Combining the flexibility of speech synthesis with the naturalness of pre-recorded audio: a comparison of two approaches to phrase-splicing TTS

Wael Hamza and John F. Pitrelli
IBM T.J. Watson Research Center, P.O. Box 218, Yorktown Heights, NY 10598 U.S.A.
{hamzaw,pitrelli}@us.ibm.com

Abstract

Many applications of TTS incorporate both unpredictable words, which require the flexibility of TTS, and static phrases, for which the quality of recorded speech is unmatched by TTS. “Phrase-splicing” TTS attempts to provide the optimal combination of the two, by customizing concatenative TTS to such applications by incorporating application-specific recordings at the word or phrase level while resorting to smaller-unit synthesis to fill the gaps not covered by those recordings. In the past, we have achieved this by using a word-level search on the application-specific recordings followed by a general-purpose TTS search, in our case using sub-phonetic units, to fill the gaps. However, recent trends toward larger-unit roles in general-purpose TTS suggest a single-search approach for phrase splicing. A listening test shows that we achieve at least as high quality with the new one-search algorithm as with two-search.

1. Introduction

While today’s state-of-the-art concatenative text-to-speech (TTS) synthesis systems [1] [2] [3] provide great improvements in intelligibility and naturalness compared to prior technology, they still fall far short of the naturalness of human speakers. As a result, deployed systems typically use pre-recorded human utterances whenever possible, forgoing the flexibility of TTS. However, many applications require frequent use of a limited set of phrases intermingled in utterances with some open-vocabulary material. For example, an application providing driving directions might say “Go straight for 3.5 miles; then turn right onto Hartwell Street.” In this case, most of the sentence is sufficiently frequently-occurring that it is reasonable to pre-record it, while the street name would probably need to come from TTS, and the number would probably come from another utterance, because the combination of all numbers in all such phrases is likely prohibitively large to record.

While we could use general-purpose TTS for text like this, we would like to exploit the naturalness of the pre-recorded speech which is available for the majority of this utterance, and sacrifice this quality for the flexibility of TTS only where necessary, such as for the street name. For this reason we have developed a “phrase-splicing” mode for our TTS engine, in which naturally-spoken phrases can be incorporated wholesale from the corpus when available, and standard concatenative TTS is spliced in as needed for words which were not pre-recorded. In this way, we enjoy the best combination of the naturalness of pre-recorded speech and the flexibility of concatenative TTS, as has been discussed in the literature [4].

Our prior implementation of phrase splicing [5] involves adding a word-level “pre-search” to our standard concatenative engine. The pre-search identifies word sequences in common between the text to be synthesized and the text represented in the corpus, to identify the longest sequences of natural speech to be incorporated wholesale. Then the standard concatenative search is applied to fill in any missing words, and standard TTS signal processing is applied to smooth the splices.

The purpose of the present study is to evaluate an experimental new approach to pursue phrase splicing while dispensing with the overhead of the pre-search, by exploiting newer algorithms in our concatenative engine. Specifically, “one-search phrase splicing” introduced here employs a merger of the application-specific corpus and the general-purpose corpus. It achieves phrase splicing within the single concatenative TTS search by using techniques which bias the search toward including contiguous passages from the corpus, whether application-specific or general-purpose, and by reducing signal processing within such passages in order to preserve the natural characteristics of the original recordings. The aim of this study is to determine whether this new approach achieves the quality of the two-search method for phrase-spliced TTS.

2. The IBM concatenative speech synthesis system

The IBM text-to-speech system consists of three major components: a front end which normalizes text and determines pronunciations, a prosody module which generates \( f_0 \) and duration targets, and a back end which searches a speech corpus to select segments to minimize a cost function, concatenates them, and optionally processes the resulting synthetic speech signal. The system was described in detail previously [6] [2], and is summarized briefly here.

During the process of building the speech corpus, we record a training script which provides several hours of speech. The recordings are made at a 22-kHz sampling rate. The training script is chosen for phonetic coverage as well as real-world applicability. In the experiments reported here, we use a software-only approach to determine moments of glottal closure in regions of voiced speech; these are used by the segment concatenation/interpolation module in synthesis.

Speech is encoded into 12-dimensional mel-frequency cepstral coefficients plus log energy, and the first and second derivatives of these parameters. Speech is then time-aligned to a sequence of sub-phonetic units corresponding to the states in a speaker-independent hidden Markov model (HMM). The resulting alignment is used for training speaker-dependent decision-trees [7] state-clustered HMMs. These HMMs are used to obtain the final alignment of the speech, which is used to build acoustic decision trees for each unclustered HMM state. The speech segments belonging to each leaf in the resulting trees
are kept together with information about their energy, \( f_0 \), duration and endpoint spectra. Spectral-continuity-cost decision trees are built using the final alignment; these trees are used to determine the spectral continuity cost between speech segments during the run-time segment search in synthesis [8].

