# Design of the CMU Sphinx-4 Decoder

Lamere, P.; Kwok, P.; Walker, W.; Gouva, E.; Singh, R.; Raj, B.; Wolf, P.

TR2003-110 August 2003

## Abstract

The decoder of the sphinx-4 speech recognition system incorporates several new design strategies which have not been used earlier in conventional decoders of HMM-based large vocabulary speech recognition systems. Some new design aspects include graph construction for multilevel parallel decoding with independent simultaneous feature streams without the use of compound HMMs, the incorporation of a generalized search algorithm that subsumes Viterbi and full-forward decoding as special cases, design of generalized language HMM graphs from grammars and language models of multiple standard formats, that toggles trivially from flat search structure to tree search structure etc. This paper describes some salient design aspects of the Sphinx-4 decoder and includes preliminary performance measures relating to speed and accuracy.

*Eurospeech 2003*

**Publication History:**

1. First printing, TR-2003-110, August 2003

# DESIGN OF THE CMU SPHINX-4 DECODER

Paul Lamere[1], Philip Kwok[1], William Walker[1], Evandro Gouvêa[2], Rita Singh[2],
Bhiksha Raj[3], Peter Wolf[3]

[1]Sun Microsystems, [2]Carnegie Mellon University, [3]Mitsubishi Electric Research Labs, USA

## ABSTRACT

The decoder of the sphinx-4 speech recognition system incorporates several new design strategies which have not been used earlier in conventional decoders of HMM-based large vocabulary speech recognition systems. Some new design aspects include graph construction for multilevel parallel decoding with independent simultaneous feature streams without the use of compound HMMs, the incorporation of a generalized search algorithm that subsumes Viterbi and full-forward decoding as special cases, design of generalized language HMM graphs from grammars and language models of multiple standard formats, that toggles trivially from flat search structure to tree search structure etc. This paper describes some salient design aspects of the Sphinx-4 decoder and includes preliminary performance measures relating to speed and accuracy.

## 1. INTRODUCTION

The Sphinx-4 speech recognition system is a state-of-art HMM based speech recognition system being developed on open source (cmusphinx.sourceforge.net) in the Java™ programming language. It is the latest addition to Carnegie Mellon University's repository of Sphinx speech recognition systems. The Sphinx-4 decoder has been designed jointly by researchers from CMU, SUN Microsystems and Mitsubishi Electric Research Laboratories. Over the last few years, the demands placed on conventional recognition systems have increased significantly. Several things are now additionally desired of a system, such as the ability to perform multistream decoding in a theoretically correct manner, with as much user control on the level of combination as possible, that of at least some degree of basic easy control over the system's performance in the presence of varied and unexpected environmental noise levels and types, portability across a growing number of computational platforms, conformance to widely varying resource requirements, easy restructuring of the architecture for distributed processing *etc*. The importance of a good and flexible user interfacing is also clear as a myriad of devices attempt to use speech recognition for various purposes. The design of the Sphinx-4 is driven by almost all of these current-day considerations, resulting in a system that is highly modular, portable and easily extensible, while at the same time incorporating several desirable features by extending conventional design strategies or inventing and incorporating new ones. Its design is quite a bit more utilitarian and futuristic than most existing HMM-based systems.

This paper describes selected set of important design innovations in the Sphinx-4 decoder. The sections are arranged as follows: section 2 describes the overall architecture of the decoder, and some software aspects. Section 3 describes the design of the graph construction module and design of the language-HMM graph for parallel decoding. Section 4 describes the design of the search module, and the generalized Bushderby [2] classification algorithm used in it. Section 5 describes the design of the frontend and acoustic scorer. Section 6 presents some performance measures and lists the future remaining work on the decoder.

## 2. OVERALL ARCHITECTURE

The Sphinx-4 architecture has been designed with a high degree of modularity. Figure 1 shows the overall architecture of the system. Even within each module shown in Figure 1, the code is extremely modular with easily replaceable functions.

