Blizzard Entry:
Integrated Voice Building and Synthesis for Unit-Selection TTS

Christian Weiss, Sergio Paulo, Luis Figueira, Luis C. Oliveira

Spoken Language Systems Laboratory
INESC-ID, Lisbon, Portugal
(christian.weiss, spaulo, luisf, lco)@inesc-id.pt

Abstract

In this paper we describe our system used for the 2007 Blizzard Challenge TTS evaluation task. Following the rules we were building three voices from the given speech database where a first voice was created from the full data a second voice was build from the ARCTIC subset data and a third voice from a self-defined subset. The self defined subset was choosen by a text selection algorithm that selected sentences out of the full speech data recordings. Although the Blizzard team provides an already labeled corpus we were buling all the voices from scratch using our own segmentation and pre-processing tools. As result we show in this paper our segmentaion algorithm, the text-selection algorithm for choosing an optimal subset from the full speech data corpus and the voice building and synthesis system itself. Since a TTS system can be divided in an offline and online process we describe the offline pre-processing, the modules we use to prepare the data and describe the online synthesis runtime process how the acoustic soundfiles are generated.

1. Introduction

Following the rules of the Blizzard Challenge the teams were obliged to build three TTS voices from a common speech database and synthesize a specific number of previously unknown text. The provided common speech data exists of about 8 hours recorded speech from different domains. For each of the three voices which should be used for runtime synthesis, a different amount and set of data had to be used. First building a voice and synthesizing text from the full set of recordings, second a voice and synthesis from only the ARCTIC subset and third a voice which had to be built from an subset of the full database where the size is comparable to the ARCTIC subset size but the text should be selected with a self-deployed text selection algorithm. The previously unknown given text had to be synthesized with the 3 voices. The resulting soundfiles were then evaluated.

Voice building is a non-trivial task and as we know from the widely used Festival[1] software, knowledge of speech processing algorithms and expert knowledge as well as well prepared data is an essential prerequisite to setup a successful TTS system that is capable to synthesize text. As all of these prerequisites are an expensive resource the intention of the provided integrated voice building framework was to provide an integrated voice building software that can be used by non-speech experts with less well prepared data to build a TTS voice and synthesize text in an fast and uncomplicated way. The software is designed to automatize as many steps as possible but gives the freedom to more experienced users to manually interfer to manipulate the process. The software is released as a prototype version, free for non-commercial use and can be downloaded at http://www.javiss.org.

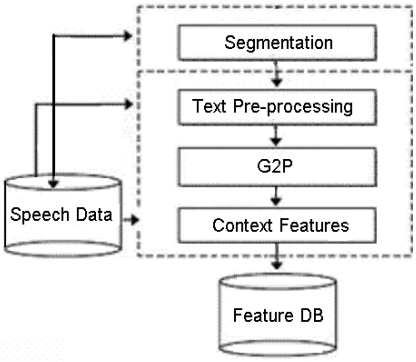


Figure 1: Schematic overview voice-building modules

In the next sections we describe the software for building the 3 voices and show how we used this software for the Blizzard 2007 challenge TTS evaluation. This paper is organized as follows: section 2 gives an overview of the integrated voice building software. Section 3 gives an overview on the segmentation of the audio files used to generate our label format. Section 4 describes the text selection algorithm used to build Voice C where Voice C represents a subset of the full speech data corpus. Section 5 describes the log-linear training algorithm used to train models for the NLP-pipeline as well as for the prosodic features duration and F0. Section 6 gives an overview on the runtime system and section 7 gives a conclusion as well as still to solve problems of the system.

2. Integrated Voice-Building

The system follows the common setup for TTS systems where a audio-file segmentation and labeling is followed by a text-preprocessing module, a prosodic feature prediction module and an acoustic synthesis module. The modules are plugged together interdependently to serve as an integrated voice building software which is capable to generate speech signals from given input text. The interchange between the modules uses an XML-based format. Figure 1 shows an schematic overview of the voice building modules. As we can see in the Figure 1 segmentation is a seperate part of the voice building itself since the input for the software are the audio-files, the according text-files and the according label-files produced by the segmantation module. As we are using our own label format as

well as we wanted to build the Blizzard voices from scratch we decided to use our own segmentation. Section 3 describes the segmentation and labeling of the given audio-files in detail. After gathering all the audio, label and text files together the voice building process can start. The software takes the input files and does a text pre-processing where special- and non-text characters are resolved. This is often referred as text-normalisation, but in our case a lexicon lookup of abbreviations and special characters is used and is therefore extensible. Second a grapheme-phoneme conversion transcribes the given input text into the phonetic symbol representation, and third a context feature module gathers all context related features. We define context features as everything that is related to the quantitative, linguistic-phonological, and prosodic context of a segement and its surrounding segments. The context dimension is freely chooseable but in our case we used only the left-right context of the current segment. Using this information we build a segment feature-description database including all the information plus the sentence id and the start and end time of the segments as well as the spectral representation of the first and last frame of the segment. One can say we build a context-structure and deploy a context-structure matching, segment selection algorithm. Please see section 6 for a detailed description of the segment selection for acoustic waveform generation. The next section gives an overview of the phonetic segmentation.

