, high-end personal digital assistants (PDAs), and networked systems within automobiles, share the availability of significant computing power as well as information access through wireless networking. Because of the small physical size and the mobile nature of these devices, use of a keyboard may be impractical and sometimes distracting, while speech-driven user interfaces seem the ideal choice. The challenge is to offer a satisfactory user experience, freeing the user from the need to remember predefined commands by allowing the use of naturally constructed sentences, and by providing a robust and accurate recognition capability.

To address this need, we developed a new recognizer, named DynaSpeak$^{TM}$, which would support the use of SRI's state-of-the-art large vocabulary continuous speech recognition (LVCSR) technology on these new platforms with minimal loss of accuracy or robustness. One of our objectives was to have a scalable, flexible recognizer architecture that would not be bounded by any initial design limitations that could make the architecture obsolete when more powerful platforms arrive.

The challenge of supporting our speech technology on the target platforms required tackling several aspects: small memory footprint, addressed in part by a dynamic grammar and dynamic search architecture combined with optimization technology, as well as by the use of compressed acoustic models; integer computation, as most platforms for mobile and embedded systems do not support floating point arithmetic; and low-overhead Gaussian and search pruning, critical for fast decoding in large search spaces. In addition, we implemented full support for our different acoustic and language models (LMs) plus a simple natural language parsing capability. To achieve robustness, we also implemented fast online speaker adaptation and noise compensation capabilities. Finally, a multithread-safe implementation efficiently supports running multiple instances of the recognizer with different grammars and even acoustic models in each instance, a valuable feature for the development of speech-to-speech translation systems. In the following sections we review the most salient characteristics of the DynaSpeak recognizer.

The organization of this paper is as follows: Section 2 describes DynaSpeak's search strategy and grammar representation. Section 3 introduces acoustic and language model support. Section 4 describes an online adaptation algorithm used for speaker and environmental adaptation. Section 5 introduces an efficient algorithm to reduce Gaussian computation and search effort. Section 6 gives a brief description of the grammar optimization algorithm. Section 7 deals with natural language processing support. Section 8 discusses some implementation issues. Section 9 presents a sample mobile application. Section 10 summarizes the paper.

## 2. DYNAMIC SEARCH AND GRAMMARS

To reduce the amount of memory required for a given recognition task, we use hierarchical data structures that include a top-level word grammar and various related word subgrammars that may be loaded on demand. Each word in turn is described by a probabilistic graph of phones that can also be loaded dynamically. Each phone is represented by a probabilistic graph corresponding to a hidden Markov model (HMM), also loaded dynamically. At all layers of this hierarchy, the graph representation uses the same data structure. Unlike typical speech recognition systems, these grammars and subgrammars are not compiled and expanded into a single large flat network structure before runtime. Instead, during recognition, memory is allocated to a subgrammar when a transition to it is made during the search. The subgrammars may then be expanded and evaluated, and the probability of a match between the speech signal and an element of the subgrammar may be computed. During the search, whenever the maximum probability of all active states of a subgrammar, a word, or a phone falls out of the pruning beam width, the related memory is released to ensure small memory consumption. Because unexpanded grammars and subgrammars take up very little memory, this method enables systems with limited memory to recognize and process a larger vocabulary than would otherwise be possible.

The dynamic search strategy and the hierarchical data structure allow for great flexibility in dealing with dynamic grammars, which means that grammars or subgrammars can be added, deleted, replaced, and selected by the application or the user, while the speech recognition system is operating. This feature enables on-the-fly configuration of a recognition application depending on the current context, which is important for many tasks.

## 3. ACOUSTIC AND LANGUAGE MODELS

### 3.1 Acoustic Model

Designed with scalability in mind, the DynaSpeak recognizer supports Gaussian mixture models (GMMs) with generic tying schemes, in particular phonetically tied mixtures (PTM) for small systems, and the more general Genonic models [1] that permit arbitrary tying, with groups of clustered triphone states sharing a common codebook, allowing for robust estimation of the model parameters while maintaining good acoustic resolution. Phone state tied mixtures (PSTMs) are a special case of Genonic model, where triphone state clustering is based on the central phone identity and the HMM state index. When model size must be small because of hardware limitations, PSTM models allow for better acoustic resolution than PTM models with similar-sized Gaussian codebooks. If we choose to use a smaller codebook size for each mixture in a PSTM model, and equate the total number of Gaussians with a PTM model, a PSTM is preferred due to a smaller number of mixture weight parameters, and for being easier to train robustly due to the smaller codebook size. In some of our experiments, with the same training data, PSTM models give equal or better performance than PTM models, with smaller storage requirements.