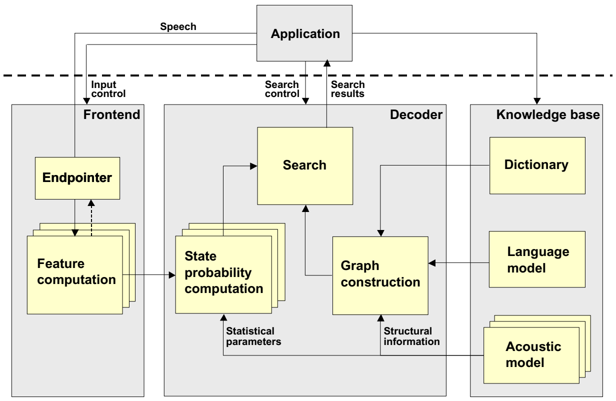

**Figure 1.** Architecture of the Sphinx-4 system. The main blocks are Frontend, Decoder and Knowledge base. Except for the blocks within the KB, all other blocks are independently replaceable software modules written in Java. Stacked blocks indicate multiple types which can be used simultaneously

There are three main blocks in the design, which are controllable by any external application: the frontend, decoder, and knowledge base (KB). The frontend module takes in speech and parametrizes it. Within it, the endpointer module can either endpoint the speech signal, or the feature vectors computed from it. The decoder block performs the actual recognition. It is comprised of a graph construction module, which translates any type of standard language model provided to KB by the application into an internal format and, together with information from the dictionary and structural information from one set or a set of parallel acoustic models, constructs the language HMM. The latter is then used by the search module to decide the allowed path-extensions in the trellis which is searched. The trellis is not explicitly constructed.Rather, it is an implicit entity as in conventional decoders. The application can tap the information in the tokens at each node to get search results at various levels (such as state, phone or word-level lattices and segmentations). The application can also control the level at which scores from parallel feature streams are combined during search, and how each information stream is pruned. The search module requires likelihood scores for any current feature vector to generate the active list. Likelihoods are computed by the state probability computation module which is the only module that has access to the feature vectors. Score computation is thus an on-demand task, carried out whenever the search module communicates a state identity to the scoring module, for which a score for the current feature (of a specific type) is desired. In Sphinx-4, the graph construction module is also called the linguist, and the score computation module is called the acoustic scorer

The system permits the use of any level of context in the definition of the basic sound units. One by-product of the system's modular design is that it becomes easy to implement it in hardware.

**Programming language**: The system is entirely developed on the Java™ platform which is highly portable: once compiled, the bytecode can be used on any system that supports the Java platform. This feature permits a high degree of decoupling between all modules. Each module can be exchanged for another without requiring any modification of the other modules. The particular modules to be used can be specified at run time through a command line argument, with no need to recompile the code. Also, the garbage collection (GC) feature of Java simplifies memory management greatly, memory management is no longer done through explicit code. When a structure is no longer needed, the program simply stops referring to it. The GC frees all relevant memory blocks automatically. Java also provides a standard manner of writing multithreaded applications to easily take advantage of multi-processor machines. Also, the Javadoc™ tool automatically extracts information from comments in the code and creates html files that provide documentation about the software interface.

## 3. GRAPH CONSTRUCTION MODULE

In an HMM-based decoder, search is performed through a trellis, a directed acyclic graph (DAG) which is the cross product of a *language-HMM* and time. The language-HMM is a series-parallel graph, in which any path from source to sink represents the HMM for a valid word sequence in the given language. This graph has loops, permitting word sequences of arbitrary lengths. Language probabilities are applied at transitions between words. The language-HMM graph is a composition of the language structure as represented by a given language model, and the topological structure of the acoustic models (HMMs for the basic sound units used by the system). Figure 2 shows the relation between acoustic models, language-HMM, and the trellis which is searched, and also shows intermediate level graphs which are used implicitly in the composition of the language-HMM.

The graph construction module in Sphinx-4, also called the *linguist,* constructs the language-HMM using the output of another module which interprets the language model provided by the application as a part of the KB, and converts it into a single internal grammar format. This permits the external grammar to be provided by the application in any format, such as statistical N-gram, CFG, FSA, FST, simple word lists *etc*. The internal grammar is a literal translation of the external representation. For word lists all words are linked parallely to a single source and sink node and a loopback is made from the sink to the source. For N-gram LMs an explicit bigram structure is formed where every word is represented by a node, and there are explicit links from every node to every other node. The language-HMM accommodates parallel feature streams as shown in Figure 3. The design does not use compound HMMs [3] as in conventional systems, but maintains separate HMMs for the individual feature streams. Time-synchronization of paths is ensured at the boundaries of combined units, during search.

