A FLEXIBLE, SCALABLE FINITE-STATE TRANSDUCER ARCHITECTURE FOR CORPUS-BASED CONCATENATIVE SPEECH SYNTHESIS[1]

Jon R. W. Yi, James R. Glass, and I. Lee Hetherington

Spoken Language Systems Group
Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139 USA
{jonyi, jrg, ilh}@sls.lcs.mit.edu

ABSTRACT

In this paper we describe our work involving the conversion of our phonologically-based synthesizer into a finite-state transducer (FST) representation which can be used for real-time natural-sounding synthesis. We have designed a transducer structure to efficiently perform the common task of unit selection in concatenative speech synthesis. By encapsulating domain-independent concatenative synthesis costs into a constraint kernel, we have obtained a topology that scales linearly with the size of the synthesis corpus. The FST representation provides a flexible, unified framework in which we can leverage our previous work in speech recognition in areas such as pronunciation modelling and search. The FST synthesizer has been incorporated into two servers which operate within our conversational system architecture to convert meaning representations into waveforms. We have had preliminary success with the new FST-based synthesis in several constrained spoken dialogue applications.

1. INTRODUCTION

Corpus-based concatenative methods and unit selection mechanisms have recently received increasing attention in the speech synthesis community (e.g., [4, 6]). Our previous work with speech synthesis has focused on using unit selection and waveform concatenation techniques to synthesize natural-sounding speech for constrained spoken dialogue domains [10]. Following Hunt and Black (e.g., [4]), we organized the synthesis constraints of unit selection into concatenation and substitution costs, which essentially prioritize where the speech signal is spliced and which units are appropriate for concatenation. Our concatenative synthesizer, called ENVOICE, is phonologically-based (i.e., symbolic), uses phones as the fundamental synthesis unit, and selects variable-length segments for concatenation. When combined with a domain-dependent corpus, it has produced very natural sounding speech for several of our spoken dialogue domains.

Our recent work has revolved around developing a more general framework for this synthesizer that would be easier to maintain, extend, and deploy. Following our successful use of finite-state transducers (FSTs) for speech recognition [3], we have also converted our synthesizer to an FST-based representation. Synthesis is now modeled as a composition of five FST components: the word sequence, *W*, a lexicon, *L*, containing baseform pronunciations, a set of phonological rules, *P*, a set of transition phones, *T*, (used primarily for search efficiency), and a synthesis component, *S*, which maps all possible phone sequences to waveform segments for a given speech corpus. Unit selection is accomplished with a Viterbi or *N*-best search. Just as we, and others, have found for speech recognition, the FST formulation provides clarity, consistency, and flexibility. In our case, it also allows leveraging off of our previous work with FST-based speech recognition (e.g., pronunciation modelling and search).

One of the main challenges of converting our phonological synthesis framework to an FST representation was determining an efficient structure for the synthesis FST, *S*. Our solution was to introduce a series of domain-independent intermediate layers, which efficiently encapsulate both the substitution and concatenation costs between every phonetic segment in a speech corpus. This structure has the property that the size of the intermediate layers is fixed, so that the size of the FST grows linearly with the size of the corpus. This allows us to avoid pruning mechanisms which would need to be implemented if we had directly connected every segment in the corpus [2]. Furthermore, this synthesis framework incorporates both concatenation and substitution costs as part of the FST, and not just concatenation costs.

In the following sections, we first describe our efforts in converting our phonologically-based synthesis framework to an FST representation. We then lay out the search algorithm and techniques used to speed up the search. Next, we explain the issues we encountered in implementing FST-based speech synthesis as a real-time server in our GALAXY conversational system architecture. Finally, we discuss ongoing synthesis research.

2. FST REPRESENTATION

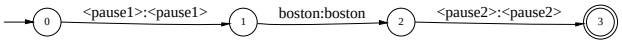
One of the key reasons we adopted an FST representation is its ability to completely and concisely capture finite-state constraints. Given a permissible input sequence, an FST can generate a graph of output sequences [5]. The process of transforming an input language into an output language is guided by the states and arcs specified in the FST topology. Arcs have optional input and output labels (the absence of a label is represented by the ϵ symbol), and can also have weights associated with them. For example, a lexical dictionary can be implemented with an FST that maps words to phonemes, with weights possibly being used to model alternative pronunciations. Since FSTs can be cascaded in succession to effect further mappings, a second phoneme-to-phone FST could be used to transduce the phonemic sequence

[1]This research was supported by DARPA under Contract N66001-99-1-8904 monitored through Naval Command, Control and Ocean Surveillance Center, and contract DAAN02-98-K-003, monitored through U.S. Army Natick Research Development and Engineering Center.

into a phonetic sequence. The overall composition would then map words to phones.