To satisfy the memory constraints of some applications we have introduced several techniques to make models compact, and yet accurate: First, we developed a decision-tree-based algorithm for mixture weight clustering to reduce the number of distinct probability distributions, thus saving significant storage space. Second, we developed a Gaussian merging algorithm to reduce the number of Gaussian components in different codebooks in an extension of our previous work [2]. The original algorithm was specially designed for PTM models, while the new algorithm allows Gaussian merging in the whole acoustic space, which is more effective in compressing models. Third, we replace mixture weights below a certain threshold with their average value [3], further reducing the number of distinct parameters and enabling compact storage and representation. Finally, we quantize Gaussian and mixture weight parameters to eight bits. This technique must be accompanied by a zero mean/unit variance feature normalization to keep Gaussian parameters within a common range, and with a saturation mechanism in the Gaussian probability computation. Our experiments showed that these model compacting techniques brought very little performance degradation.

Table 1 gives an example of model size compression. The baseline model is a PSTM model trained on 231,000 utterances comprising digit strings, command-and-control words, spellings, and dictations. The acoustic feature vectors are formed by C1-C12 plus normalized C0 mel cepstrum coefficients, with the addition of first- and second-order derivatives, extracted using a 32 ms analysis window and 16 ms frame advance rate on waveforms sampled at 8 kHz. We used a lexicon with 46 phonemes, and three-state left-to-right HMM topology. Thus, the acoustic model has 138 Gaussian codebooks (Genones). We selected a codebook size of 32 Gaussians per Genone. In context-dependent training, 22,194 triphone states were generated, each with 32 mixture weight parameters.

We tested this acoustic model on an unconstrained digit string recognition task with 1000 utterances collected in-house through telephone channels, and obtained a word error rate (WER) of 1.0% and a string error rate (SER) of 7.1%, (second row of Table 1). We then performed the mixture weight clustering and the Gaussian merging procedure, which reduced the parameter size to 10% of the original mixture weights and to 67% of the original Gaussians as shown in the second row. The error rate remained unchanged. A further compression that cut the number of Gaussians by 50% increased the error rate only slightly (last row of Table 1).

**Table 1. Effectiveness of acoustic model compression**

| | **# of mixture weights** | **# of Gaussians** | **WER in %** | **SER in %** |
|---|---|---|---|---|
| Baseline | 22194x32 | 4406 | 1.0 | 7.1 |
| Compressed 1 | 2000x32 | 3000 | 1.0 | 7.1 |
| Compressed 2 | 2000x32 | 2208 | 1.1 | 7.3 |

### 3.2 Language Model

Within the DynaSpeak recognizer, grammars are represented as weighted directed graphs, where nodes are associated with words or subgrammars, and arcs with transition probabilities. This representation is general enough for almost all types of grammars used in automatic speech recognition (ASR). As discussed earlier, a hierarchical grammar structure is allowed. At the user interface, DynaSpeak supports two forms of grammar specification: Java$^{TM}$ speech grammar format (JSGF) and DECIPHER$^{TM}$ probabilistic finite state grammar (PFSG) format.

JSGF specifies rule-based grammars, which are typically used for tasks like command-and-control. JSGF uses a textual representation that can be generated by both developers and computer programs. By an extension of the JSGF specification, we also allow pronunciations of new vocabulary items to be loaded dynamically along with the recognition grammars. PFSG is a special form of finite state string-to-weight transducer, in which words are emitted at nodes instead of arcs. This format is used by SRI's research LVCSR system, the DECIPHER recognizer. PFSGs can be conveniently used to represent statistical LMs, such as N-grams. In addition, DynaSpeak's hierarchical grammar structure allows an efficient representation of class-based statistical LMs [4], where word classes can be represented as subgrammars. The advantage of this representation is that word classes can be dynamically updated at runtime, which provides great flexibility for many applications. LM training and PFSG manipulation are supported by the SRI Language Modeling Toolkit [5].