During synthesis, text processing, text-to-phone conversion, and phrase boundary placement are performed by an independent, rule-based front end. Statistical decision trees are used for predicting both \( f_0 \) (three values per sonorant phone) and duration (one value per phone) targets, using a set of features representing phonetic and syntactic context. For duration modeling, the feature set is augmented by some of the same features determined for two preceding and two following phones. The results of front-end processing and prosodic-target generation are passed to the back end, which generates synthetic speech by selecting units to minimize a cost function consisting of a manually-tuned weighting of spectral and \( f_0 \) discontinuity costs, costs for units’ deviation from \( f_0 \) and duration targets, and a bias favoring units which were contiguous in the original corpus. The system uses sub-phonetic segments, as described above, as its basic synthesis units. Decision trees for unit definition based on phonetic context are used in conjunction with a dynamic programming segment search over the speech segments in each leaf of the synthesis leaf sequence, to obtain the segment sequence that is closest to the prosodic targets while minimizing the spectral continuity cost between adjacent segments. The optimal segment sequence is passed to the concatenation/modification module, which concatenates the segments after modifying their \( f_0 \) to match a smoothed version of the target \( f_0 \) contour. However, experiments were performed with duration modification off, meaning duration targets were used at search time, but the chosen segments did not have their durations modified to match the targets.

Two relatively new features in our system have been motivated by the desire to reduce splice count and to preserve the characteristics of contiguous natural speech when it is available at synthesis time. These are:

- Adding to the search space some segments other than those prescribed by the acoustic trees, in cases in which alternate pronunciations of the same word would cause different acoustic-tree leaves to apply to the same position in the word [9]. This is essential for picking up contiguous passages from the corpus, in the face of pronunciation variation. For example, in building the corpus, we must allow for variants such as /\vowel/ vs. /\vowel/ for the stressed vowel in “tomato”. But the front end will predict only one of these two pronunciations, and if it differs from the one chosen by the corpus speaker, without these alternates we could never retrieve the entire original recording of the word even though it exists in the corpus. Such circumstances are considerably more common than such well-known multiple-pronunciation words such as “tomato” because acoustic models to label the corpus need to allow for such alternates as varied-lexical-stress-level versions of vowels, syllabic vs. consonantal liquids and nasals, etc.
- Suppressing most signal processing, such as smoothing \( f_0 \) or adjusting it to fit targets, when within contiguous passages, so as to preserve the natural characteristics of such speech, restricting most processing to be applied only in regions near splices [2]. A parameter controls how much speech on each end of a contiguous passage is processed normally, in order to smooth the transition from such a passage to another segment of speech. Typically this parameter is set to be approximately one phone.

These features are key to preserving natural-speech characteristics when made possible by a match between the corpus text and the input text, and are thus key to improved phrase splicing. An interactive demo of the engine including these features is available at http://www.research.ibm.com/tts.

We typically “pre-select” concatenative corpora so as to reduce the memory footprint and search computation by eliminating less-frequently-used segments. A large text, typically around 100,000 sentences representing general usage of the language, is processed by the synthesizer using the full corpus, and statistics of segment usage are compiled. Then, within each leaf in the acoustic tree, the most-used segments are retained as described previously [10], and the remainder are discarded, typically yielding a substantial savings in memory usage and computation at a relatively small cost in quality.

3. Approaches to phrase splicing

As mentioned above, the goal of phrase splicing is to customize TTS to an application, taking advantage of any predictable phrases in the application by pre-recording them, then making them available to the synthesis algorithm to incorporate wholesale into output sentences. Here, we provide details on the two-search algorithm we have used for phrase splicing in the past, and the new one-search algorithm we propose.

3.1. Two-search

As described previously [5], our two-search phrase-splicing algorithm employs two separate corpora. In addition to the general-purpose concatenative speech corpus, we add another corpus, consisting of application-specific material catalogued at the word level and not reduced by pre-selection. At run time, a pre-search compares the sequence of words to be synthesized against the sequences of words in the application-specific corpus, seeking the longest sequences of words which cover the maximum portion of the text to be synthesized. Then the standard concatenative TTS search is performed, with its data modified as follows. When word occurrences have been found by the pre-search, the utterances containing them are made available to the standard search, and this search’s \( f_0 \) and duration targets for those words’ segments are overwritten by the actual values within the found word occurrences. In this way, we ensure that any words chosen by the pre-search remain chosen by the standard search. The standard search proceeds to select segments for any words unmatched by the pre-search in the normal way, thereby completing the synthesized utterance.

Two changes have been introduced to this phrase-splicing algorithm since it was described previously [5]. One is that for a word to be considered a match between the requested text and the corpus, the presence or absence of silence preceding and following the word in the corpus must match that predicted by the front end for the text to be synthesized. The other is that any punctuation immediately following the word in the corpus – comma, question mark, exclamation mark or period – must match the punctuation of the text to be synthesized.

3.2. One-search

As mentioned above, two recent improvements to our standard concatenative TTS algorithms have been motivated by the desire to reduce splice count and to preserve the characteristics
of contiguous natural speech when it is available at synthesis time. While these improvements have been applied successfully to general-purpose TTS, as well as in the above two-search phrase-splicing algorithm, the question motivating the current study is whether they are sufficient to achieve the goals of phrase-splicing synthesis without requiring the pre-search. Accordingly, for “one-search phrase splicing”, while building the corpus, we again exempt application-specific speech from being eliminated via pre-selection, and we combine this corpus with the standard concatenative corpus. Then, at run time, we increase the cost function’s bias toward contiguous passages, turn on the alternates feature, and enable suppression of signal processing within contiguous passages, as described above.