With this representation, unit selection for concatenative speech synthesis can be modeled as a mapping from words to waveform segments. Five FST components, *W*, *L*, *P*, *T*, and *S*, perform the intervening steps. This factoring allows us to independently model, design, and refine the underlying processes. When combined with a search, these components work in succession from an input sequence of words to produce a sequence of waveform segments suitable for concatenation.

The first FST component, *W*, consists of a trivial chain topology mapping (optional) input sequences to output word sequences. For illustrative purposes, we show an example of *W* for the word 'Boston' containing four states and three arcs, with input and output labels (separated by a colon) on each arc.



To obtain a pronunciation for 'Boston', *W* is *composed* with the lexical modelling portion consisting of *L* and *P*. Typically, the lexicon and phonological rule components are composed (i.e., $L \circ P$) and optimized in advance for efficiency. Mapping words to phones, they are transposed versions of what is used in our speech recognizer for lexical decoding. The following pronunciation graph ($W \circ (L \circ P)$) is generated:

As will be explained in more detail in the next section, the phonetic pronunciation graph is embedded with transition phones in order to improve search efficiency. Adjacent phones are rewritten via FST composition to contain intervening transition phones (e.g., $/\alpha/\ /\beta/ \rightarrow /\alpha/\ \alpha|\beta\ /\beta/$). This is accomplished with the transition FST, T. The final graph of phonetic pronunciation embedded with transition phones, $(W \circ (L \circ P)) \circ T$, is optimized for subsequent composition in the unit selection search.

The final ingredient in FST-based unit selection is the synthesis FST, S, which maps phones (with transition phones) to waveform segments. Desirable properties of S include low perplexity (average branching factor) and scalability. Our solution decouples the domain-independent and domain-dependent portions residing in S. Specifically, domain-independent costs relating only to the pertinent language form a constraint kernel that encodes speech knowledge for concatenative speech synthesis. Individual waveforms are represented as finite-state phone chains and are connected to the constraint kernel at the phone level.

2.1. Constraint Kernel Topology

In Figure 1, we see a diagram depicting four intermediate layers of the constraint kernel, along with portions of two corpus utterances (top and bottom). Some details have been omitted for clarity (e.g., output labels of corpus waveform arcs). In the example shown, the layers bridge a pair of source ('Bosnia') and destination ('Austin') utterances to synthesize 'Boston'. The outermost layers of the kernel (1 and 4) serve to multiplex and demultiplex transitions to and from waveform segments in the synthesis corpus, respectively. Each state in these two layers connects all instances of a transition between two particular phones in the corpus. Because our synthesizer is phonologically-based, the concatenation and substitution costs depend only on the local phonetic context of the transition. For example, the illustrated layer 1 state in Figure 1 gathers all instances of transitions between /ɔ/ and /z/ in the corpus (with one of them being in the word 'Bosnia'). Likewise, the illustrated layer 4 state connects to all instances of transitions between /ɔ/ and /s/ in the corpus (with one of them being in the word 'Austin'). In general, if there are P phone labels in the corpus, there are P^2 states in each of these two layers.