3. Phonetic Segmentation

Blizzard databases were segmented using speaker-adapted context-independent Hidden Markov Models (HMM). Distinct models were trained for each database. The final database segmentation was performed by a WFST-based segmenter, allowing multiple phonetic sequences [2], using the database-specific acoustic models.

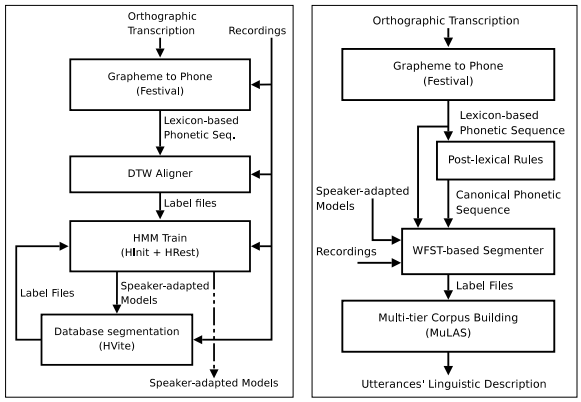
3.1. Speaker-adaptation of Acoustic Models

The speaker-adaptation procedure makes use of various tools from the Hidden Markov Toolkit (HTK) [3]. Those are HCopy, HInit, HRest and HVite. The HMMs have the usual left-right topology, with 8 mixtures per state. Each 16 kHz, 16 bit/sample waveform was parameterized into vectors of 39 coefficients (12 MFCC, energy, and its deltas and delta-deltas), extracted every $5\ ms$, using $20\ ms$ wide Hamming windows (HCopy).

In the absence of acoustic models for the *radio* phone set, which is the phone set used by American English voices in the Festival Speech Synthesis System [1], the databases were initially aligned by a DTW-based aligner using multiple acoustic features [4]. The phonetic sequences together with the reference synthetic signals used by the alignment procedure were generated by means of the Festival's *kal_diphone* voice, which is an American English voice based on diphone concatenation. Then, the HMMs were initialized (HInit) and re-estimated by applying the Baum-Welch algorithm (HRest). After model re-estimation a forced Viterbi alignment was performed (HVite) in order to generate training data for the next training iteration. The train+alignment cycle was repeated three times for each individual database, as described in Fig. 2(a).

3.2. Corpora Segmentation

The final step of the prompt segmentation procedure is depicted in Fig. 2(b). A WFST-based segmentation tool is provided with the speaker-adapted models, the feature vectors for each recorded prompt and two different phonetic sequences:



(a) Speaker adaptation procedure.

(b) Final database segmentation.

Figure 2: Model training and prompt segmentation schematics.

- Lexicon-based phonetic sequence, obtained by concatenating the pronunciations produced by the Grapheme-to-Phone tools while words are taken as isolated;
- Canonical phonetic sequence, obtained after applying a set of post-lexical phonological rules to the Lexicon-based phonetic sequence;

The results of the phonetic segmentation were then propagated to all other linguistic levels of the utterance description by using the *MuLAS-based* [5] synchronization method.

4. Sentence Selection

Definition: Let a unit occurring within a specific context be known as a token.

Selecting a particular sentence subset capable of covering all the distinct tokens existing in a large sentence collection is an *NP-complete* problem, [6]. Although the optimal sentence set cannot usually be computed,[1] approximate solutions to the problem[2] can be obtained by means of *greedy* approaches, [7].

Several methods have been proposed for the solution of the sentence selection problem, [8, 9, 10]. A large majority of them builds on a *greedy* algorithm.

4.1. Sentence Selection Method

We also developed a greedy-based sentence selection approach. Our method consists in a multi-level token search. Our aim is to simultaneously cover diphones, triphones and syllables, as we force our primary target level (diphone level) to be covered faster.

4.1.1. Level-specific contextual features

We used the contextual features described in Table 1 to define a set of tokens to be covered at each level. Since we still do not have accurate text analysis tools for English, we decided to use the linguistic data available at the Festival utterance descriptions provided together with the recordings.