4. Evaluation

We evaluate these two techniques for phrase-splicing TTS using a scenario motivated by an in-car application, specifically, a system to provide driving directions and traffic reports. We compare the two algorithms against a baseline of standard concatenative TTS which has not been customized to the application, and perform a listening test, as described herein.

4.1. Corpora

Approximately eight hours of speech was collected from a native female professional speaker of American English reading a script which contains a general-purpose portion and an application-customized portion as follows. The general-purpose corpus consists of about seven hours of speech, whose sentences were chosen for phonetic coverage as well as real-world applications other than the in-car applications targeted here. The application-specific corpus consists of the remaining approximately one hour of speech and contains phrases and sentences particularly useful for driving directions and traffic reports.

4.2. Experimental systems

We compare three systems, matched for the overall quantity of speech available to the system:

1. Pure concatenative / no customization to application: The general-purpose corpus is pre-selected down to approximately four hours and is used as the sole corpus for synthesis.

2. Two-search phrase splicing: The general-purpose corpus is pre-selected down to approximately three hours of speech, and is used in conjunction with the entire one-hour application-specific corpus.

3. One-search phrase splicing: The same three hours of general-purpose speech and one hour of application-specific speech as in two-search phrase splicing are combined into a single corpus for the single search.

4.3. Listening test

Typically in applications such as the one studied here, some utterances are “more in-domain” than others. Accordingly, we used three sets of sentences in our listening test:

1. In-script: sentences which are in the application domain and composed completely of words and phrases within the application-specific corpus. However, no sentence appears verbatim as a single contiguous passage in the corpus. An example is “Take I-45 south toward Houston.”, in which alternating italic and non-italic passages represent phrases covered by different utterances in the corpus, implying the minimum splicing which will be required at synthesis time by the phrase-splicing systems.

2. In-domain, partially in-script: sentences which are in the application domain but include some words which are not in any part of the script, e.g., less-common names of roads and towns. An example is “Brimfield Street will be closed Sunday in Sturbridge from ten A.M. to nine P.M. for the festival.” Here, italics represent words not in the corpus; splices will be required within these words, as well as between the in-script passages comprising the remainder of the sentence.

3. Out-of-domain: sentences on other topics which have little or no text in common with the application script. These include sentences on various topics such as news, financial transactions, sports, and weather.

Eleven sentences were created for each of these three sets, and synthesized using each of the three systems, for a total of 99 stimuli, an approximately 25-minute listening test. The order of stimuli was randomized 15 times, subject to the constraint that no two of the three stimuli for each sentence could appear consecutively in any ordering. Thirty native adult speakers of American English, 15 male and 15 female, served as listeners; each stimulus order was used once for one listener and in reverse for another. In this way, ordering effects are controlled for, and we have a within-subject design for synthesis algorithm and stimulus text. Before beginning, listeners were presented two other stimuli as practice examples chosen to illustrate the task. For each stimulus, listeners were asked to rate the quality of the speech from 1, poor, to 5, excellent. Listeners were allowed to repeat playing each sentence as many times as they chose.

4.4. Results

Results are summarized in Figure 1. As can be seen, one-search phrase-splicing at least matches the performance of two-search, indicating achievement of the primary objective of this experiment. In fact, one-search appears to out-perform two-search on in-script material, though this distinction of 0.06 mean-opinion-score (MOS) point is statistically insignificant, and the high 4.10 and 4.16 MOS scores reflect the high degree of preservation of natural speech characteristics enabled by both algorithms when the text to synthesize is well-represented in the corpus. On the other two categories of text, reflecting less overlap with the application script, the performance difference grows somewhat, with one-search providing results superior to two-search by 0.18 and 0.15 MOS point, differences whose significance levels fall between 0.01 and 0.05. As expected, either phrase-splicing method significantly and substantially outperforms uncustomized TTS on either category of in-domain text. The performance difference degrades gracefully on out-of-domain material, as one would expect, with the customization never becoming a liability.

5. Discussion and Conclusions

A standard sub-phonetic concatenative TTS search, with algorithm upgrades designed to improve the use of contiguous passages regardless of customization, can be applied to achieve phrase-splicing customized TTS at a level of quality which
by incorporating contiguous phrases, and the flexibility of sub-phonetic concatenative synthesis and smoothing, are gracefully weighed against each other simultaneously by a single search to find the best operating point for application-customized TTS.

In the future, we envision implementing the search in a more flexible framework such as weighted finite-state transducers [11]. This framework allows exploring more ideas, and it relaxes research-limiting constraints imposed by our current standard Viterbi search.

6. Acknowledgments

We are grateful to members of the IBM speech synthesis team, including Raimo Bakis, Rob Donovan, Ellen Eide and Michael Picheny, for their assistance, advice, and work on the synthesizer, and to Allen Delmar for assistance with the listening tests.

7. References