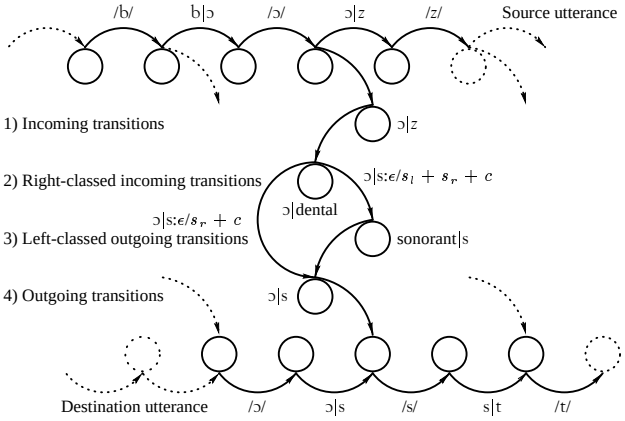


Figure 1: Topology of constraint kernel

For complete generality, layers 1 and 4 should be fully interconnected so that any state in layer 1 can connect to any state in layer 4 with an associated concatenation cost, c, and substitution cost, s. A full interconnection would require P^4 arcs between these two layers. Due to the design of our phonologically-based synthesizer however [10], we are able to significantly reduce the number of arcs by introducing two additional layers (2 and 3). This is because the substitution cost is broken down into a left and right cost, s_l and s_r, respectively. The left cost is associated with matching the left phone of a layer 1 state with the left phone of a layer 4 state. The right cost is the penalty associated with matching the corresponding right phone labels. Equivalence classes are used to group phones into classes which exhibit similar concatenation behavior, and can thus share the same concatenation or substitution costs. The equivalence class used for the left phone depends on the right context, and vice versa.

In Figure 1 with the ɔ|z transition in layer 1 for example, the right equivalence class for the /z/ is determined by the /ɔ/. In this case, all consonants with an alveolar or dental place of articulation are deemed to have the same effect on the vowel, and are grouped into a dental equivalence class, as shown in layer 2. Similarly, with the ɔ|s transition in layer 4, the left equivalence class for the /ɔ/ is determined by the /s/. In this case, all sonorants are deemed to have the same effect on the fricative, and are grouped into a sonorant equivalence class, as shown in layer 3. The cost of connecting these two states is the cost, s_l, of substituting an /ɔ/ as a sonorant, the cost, s_r of substituting an /s/ as a dental consonant, and the concatenation cost, c, of making a splice between a vowel /ɔ/ and a fricative /s/ (arc weights are delineated by a '/' in Figure 1). In this example, the substitution and concatenation are very reasonable, so the overall cost is small.

Each state in layer 1 connects to only one state in layer 2, and each state in layer 4 connects to only one state in layer 3. If there are an average of $\bar{C}$ equivalence classes per phone, then layers 2 and 3 each have a total of $\bar{C}P$ states. To provide complete generality, these two layers are fully connected. Since $\bar{C} << P$, this structure requires significantly fewer arcs than fully connecting all speech segments in a large corpus. The size of such a network would be quadratic with the number of phones in the corpus, and would require pruning to reduce the connectivity [2].

In order to account for making splices where the contexts match exactly, P^2 direct connections are made between states in layers 1 and 4 which have identical labels. The only cost associated with these arcs is the concatenation cost, c, which depends on the particular transition context. Similarly, there are $\bar{C}P^2$ direct connections between layers 2 and 4 for all cases where the left labels are identical. An example arc is illustrated in Figure 1 for this case since the /ɔ/ vowel is common between the two states. In this case the total cost is the right substitution cost, s_r, and the concatenation cost. Finally, there are also $\bar{C}P^2$ direct connections between layers 1 and 3 for all cases where the right labels are identical. The cost on these arcs is a concatenation cost plus a left substitution cost. The total number of arcs in the constraint kernel, $(P(\bar{C}+1))^2$, is therefore fixed, and is independent of the size of the speech corpus itself.

It should be noted that the cross-connections between layers 2 and 3, 1 and 3, 2 and 4, and 1 and 4, absorb the transition labels which are inserted between phones via the T FST described earlier. The use of the transition labels in this manner adds significant constraint to the topology of S, greatly reducing the number of active search nodes, and enabling real-time synthesis. Without the transition labels, all arcs in the constraint kernel would have an ϵ label and would have to be expanded during the search.

Information about speech utterances are stored as finite-state phone chains. For an utterance with N phones, there are an additional $N-1$ transition phones for a total of $2N-1$ proper and transition phone arcs. These $2N-1$ arcs of zero weight string the $2N$ states into a chain. Proper phone arcs emit waveform segment descriptors (not shown in Figure 1), whereas transition phone arcs produce no output. A descriptor specifies the token label, waveform filename, and start and end times of the waveform segment. That the arcs have zero weight allows the chain to be traversed without penalty; consequently, successively spoken speech segments can be concatenated without cost.