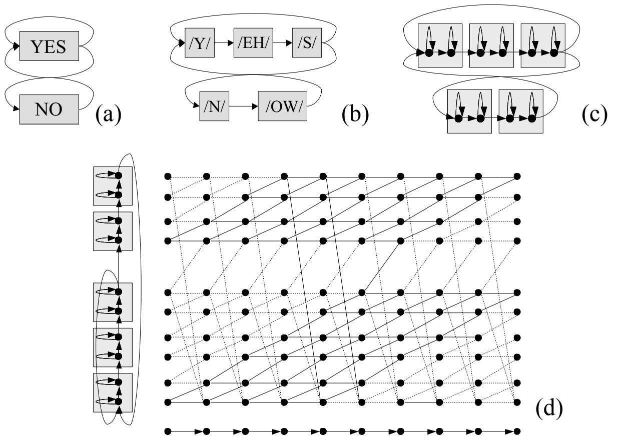

**Figure 2.** (a) Language graph for a simple language with a two word vocabulary. (b) Language-phonetic graph derived from (a). (c) Language-phonetic-acoustic graph derived from (b). (d) Trellis formed as the crossproduct of (c) and a linear graph of observation data vectors.

The internal grammar is then converted to a language-HMM by another module, which is independent of the grammar construction module. Note that in the architecture diagram both modules are represented by a single graph construction module. In the formation of the language-HMM, the word-level network of the internal grammar is expanded using the dictionary and the structural information from the acoustic models. In this graph, sub-word units with contexts of arbitrary length can be incorporated, if provided. We assume, however, that silence terminates any context - sub-word units that are separated from each other by an intermediate silence context cannot affect each other. The linguist converts the word graph to a Language-HMM either dynamically or statically. In dynamic construction, the HMMs for grammar nodes that can follow a current node and are not already instantiated are constructed at run time using the appropriate context dependent phonemes at the word boundaries. In static construction, the entire language-HMM is constructed statically. HMMs are constructed for all words in the vocabulary. Each word HMM is composed with several word-beginning context dependent phones, each corresponding to possible crossword left context. Similarly, each word HMM also has several word-ending context dependent phonemes. Each word is connected to every other word. In linking any two words, only the appropriate context-dependent phones are linked.

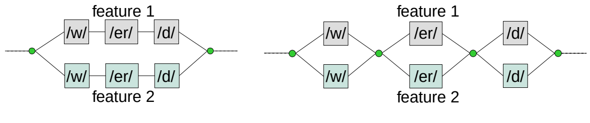

**Figure 3.** Graph construction with two parallel features.In the left panel scores for words are combined at word boundaries. In the right panel, they are combined at phoneme boundaries.

There are a number of strategies that can be used in constructing the language-HMM that affect the search. By altering the topology of the language-HMM, the memory footprint, perplexity, speed and recognition accuracy can be affected. The modularized design of Sphinx-4 allows different language-HMM compilation strategies to be used without changing other aspects of the search.

The choice between static and dynamic construction of language HMMs depends mainly on the vocabulary size, language model complexity and desired memory footprint of the system, and can be made by the application

# 4. SEARCH MODULE

Search in Sphinx-4 can be performed using conventional Viterbi algorithm, or full-forward algorithm, so far as the latter can indeed be approximated in the compact language HMM graphs used by conventional. HMM-based decoders, including sphinx-4. However, the unique feature about search in Sphinx-4 is that these conventional algorithms are merely special cases of a more general algorithm called Bushderby [2], which performs classification based on free energy, rather than the Bayesian rule. Likelihoods are used in the computation of free energy, but do not constitute the main objective function used for classification. The theoretical motivations for this algorithm are described in [2].

From an engineering perspective, the algorithm can be viewed as the application of a single-parameter function in the trellis which operates on path scores. The parameter can take meaningful values in $(0, \infty]$, and is user-controllable. Mathematically, the function performs an $\eta$ -norm over the score of a set of $\pi$ edges incident on a node $U$ in the trellis that is being searched for the best hypothesis as:

$$\mathrm{score}(U) = \left( \sum_{\pi \in U} \mathrm{prob}(\pi)^{\eta} \right)^{\frac{1}{\eta}} \qquad (1)$$

When $\eta = 1$, this reduces to full-forward or Bayesian decoding:

$$\mathrm{score}(U) = \sum_{\pi \in U} \mathrm{prob}(\pi) \qquad (2)$$

When $\eta = \infty$, it reduces to viterbi decoding:

$$\mathrm{score}(U) = max_{\pi \in U}(prob(\pi)) \qquad (3)$$

For values of $\eta$ which are not equal to either 1 or $\infty$ , the expression in equation (1) has no bayesian interpretation. However, it can be related to free energy. Classification over mismatched data can directly be controlled through this Bushderby parameter, and has been shown to yield significant improvements in recognition performance.

The search module constructs a tree of hypotheses using a token-passing algorithm, which is used in many conventional decoders [1]. The token tree consists of a set of tokens that contain information about the nodes traversed in the trellis and provides a complete history of all active paths in the search. Each token contains the overall acoustic and language scores of the path at a given point, a *Language HMM* reference, an input feature frame identification, and a reference to the previous token, thus allowing backtracing. The Language HMM reference allows the search manager to relate a token to its senone, context-dependent phonetic unit, pronunciation, word, and grammar state. The search module also communicates with the state probability estimation module, also called the *acoustic scorer,* to obtain acoustic scores for the current data, which are only seen by the acoustic scorer.