## 4. ONLINE ADAPTATION

Mismatches between training and working conditions pose challenges to speech recognition technology in real-world applications, especially for embedded and mobile computing. The mismatches could stem from many sources; different speaker characteristics and environmental noise are two major factors. To improve the recognizer's robustness against these mismatches, and therefore promote the usability of speech recognition, acoustic model adaptation techniques have been extensively studied and applied. Based on the operation mode, adaptation approaches can be classified into two categories, online adaptation and offline (batch) adaptation. The former category adjusts model parameters at runtime, incrementally and instantly adapting the acoustic model to match the speaker and the acoustic environment. The latter category modifies the acoustic model based on certain amounts of prerecorded data. Based on the availability of word transcriptions for the speech data, adaptation algorithms can be divided into supervised algorithms, which modify model parameters according to correct word strings, and unsupervised algorithms, which use recognition hypotheses. Clearly, supervised adaptation requires the user's aid and cooperation in the process. In DynaSpeak's application scenarios, the most relevant technique is unsupervised online adaptation, which is also the most difficult; the challenge is to be robust enough to deal with very small amounts of adaptation data and possible recognition errors.

Among the prevailing adaptation techniques, transformation-based approaches have the advantage of needing small amounts of adaptation data and being robust to recognition errors, and are therefore the best candidates for unsupervised online adaptation. As stated in [6], transforms can be applied either to features or to models. We chose to perform feature-based transformation since it does not change the acoustic models, so that we need to save only a single copy of the acoustic models for different adaptation conditions, which is important in keeping a small memory footprint. Although full-matrix or block-diagonal matrix linear transforms proved to be more powerful, we use diagonal transforms because the former require more computation and numerical precision than many current embedded platforms can provide.

We developed an expectation-maximization (EM) algorithm for estimating linear transforms based on the maximum likelihood criterion. This adaptation algorithm was first proposed in [7], where the transforms were applied to the models instead of features. In our implementation, the inverse transform is applied to the acoustic features, which in theory has the same effect. We apply this algorithm in an online unsupervised mode, that is, recognition hypotheses are used to generate the statistics needed by the EM algorithm, and these statistics are accumulated during the use of the system, which produces incrementally refined estimates of the feature transforms. For computational efficiency, Viterbi alignments between HMM states and input speech were used instead of forward-backward alignments, and results proved to be similar. To reduce hardware requirements, we used only integer arithmetic operations in the algorithm implementation. With careful design, this causes little degradation in performance. We also use estimated confidence values to filter out possibly wrong recognition hypotheses, so only the utterances with high confidence values are used to update the transforms. This can improve the robustness of the adaptation algorithm. The transformation-based adaptation brought significant error rate reductions in several tasks.

The following experiment demonstrates the effectiveness of adaptation. We recorded 100 telephone numbers in each of four different conditions using a Compaq iPAQ personal digital assistant: inside a quiet office, off a busy street, at a restaurant, and in a dynamically changing noisy environment. We recognized these 400 utterances using an acoustic model trained as described in Section 3.1 with an unconstrained digit string grammar. Clearly, the training and working conditions are mismatched except for the case of office recordings. Table 2 compares the WER with and without online adaptation.

**Table 2. WER comparison with and without online adaptation for different acoustic environments**

| | Office | Street | Rest. | Noisy | Total |
|---|---|---|---|---|---|
| No adapt | 0.8% | 12.2% | 13.5% | 20.2% | 11.5% |
| Adapt | 0.8% | 8.6% | 9.2% | 13.3% | 8.0% |

As Table 2 shows, online adaptation brings WER reductions from 30% to 35% in all unmatched conditions.