For every utterance with N phones, there are $2N$ connections made to the constraint kernel: N arcs connecting from the end of every phone to the matching transition state in layer 1, and N arcs connecting from the outgoing transition states of layer 4, to the start of the matching phone in the utterance. Note that another advantage of the transition label is to avoid considering self loops through the constraint kernel during the Viterbi search.

2.2. Search

The role of the search component is to find the least-cost sequence of speech utterance segments for a given text input. Specifically, the search finds the least-cost path through the composition of $(W \circ (L \circ P) \circ T)$ with S. In keeping with our parallelism between recognition and synthesis, the search we use is essentially the same Viterbi-style dynamic programming beam search that is used for recognition, except that different graphs are searched. For synthesis, we optimize $(W \circ (L \circ P) \circ T)$ and then walk through its states in topological order, exploring compatible arcs in S at the same time. Pruning consists of dynamic programming pruning plus score- and count-based beam pruning, which are tuned to achieve real-time synthesis.

To reduce latency when synthesizing a long system response, we break the response into chunks separated by sentence boundaries and explicitly referenced pauses or waveform segments called "shortcuts" (described in the next section). Since the state of the system is known at these boundaries, the searches for each chunk can be performed separately, allowing us to perform waveform output for one chunk while performing the search for a subsequent chunk.

3. IMPLEMENTATION

Based on the structures and algorithms described in the previous section, we have developed a set of software tools and servers for working with this new FST framework. These tools encompass the steps that are performed in the assembling, testing, and running of an FST-based concatenative speech synthesizer. We use command-line utilities for lexicon creation, constraints compilation, and corpus instrumentation, as well as for synthesis testing. The lexicon creation process reuses tools from our speech recognizer. The constraint kernel of the synthesis FST is compiled from substitution and concatenation costs matrices. The synthesis FST is then populated with phones from a corpus of time-aligned waveforms. FST synthesis can then be tested with arbitrary sequences of in-vocabulary words.

We have integrated FST synthesis as networked servers into the GALAXY COMMUNICATOR architecture which we use for all of our spoken dialogue systems [7]. Two servers fulfill the text-to-speech conversion component, and handle the separate tasks of unit selection and waveform concatenation. Based on a client-server architecture, they communicate with a central hub which coordinates all tasks in a conversational system. The two servers are pipelined and perform synthesis at speeds sufficient for interactive purposes.

Within our GALAXY COMMUNICATOR implementation, the first step of synthesis actually begins in our natural language generation server, GENESIS [1]. GENESIS recursively expands internal meaning representations into text strings which can be displayed directly on a display, or sent to a synthesis server. The server relies on a message or template file, a lexicon, and a set of rewrite rules to perform generation. For synthesis, the message file is identical to that used for text generation. The lexicon can optionally be modified to expand abbreviations, or explicitly represent waveform segments. These synthesis "shortcuts" allow the developer to bypass the search when desired, and provide backwards compatibility with our earlier word and phrase concatenation work. Another feature of GENESIS which we use for synthesis is the ability to specify features for entries in the lexicon. We have used this to help select words and syllables with the correct prosodic context.

The GENESIS rewrite rules can be used to perform text preparation for synthesis beyond what may be needed for text genera-

tion. For example, we have designed regular-expression rules that rewrite flight numbers and times originally in numerical form into written form (e.g., 6425 → sixty four twenty five, 11:05 → eleven oh five). This configuration performs the responsibilities typically assumed by a TTS text pre-processing stage. Because it is part of the generation component, it offers increased accuracy (e.g., unambiguous abbreviation expansion) and flexibility when developing multiple domains and languages.

In the next step of the synthesis chain, the unit selection server receives pre-processed text to synthesize from the natural language generation component. The word sequence is converted into a phonetic sequence by the lexical FST and prepared for searching by the transition label FST. If the word sequence is interrupted with waveform segment "shortcuts", phonetic context must be maintained before and after the waveform segment to ensure correctness of the search.

Based on the results of the unit selection search, the waveform concatenation component receives instructions to concatenate the appropriate waveform segments. Currently, concatenation is performed without signal processing. For performance considerations, the waveform concatenation server loads the entire corpus of utterances from disk into memory at startup time. As concatenation instructions are received, waveform samples are streamed to the output audio server. The waveform concatenation and output audio servers can be co-located for efficiency.