[1] Given the computational complexity of this sort of problems.

[2] Much less expensive in terms of computational effort.

Level	Feature	Definition
Diphone	syl_boundary	Binary feature accounting for the presence of a syllable boundary between the comprising phones (0 or 1).
Diphone	word_boundary	Binary feature accounting for the presence of a word boundary between the comprising phones (0 or 1).
Diphone	syl_stress1	Binary feature that is set (value = 1) when the first phone is a stressed vowel, and unset (value = 0) otherwise.
Diphone	syl_stress2	Binary feature that is set (value = 1) when the last phone is a stressed vowel, and unset (value = 0) otherwise.
Syllable	accented	Binary feature that is set (value = 1) when one or more intonation events are associated to the syllable, and unset otherwise (value = 0).
Syllable	syl_break	Level of disjuncture between the current syllable and the following one, if any, (0, 1, 2, 3 or 4).
Syllable	*ToBI* _accent	*ToBI* accent related to the syllable (!H*, H*, L+H* or NONE).
Syllable	*ToBI* _endtone	*ToBI* end tone related to the syllable (L-L%, L-H%, H-H% or NONE).
Triphone	syl_stress1	Binary feature that is set (value = 1) when the first phone is a stressed vowel, and unset (value = 0) otherwise.
Triphone	syl_stress2	Binary feature that is set (value = 1) when the intermediate phone is a stressed vowel, and unset (value = 0) otherwise.
Triphone	syl_stress3	Binary feature that is set (value = 1) when the last phone is a stressed vowel, and unset (value = 0) otherwise.

Table 1: Definition of level-specific features used to define the to-be-covered tokens in each level.

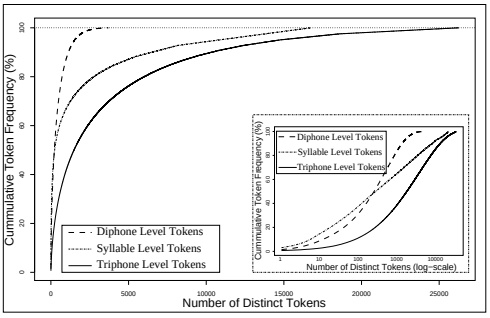


Figure 3: Accumulated frequencies of the level-specific tokens within the *Blizzard* corpora.

Level	#Tokens		#Token Occurrence			
	Overall	Distinct	Avg	Med	Max	Min
Diphone	312,188	3,700	84	16	4549	1
Syllable	118,815	16,737	7	1	3614	1
Triphone	310,045	26,268	12	3	1976	1

Table 2: Statistics on level-specific token distribution within the *Blizzard* corpora, where Avg, Med, Max and Min stand for average, median, maximum and minimum values, respectively.

4.1.2. Data

Accumulated frequencies of diphone-, syllable- and triphone-level tokens are depicted in Fig. 3 (the tokens were sorted by their relative frequency). Those graphics together with Table 2 show that the token's relative frequencies follow an uneven distribution. For instance, more than 50% of the syllable-level tokens occur only once in the corpora.

4.1.3. Algorithm

We use the *greedy* algorithm to iteratively select the sentence with the highest score. Sentences are scored according to (1), where j is a sentence to be scored, i accounts for the token level, *diphone, syllable or triphone*, and c_i is the level-dependent multiplying factor (LMF).

$$Score(j) = \sum_{i=1}^{3} c_i \cdot \overline{S_{i,j}} \qquad (1)$$

Given a sentence, say j, and a level, say i, an $S_{i,j}$ value is computed for every i-level token of sentence j according to equation (2).

$$S_{i,j}(T) = \begin{cases} \frac{2}{1+\varphi_i(T)} & if\ T\ is\ not\ covered\ yet; \\ 0 & otherwise; \end{cases} \qquad (2)$$

Due to the uneven nature of the token frequency distribution, we do not equally score all the *to-be-covered* tokens, as suggested in [10]. Therefore, $\varphi_i(T)$ is the ratio between the frequency that a token T of level i occurs within the corpora and that frequency value averaged over all tokens of that level. The $\varphi_i(T)$ value depends on that ratio, rather than on the absolute occurring frequency, in an attempt to balance the relative impact of the scores produced at the different levels on the overall sentence score.