This module maintains an *active list.*list of tokens. Active lists are selected from currently active nodes in the trellis through pruning. Sphinx-4 performs both relative and absolute pruning, and also pruning for individual features when decoding with parallel feature streams. The pruning thresholds are controllable by the application. Search can be performed in either *depth-first* or *breadth-first* manner. Depth-first search is similar to conventional stack decoding, where the most promising tokens are expanded in time sequentially. Thus, paths from the root of the token tree to currently active tokens can be of varying lengths. In breadth-first search, all active tokens are expanded synchronously, making the paths from the root of the tree to the currently active tokens of equal in length. The Bushderby algorithm used during search currently applies to breadth-first search.

Sphinx-4 provides a beam pruner that constrains the scores to a configurable minimum relative to the best score, while also keeping the total number of active tokens to a configurable maximum. New implementations can easily be created that provide alternative methods of storing and pruning the active list. The garbage collector automatically reclaims unused tokens, thus simplifying the implementation of the pruner by merely allowing a token to be removed from the set of active tokens.

When parallel streams of features are being decoded, token stacks are used at the node sin the trellis. The search module combines weighted scores from the features at appropriate nodes, at the specific level in the graph as specified by the application. Scores are combined from tokens which have identical word histories, word entry, and exit times. This is done under the assumption that parallel features derived from the speech signal must at least traverse a word in the speech signal at concurrent times For decoding with parallel feature streams, the pruning strategy has to be very carefully designed, as it can lead to serious problems during search. A schematic representation of such a problem for the case of two feature streams is shown in Figure 4. To avoid this problem, the search module switches to a different pruning scheme in the case of parallel features. In this scheme, pruning is done at two levels. One level consists of wide-beam or loose pruning which is done separately for separate features. This is possible since each token in the stack at each node carries scores for individual features as well as the combined score. The second level of pruning is done at the specific user defined levels at which scores combine, and is done only with respect to the combined scores. This pruning uses much narrower beams. Also, features are combined in a weighted manner, with weights that can be application controlled either directly or through any application-enforced learning algorithm.

The result generated by the search module is in the form of a token tree, which can be queried for the best recognition hypotheses, or a set of hypotheses. the tree also encodes the recognition lattice at all levels, senones, phoneme, word etc.

# 5. FRONT END AND ACOUSTIC SCORER

The sequence of operations performed by the Sphinx-4 front-end module in its default configuration is very similar to what other speech recognition systems do, creating mel-cepstra from an audio file. It is however parallelizable, and can currently simultaneously

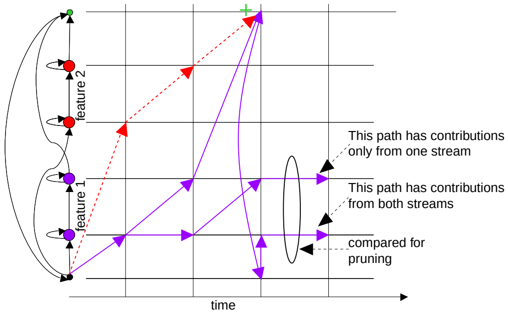
feature 2
feature 1
time
This path has contributions only from one stream
This path has contributions from both streams
compared for pruning

**Figure 4.** The pruning problem encountered by the search module in decoding parallel feature streams. If pruning is based on combined scores, paths with different contributions from the multiple feature streams get compared for pruning.

compute MFCC and PLP cepstra from speech signals, with easy extensibility to other feature types.

The module is organized as a sequence of independent replaceable communicating blocks, each with an input and output that can be tapped. (*e.g.*, a pre-emphasizer, a spectrum analyser). Thus outputs of intermediate blocks can be easily fed into the state computation module, for example. Features computed using independent sources by the application, such as visual features, can also be directly fed into the State probability computation module, either in parallel with the features computed by the frontend for parallel decoding, or independently. Moreover, one can easily add a processor between two existing processors, making it very easy to add a noise cancellation or compensation module.

The communication between blocks follows a *pull* design. In this design, a block requests input from an earlier block when needed, as opposed to the more conventional *push* design, where a block gives data to the succeeding block as soon data are available to it. At a global level, in a pull design, more speech is captured only when recognition is requested. In a push design, recognition is requested after speech is captured.