We have converted several of our domains to use the ENVOICE synthesizer we have developed [10]. The most recent system consists of the MERCURY air travel domain for flight information and pricing [8]. Synthesizer development typically begins once the natural language generation component has been completed for a displayful system. Since the synthesizer currently relies on a domain-dependent corpus, the most time-consuming process is usually the design of a set of prompts to be read. As utterances are recorded, they can be transcribed with a speech recognizer and inserted into the synthesis FST.

For domain specific synthesis, we have used both manual and semi-automatic means of designing recording prompts. For example, static responses are recorded as a whole. For covering responses with more dynamic content, we use an underlying generation template and fill it in with different vocabulary items, such as numbers, and names of cities and airlines. We have also experimented with semi-automatic means of selecting recording prompts. In the past, for the purpose of synthesizing proper nouns, we have used iterative, greedy methods to compactly cover an inventory of stress-marked, syllable-like units [10].

4. CONCLUSIONS AND ONGOING WORK

In this paper we have introduced a scalable, FST implementation for unit selection in concatenative speech synthesis. We have deployed this technology within real-time synthesis servers operating in the GALAXY COMMUNICATOR environment. It is integrated such that phrase, word, and sub-word approaches are combined seamlessly to produce natural-sounding real-time synthesis for constrained conversational domains. Synthesizers are developed using software that works in conjunction with our speech recognizer. We believe that this framework provides us with a flexible platform for future synthesis research.

The design of the FST representation was heavily influenced by our phonologically-based synthesizer whose costs are derived solely from symbolic contextual information. A consequence of this design is that it is currently not possible to introduce segment-specific concatenation or substitution costs based on acoustic information, as can be done elsewhere (e.g., [2, 4]). While we believe many cases do not require such a detailed distance metric, we are considering introducing a finer level of detail in the constraint kernel, which could incorporate quantized acoustic information.

There are many other issues which we plan to address in future work. In order to reduce the abruptness of some concatenation artifacts, we have begun to explore the use of signal processing techniques to modify both fundamental frequency and segment duration. To date, we have taken advantage of the constrained nature of outputs in our conversational domains, and have avoided the use of any kind of prosodic generation module. Prosody has mainly been incorporated at the lexical level in our GENESIS language generation module, and with our ongoing design of a general corpus for the natural-sounding synthesis of arbitrary words (e.g., proper nouns). We would like to investigate corpus-based prosodic generation in future work, however.

Finally, we are interested in developing synthesis capabilities for languages other than English, and are actively working on a version for Mandarin Chinese in a weather information domain [9]. This system is currently using syllable onsets and rhymes as the fundamental synthesis units with tokenized phrases as the lexical representation. We also have plans to work on Spanish and Japanese synthesizers in the near future.

Acknowledgments Scott Cyphers, Joe Polifroni and Stephanie Seneff helped to rectify many system and server issues.

5. REFERENCES

1. L. Baptist and S. Seneff, "GENESIS-II: A versatile system for language generation in conversational system applications," *these proceedings*.
2. M. Beutnagel, M. Mohri, and M. Riley, "Rapid unit selection from a large speech corpus for concatenative speech synthesis," *Proc. Eurospeech*, 607–610, Budapest, 1999.
3. J. Glass, T. J. Hazen, and L. Hetherington, "Real-time telephone-based speech recognition in the JUPITER domain," *Proc. ICASSP*, 61–64, Phoenix, 1999.
4. A. J. Hunt and A. W. Black, "Unit selection in a concatenative speech synthesis system using a large speech database," *Proc. ICASSP*, Atlanta, 373–376, 1996.
5. E. Roche and Y. Shabes (eds.), "Finite-State Language Processing," MIT Press, 1997
6. Y. Sagisaka, "Speech synthesis by rule using an optimal selection of non-uniform synthesis units," *Proc. ICASSP*, 679–682, New York, 1988.
7. S. Seneff, et al., "GALAXY-II: A reference arch. for conversational system development," *Proc. ICSLP*, 931–934, Sydney, 1998.
8. S. Seneff and J. Polifroni, "Formal and natural language generation in the MERCURY conversational system," *these proceedings*.
9. C. Wang, et al., "MUXING: A telephone-access mandarin conversational system," *these proceedings*.
10. J. Yi and J. Glass, "Natural-sounding speech synthesis using variable-length units," *Proc. ICSLP*, 1167–1170, Sydney, 1998.