4.1.4. Experiments

We used three distinct sets of LMFs in order to evaluate its impact on the token coverage at the multiple levels. Since the sentence subset allowed to be used in this application (*voice C*) is very size-limited, we decided to ensure a good diphone-level token coverage at the expense of all other levels. Therefore, the LMFs c_i of (1) take values of 10, 1 and 1, for diphones, syllables, and triphones, respectively. The results of this experiment

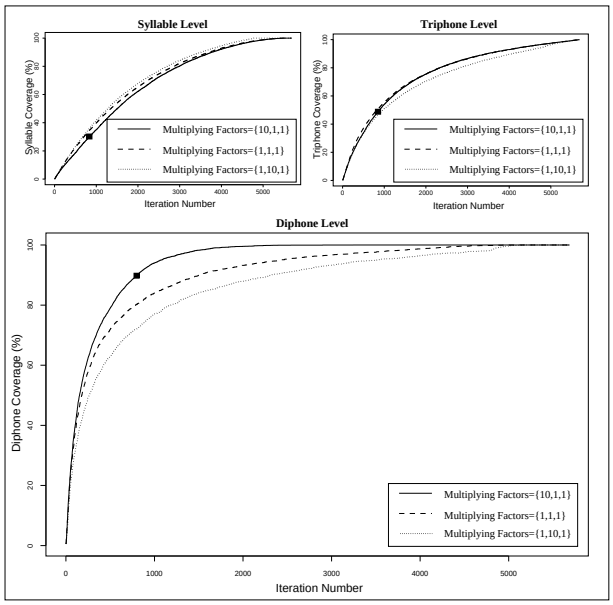


Figure 4: Token coverage for three distinct sets of multiplying factors, $\{c_1, c_2, c_3\} = \{c_{diphone}, c_{syllable}, c_{triphone}\}$.

(solid line) and another two experiments, for which we used LMFs of $\{1, 1, 1\}$ (dashed line) and $\{1, 10, 1\}$ (dotted line), are depicted in Fig. 4. Black squares in that figure are used to mark the token coverage of the sentence subset that gave rise to the unit inventory of *voice C*. The coverage values are 90%, 30% and 48% for diphones, syllables and triphones, respectively.

5. Conditional Log-linear Models

For Grapheme-Phoneme conversion, Part-of-Speech Tagging, syllable boundary detection, as well as for duration and F0 prediction we applied conditional log-linear models also known as Maximum-Entropy models [11]. The conditional log-linear model framework is a well known approach for ambiguities resolution in natural language processing [12] where many problems can be reformulated as a classification problem. The task of such a reformulation is to include a context and to predict a correct class. The objective is to estimate a function, $X \rightarrow Y$ which predicts an object $x \in X$ to its class $y \in Y$. Y represents the predefined classes for either each task of our prediction problem. In the field of stress prediction we are dealing with a binary classification where the class is true for stressed syllables and false for non-stressed. The same binary classification task has to be solved in the domain of syllabification where we have a syllable boundary or not. X consists of quantitative and phonological features where we include the context and the resulting input for the classification. The classifier can be seen as a conditional probability model in the sense of $C(x) = argmax_y p(y|x)$ where x is the object to be classified and y is the class. Including the context we get a more complex classifier $C(x_1, x_2, x_3, ..., x_n) = argmax_{y_{1,...,y_n}} \prod_{i=1} p(y_1|x_1, ..., x_n, y_1, ..., y_{i-1})$ where $x_1, ..., x_n, y_1, ..., y_{i-1}$ is the context at the i^{th} decision and y_t is the outcome.

We use this model in all our dynamic feature prediction tasks during the offline voice building process as well as during synthesis runtime. The features used for training and building the context-structure are listed in Table 3.

Unit	Feature
word	preceeding, succeeding
	sentence type
	distance left/right in sentence
	part-of-speech
	duration, log. duration
	avg. F0, log. F0
	first/last frame MFCC
syllable	preceeding, succeeding syllable
	distance left/right in sentence/word
	stress
	duration, log. duration
	avg. F0, log. F0
	first/last frame MFCC
phone	distance left/right word, syllable
	duration, log. duration
	avg. F, log. F00
	first/last frame MFCC