## 5. GAUSSIAN AND PHONETIC PRUNING

Aside from memory constraints, limited processing speed also brings challenges to the speech recognition application on current hardware platforms of many embedded and mobile systems. A response time that is too long could affect the usability of an ASR application as seriously as recognition errors. A practical recognition system must strike a compromise between speed and accuracy; the well-known beam pruning technique in the Viterbi search algorithm is one approach to achieve this kind of tradeoff. To achieve better speed and less accuracy loss, we developed other pruning algorithms in addition to beam pruning, in order to reduce Gaussian and search computation. These algorithms are based on Gaussian shortlists and a form of phonetic pruning, both of which use a vector quantization (VQ) of the acoustic space and add little computational overhead.

Through a VQ algorithm, we partition the whole acoustic space into a number of VQ regions. The VQ codebooks are organized as a tree for quickly locating the VQ region in which a given input feature vector falls. For Gaussian shortlist pruning, we associate each VQ region with a list of Gaussians, which is built in a training procedure in which Gaussians having high likelihood scores for training samples falling within that region are put into the list. During recognition, at each frame, a VQ region is first located according to the feature vector of that frame. Then, only the Gaussians in the corresponding list are evaluated, while others are simply ignored. Thus, a significant part of Gaussian computation is avoided. This technique was first proposed in [1] and has been used in SRI's DECIPHER recognizer.

We recently extended the shortlist technique to prune the search space using a phoneme-based method. We compute the average phoneme posterior probabilities for each VQ region according to the training data samples falling into that region, and store the probabilities in a table. With this table, we can estimate phone posterior probabilities at each frame during search. Then, we prune the search hypotheses corresponding to phone posteriors below a chosen threshold. The method of phone-posterior-probability-based pruning was originally proposed in [8], where it was called *phone deactivation pruning*. In that work, a multilayer perceptron (MLP) was used to estimate the desired posterior probabilities. While this method is convenient within the hybrid connectionist/HMM approach, it is harder to apply to the more common GMM/HMM systems, like DynaSpeak. The proposed VQ-based phone posterior probability estimation method has very little computational overhead, and can be used over a window of consecutive frames for improved estimation robustness. Gaussian shortlist pruning and phonetic pruning can be combined, by sharing the same VQ codebook.

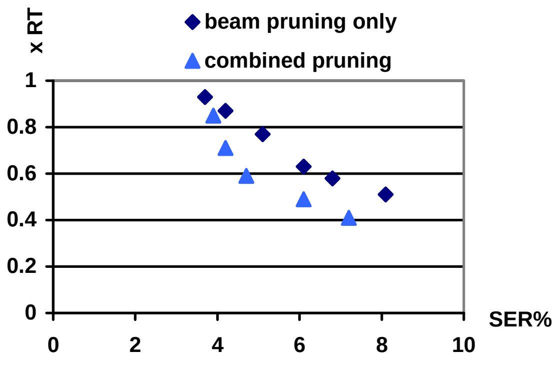

**Figure 1: Comparison between two pruning schemes: beam pruning only and the combination of beam pruning, Gaussian shortlist, and phonetic pruning.**

The following example shows how Gaussian shortlists and phonetic pruning improve the speed-accuracy tradeoff beyond the standard beam pruning on a task that is constrained by limited processing power. Using the same grammar and acoustic model, we tested two different pruning schemes: standard beam pruning only, and a combination of beam pruning, Gaussian shortlists, and phonetic pruning. Figure 1 shows operating points in terms of SER and real time factor (xRT) obtained with different pruning parameters, using the two pruning schemes. (The recognition experiments were performed on a development platform that is more powerful than the target hardware platform; the target performance therefore lies below 1xRT in this case.) It is clear that the combined pruning scheme is superior since it achieves a smaller error rate with faster recognition speed.