Each block operates in response to control signals which are interpreted from the data requested from the predecessor. The control signal might indicate the beginning or end of speech, might indicate data dropped or some other problem. If the incoming data are speech, they are processed and the output is buffered, waiting for the successor block to request it. Handling of control signals such as start or end of speech are essential for livemode operation. This design allows the system to be used in live or batchmode without modification.

In addition to being responsive to a continuous stream of input speech, the frontend is capable of three other modes of operation: (a) fully endpointed, in which explicit begining and end points of a speech signal are sensed (b) click-to-talk, in which the user indicates the beginning of a speech segment, but the system determines when it ends, (c) push-to-talk, in which the user indicates both the beginning and the end of a speech segment. Currently, endpoint detection is performed by a simple algorithm that compares the energy level to three threshold levels. Two of these are used to determine start of speech, and one for end of speech.

**Acoustic scorer:** This is the State output probability computation module shown in the architecture diagram of Sphinx-4. Its operation is straight forward: given a set of acoustic models and a request from the search module to score a particular state, it performs the mathematical operations required for score computation. It matches the acoustic model set to be used against the feature type in case of parallel decoding with parallel acoustic models. There are no restrictions on the allowed topology for the HMMs used in parallel scoring.

The scorer retains all information pertaining to the state output densities. Thus, the search module need not know the scoring is done with continuous, semi-continuous or discrete HMMs. Any heuristic algorithms incorporated into the scoring procedure for speeding it up can be performed locally within the search module.

## 6. EXPERIMENTAL EVALUATION

The performance of Sphinx-4 is compared with Sphinx-3 on the on the speaker independent portion of the Resource Management database (RM1) [4] in Table 1. Sphinx-3 builds the language-HMM dynamically, while the code for dynamic construction in Sphinx-4 is not yet fully operational. In any decoder, static construction of a language-HMM takes far longer than dynamic construction. Hence graph construction times have been factored out of the real-time numbers reported in Table 1. Acoustic models were 3 state, 8 Gaussians/state HMMs with 1000 tied states, trained with the RM training data using the training module of Sphinx-3. Test results are reported using statistical N-gram language models: a flat unigram, a unigram, and a bigram. The Lams were created from the LM training data provided with the RM database. Sphinx-4 is currently also capable of working from an external FST language model. We do not report those results here, although they are consistently better than n-gram results.

Table 1 shows both word error rate (WER) and decoding speed in times real time. All experiments were run on a Sun Blade™ 1000 workstation with dual 750 MHz UltraSPARC® III processors.

| Type of N-gram LM | Speed (xRT) | | WER (%) | |
|---|---|---|---|---|
| | S3 | S4 | S3 | S4 |
| Flat unigram | 3.0 | 13.7 | 18.7 | 17.3 |
| Unigram | 2.9 | 15.3 | 13.1 | 14.5 |
| Bigram | 2.8 | 19.0 | 2.0 | 3.3 |

**Table 1:** Performance comparison between Sphinx-3 (S3) and Sphinx-4 (S4). The speeds do not include loading time, which are vastly different for s3 (dynamic language-HMM construction) and s4 (currently static language-HMM construction)

The Sphinx-4 performance has not been optimized on the bigram and trigram tasks at the time of this submission. As such, the evaluation of medium vocabulary tasks is ongoing, and large vocabulary tasks will be approached shortly. The optimized trigram tasks and the completed medium and large vocabulary evaluations will be completed by the time the paper is presented. They will also be reported on SourceForge as and when they are completed.

## ACKNOWLEDGEMENTS

We thank Prof. Richard Stern at CMU, Robert Sproull at Sun Microsystems, and Joe Marks at MERL, for making this team possible. We also thank Sun Microsystems and the current management for their continued support and collaborative research funds. Rita Singh was sponsored by the Space and Naval Warfare Systems Center, San Diego, under Grant No.

N66001-99-1-8905. The content of this paper does not necessarily reflect the position or the policy of the US Government, and no official endorsement should be inferred.

# REFERENCES

[1] S.J. Young, N.H.Russel, and J.H.S. Russel “Token passing: A simple conceptual model for connected speech recognition systems,” *Tech. Report*, Cambridge Univ. Engg. Dept., 1989

[2] R. Singh, M. Warmuth, B. Raj, and P. Lamere “Classification with free energy at raised temperatures,” in *EUROSPEECH 2003,* 2003.

[3] Compound HMM reference.............

[4] P. Price, W.M. Fisher, J. Bernstein, and D.S. Pallett, “The DARPA 1000-word resource management database for continuous speech recognition,” in *Proc. ICASSP 1988,* Vol. 1. p. 651-654, 1988.