## 6. GRAMMAR OPTIMIZATION

Although DynaSpeak's hierarchical grammar structure allows for maximum flexibility and minimum static storage space, it causes some computational overhead to the search algorithm because of the frequent grammar expansion operations. More importantly, it impedes effective inter-grammar optimization, which could merge common partial search paths in different grammars to reduce the search effort. For speed-critical tasks where a small memory footprint is not essential, we developed an algorithm to selectively expand subgrammars and reduce the grammar hierarchy. This algorithm is not designed to fully flatten the hierarchical grammar into a word graph, but to remove certain unnecessary hierarchies for achieving better speed-size tradeoff. In fact, DynaSpeak supports some grammars that cannot be flattened, such as grammars with recursion. In some cases complete flattening could result in serious memory inefficiency; for example, a flat word graph is much less efficient in representing class-based trigram LMs with large word classes being referenced multiple times, than using subgrammars to represent word classes. In other cases, users might want to preserve certain subgrammars for special purposes, like online update, while removing other subgrammars for speed. Our algorithm has control parameters that allow the user to select the desired speed-size tradeoff, and to protect critical subgrammars in the expansion procedure. The design of this algorithm again reflects the main features of DynaSpeak, scalability and flexibility.

User-provided grammars, especially manually created ones, could have numerous redundancies, resulting in duplicated partial search paths. Removing these redundancies not only helps to achieve a compact and memory-efficient grammar representation, but also speeds up recognition. The above-mentioned selective grammar expansion procedure will create more room for possible grammar reduction compared to unexpanded hierarchical grammars. A commonly used approach views grammars as weighted finite state acceptors (WFSAs, a form of transducer) and performs the well-known determinization/minimization procedure [9]. We did not choose this approach for the following reasons: First, not all grammars are determinizable, and the determinization procedure has exponential complexity in the worst case. Second, determinization does not necessarily reduce grammar size; instead, it could increase the size exponentially. Third, the speed advantage of deterministic grammars over nondeterministic grammars is not very obvious in the context of speech recognition. Determinism could make pruning more efficient, but grammar size in terms of number of nodes and arcs also plays a big role. Finally, DynaSpeak internally does not represent grammars in WFSAs; bidirectional format conversion is needed in order to apply WFSA algorithms. In DynaSpeak's context, we need an efficient algorithm that can improve grammar compactness and determinism, but not necessarily to the minimal point.

With these considerations in mind, we designed a grammar reduction algorithm based on the following property: In a grammar, we can merge two nodes having identical word or subgrammar labels and the same set of successors or predecessors with the same transition probabilities without changing the grammar's function. This fact was previously used in [10] to reduce word lattices (acyclic directed word graphs); the new algorithm handles general grammars with special treatment for self-transition arcs.

We implemented this algorithm in a very efficient way based on a specially defined lexicographic order in the node space, and a fast sorting algorithm with linear complexity. In addition, we use a *pushing* algorithm [9] to redistribute probabilities across the graph, moving them as close as possible to the beginning of the sentence. This not only improves the pruning efficiency in the search, but also makes the reduction algorithm more effective. To further improve the reduction algorithm, we used a null node removal algorithm before performing grammar reduction, and a null node reintroduction algorithm after that. Here null nodes refer to special nodes without emission, which can be used to save a number of transitions, and thus reduce grammar size. However, they could also cause search overhead. Here, too, we provide control parameters for null node manipulation, and allow users to select the desired tradeoff between speed and size. A detailed description of the grammar optimization algorithm can be found in [11].

This grammar optimization approach has been used in many applications. For instance, in a phrase translation application described below and in [12], containing a grammar of 700 distinct phrases, the grammar optimization algorithms resulted in a speedup of 100%. We also successfully applied this approach to a class-based trigram grammar, and obtained a 140% speedup. In addition, we applied the graph reduction algorithm to generate a compact representation of multiple word pronunciations. The pronunciations of each individual word are organized as a network, with common phonemes shared. This approach achieves both memory and speed efficiency.

## 7. NATURAL LANGUAGE SUPPORT

In many applications, recognizing word strings from a user's utterance is not the final goal. The application may need to analyze the recognition result, (partly) understand the meaning of the utterance, and take corresponding actions. This capability is referred to as natural language understanding (NLU). A common NLU system parses the input text, generates a parse tree, and interprets the text segment corresponding to each tree node based on rules linked to that node. The parser usually employs its own grammar. In an application that combines ASR and NLU, the input of the NLU module are the recognized words. If the grammar for recognition is the same as the grammar for NLU we can directly use the recognizer as the parser. This, in turn, can save significant overhead and implementation. Although limited, this approach can be useful in many cases, especially for command-and-control and form filling applications.

In DynaSpeak, the support for hierarchical grammar structures allows for a straightforward implementation of this approach, which we call *slot filling*. A slot refers to a subgrammar of semantic interest. The goal of slot filling is to find the portions of the recognition hypothesis that correspond to each of the slots, and then to process them accordingly. In our implementation, we created a special grammar-level backtrace tag identifying grammar names, in addition to the word-level tags needed by the common word recognition. During search, we add grammar-level tags to the backtrace array on entering and exiting each subgrammar. At the end of the search, by retrieving the backtrace array, we can easily determine which segments of the decoded word string correspond to which subgrammars, thus identifying the slot fillers. In the grammar optimization phase, we deliberately preserve the subgrammars corresponding to slots from being expanded, while optimizing other subgrammars for speed. In addition, the DynaSpeak recognizer can output the parse tree along with the recognition hypothesis, to enable more sophisticated NLU postprocessing.

## 8. IMPLEMENTATION ISSUES

Despite rapid advances, today's hardware platforms for most embedded and mobile computing systems still do not meet the requirements for full-fledged state-of-the-art automatic speech recognition technology, mostly with regard to memory size and processing speed. Also, hardware floating point support is often missing. This gap creates significant engineering challenges to developers, who must carefully consider implementation issues to minimize resource requirements and performance loss. At the same time, the recognizer's architecture should be scalable enough to take advantage of hardware improvements in the future. In DynaSpeak, all aspects of the recognition computation, including front-end processing, observation probability density evaluation, and search are implemented in integer arithmetic. Most model parameters are stored in one-byte-long integers. Runtime estimation of the mean and variance of the acoustic features is used for feature normalization. This allows the model parameters to be kept within a uniform range that is suitable for byte encoding. The implementation is designed to be portable to floating point operation without difficulty when more powerful processors become available.

As the recognizer dynamically scales its use of resources depending on grammar and vocabulary size, we can seamlessly use DynaSpeak in a wide range of tasks, from simple command-and-control applications, to recognition of natural, unconstrained speech using large-vocabulary class-based statistical N-gram LMs. The recognizer has a multithread-safe implementation that allows multiple execution threads to be run efficiently, each with different acoustic models and grammars. The code, written in C++, is modular and portable, allowing rapid migration to new hardware and software platforms. Currently, Unix, Linux, and all versions of Windows are supported on PCs. Windows CE and embedded Linux are supported on Pocket PCs.

## 9. APPLICATION EXAMPLE

One of the first applications based on DynaSpeak is a phrase translation system, the "Phraselator" [12]. The goal of the system was to provide the capabilities of a voice-activated phrasebook, such that a user would utter a phrase from a set of allowable phrases, the phrase would be recognized among the possible alternatives, and an associated prerecorded translation would be played back. The moderate computational demand and the ubiquitous target environment for such a system make it an ideal application for the current generation of high-end PDAs. The main challenge in developing this system was to achieve real-time speaker-independent recognition for about 700 simultaneously active phrases. We used compressed PSTM acoustic models similar to those described in Section 3.1. These models were trained on 270,000 sentences of general, large-vocabulary English uttered by 7292 speakers.

We explored several recognition strategies to tackle the phrase recognition task: 1. Use a finite state grammar where each sentence is explicitly represented by concatenating the corresponding word models in a unique path; the recognized word sequence immediately corresponds to the recognized phrase. 2. Use a bigram grammar trained on the allowable phrases for recognition of the input utterance; the recognized word string is then matched against the allowable phrase set. The best lexical match is defined as the recognized phrase. 3. Use an approach similar to approach 2, but add a second pass of acoustic rescoring by using a lexical tree HMM formed from the top N candidate sentences obtained from the lexical match following the first recognition pass.

We evaluated the three recognition strategies on a test set of 1000 sentences uttered by 10 speakers. The SER was 6.0%, 11.3%, and

7.1%, respectively. These results favored approach 1, the sentence-based grammar, which was also slightly faster than approach 3. Most of the DynaSpeak features described in earlier sections were critical to the success of this application: grammar optimization, Gaussian and search VQ-based pruning for significant speed gains, speaker and environment adaptation for robustness, and dynamic loading of grammars, to allow on-the-fly change of phrase sets to deal with different translation domains. Given the speaker-independent and phonetic nature of the acoustic models, it is straightforward to create phrase sets for new domains with relatively simple tools. It is also easy to add sentence variations by either defining new sentences associated with the same translation, or by modeling each set of semantically equivalent sentences by a JSGF subgrammar.

## 10. SUMMARY

We have given a brief description of SRI's new DynaSpeak$^{TM}$ speech recognition engine. Scalability is one of the most important features in DynaSpeak's design and implementation. DynaSpeak supports Gaussian mixture model and hidden Markov model based acoustic models, and has several techniques for building small, yet accurate, models. For language modeling, it supports both rule-based grammars, specified as Java$^{TM}$ speech grammar format (JSGF), and statistical LMs, represented in the DECIPHER$^{TM}$ probabilistic finite state grammar (PFSG) format. DynaSpeak uses a dynamic search strategy based on a hierarchical data structure across grammar, word, phone, and state representations, which generally ensures a small memory footprint. In addition, it provides convenient support for class-based statistical language models. DynaSpeak has an efficient and flexible grammar optimization algorithm that allows the user to improve decoding speed according to the available memory resources. With additional pruning techniques in both Gaussian computation and search, DynaSpeak can achieve real-time recognition on complex tasks. It also provides natural language support, which allows users to develop speech-enabled applications with ease. To provide robustness to environment and speaker variability, DynaSpeak has an online adaptation capability that adjusts the acoustic models to match the input speech during use. DynaSpeak has been designed to run on current mobile computing hardware, without needing floating point support; it also remains highly portable to newly emerging platforms. In sum, the DynaSpeak ASR capabilities provide an effective enabling technology for today's and tomorrow's speech-powered mobile computing applications.

## 11. REFERENCES

[1] Digalakis, V., Monaco, P., and Murveit, H. Genones: Generalized Mixture Tying in Continuous Hidden Markov Model-Based Speech Recognizers. *IEEE Trans. Speech and Audio Processing* 4(4), 281-289, July 1996.

[2] Sankar, A. Experiments with a Gaussian Merging-Splitting Algorithm for HMM Training for Speech Recognition. In *Proc. DARPA Broadcast News Transcription and Understanding Workshop* (Lansdowne, VA), pp. 99-104, February 1998.

[3] Gupta, S., Soong, F., and Hami-Cohen, R. Quantizing Mixture Weights in a Tied-Mixture HMM, In *Proc. ICSLP* (Phildelphia, PA), pp. 1828-1831, 1996.

[4] Brown, P.F., Pietra, V.J.D., deSouza, P.V., Lai, J. C., and Mercer, R.L. Class-Based n-gram Models of Natural Language. *Computational Linguistics* 18(4), 467-481, December 1992.

[5] Stolcke, A., SRILM -- An Extensible Language Modeling Toolkit. In *Proc. ICSLP* (Denver, CO), vol. 2, pp. 901-904, 2002.

[6] Sankar, A., and Lee, C.-H. Stochastic Matching for Robust Speech Recognition. *IEEE Signal Processing Letters* 1, 124-125, August 1994.

[7] Digalakis, V. V., Rtischev, D., and Neumeyer, L. G. Fast Speaker Adaptation Using Constrained Reestimation of Gaussian Mixtures. 3(5), 357-366, September 1995.

[8] Renals, S. Phone Deactivation Pruning in Large Vocabulary Continuous Speech Recognition. *IEEE Signal Processing Letters* 3, 4-6, 1996.

[9] Mohri, M. Finite-state Transducers in Languages and Speech Processing. *Computational Linguistics* 23(2), 269-312, 1997.

[10] Weng, F., Stolcke, A., and Sankar, A. Efficient Lattice Representation and Generation. In *Proc. of ICSLP* (Sydney, Australia), vol. 6, pp. 2531-2534 November 1998.

[11] Zheng, J., and Franco, H. Fast Hierarchical Grammar Optimization Algorithm Towards Time and Space Efficiency. In *Proc. of ICSLP* (Denver, CO), pp. 393-396, September 2002.

[12] Phraselator. http://www.phraselator.com.