Table 3: Features used for log-linear model training

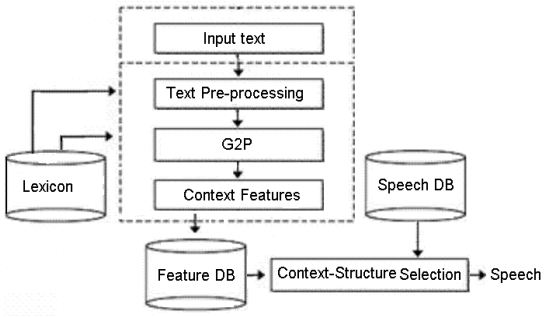


Figure 5: Schematic overview of the runtime synthesis system

6. Runtime System

The runtime synthesis system processes the given input text and text-normalizes the input where a lexicon based lookup for abbriviations, and special characters is used. The input text is converted into a phonetic represenation by the log-linear model. This model was trained using the public available CMU 0.6 dictionary. Once the input text is phonetically transcribed a context feature prediction as well as feature calculation is performed where quantitative features as well as sentence type can be extracted from the text and part-of-speech, stress, duration and F0 is predicted using the previous trained models. In the case of duration and F0 the models are trained using only values occuring in the existing speech database. One can say this approaoch is a speaker dependent duration and F0 model. Once all the context features are gathered a context-structure vector represents each of the identified speech segments. As we are using a top-down unit-selection algorithm the occuring speech segments could be: words, syllables and phonemes. Using this context structure the acoustic module performs the selection and speech signal generation. Figure 5 gives an overview of the speech generation runtime system.

6.1. Acoustic Module

The acoustic synthesis module follows the variable-size unit selection algorithm. We apply a pre-selection strategy while the algorithm tries to find a segment that matches the predefined target context-structure in a left-right context. If this does not result in any found segment we simplify the structure matching but keep the left-right context. When no segment is found on the word-level, the algorithms searches for syllable segments and in a last alternative a phoneme-level segment selection is performed. Using a predefined structure matching for segment selection we save computational resources in target and join-cost distance calculation while the search space is reduced. No exact measures are done here. The segment distance calculation is done by minimizing the distance of the selected segment and the target segment. Here a kind of normalization is deployed since the values are real integer values and e.g. the distance in the log-duration is not relative to the quantitative feature distance. This normalization is done by $x = \frac{x^2}{1+x^2}$. The join cost calculation is done by a Euclidian distance measure between the successive frames MFCCs of the segments.

7. Conclusion

The Blizzard evaluation gave us an first impression of the system and some critical problems are still to improve and some had been improved in the meantime. This is an ongoing process. The Blizzard Challenge showed that the selection algorithm had a bug where some units where not choosen appropriatetly. This led in the evaluation of the intelligibility to a less good result. Using no signal manipulation after the speech signal generation process doesn't result in a natural prosody. Here a post-processing with a common prosody model could lead to a better prosody performance. In terms of speed we could improve the selection process by using our context-structure matching to our previous search approach, but an comparison with existing systems has to be done and exact measures should show the performance. All in all the software is useable to build a voice in a very short time with less speech expert knowledge and could be an alternative to existing software used in research and universities.

8. References

[1] A. W. Black, P. Taylor, and R. Caley, *The Festival Speech Synthesis -System documentation*, 2002.

[2] S. Paulo and L. C. Oliveira, "Generation of word alternative pronunciations using weighted finite state transducers," in *Interspeech 2005*, 2005.

[3] S. Young, G. Evermann, T. Hain, D. Kershaw, G. Moore, J. Odell, D. Ollason, D. Povey, V. Valtchev, and P. Woodland, *The HTK Book (for HTK Version 3.2.1)*, 2002, available at http://htk.eng.cam.ac.uk/docs/docs.shtml.

[4] S. Paulo and L. C. Oliveira, "Dtw-based phonetic alignment using multiple acoustic features," in *Interspeech 2003*, 2003.

[5] S. Paulo and L. C. Oliveira, "Mulas: A framework for automatically building multi-tier corpora," in *Interspeech 2007*, 2007.

[6] S. Arora and B. Barak, "Complexity theory: A modern approach," Princeton University, Tech. Rep., 2006, Available at: http://www.cs.princeton.edu/theory/complexity.

[7] D. S. Johnson, "Approximation algorithms for combinatorial problems," in *Fifth annual ACM symposium on Theory of Computing*, 1973.

[8] B. Bozkurt, O. Ozturk, and T. Dutoit, "Text design for tts speech corpus building using a modified greedy selection," in *Eurospeech 2003*, 2003.

[9] A. W. Black and K. A. Lenzo, "Optimal data selection for unit selection synthesis," in *4th ESCA Workshop on Speech Synthesis*, 2001.

[10] J. P. H. van Santen and A. L. Buchsbaum, "Methods for optimal text selection," in *Eurospeech 97*, 1997.

[11] A. Berger, S. A. DellaPietra, and V. J. DellaPietra, "A maximum-entropy approach to natural language processing," *Computational Linguistics*, vol. 22(1), 1996.

[12] A. Ratnarparki, *Maximum Entropy Models for Natural Language Ambiguity Resolution*. PhD Dissertation, University of Pennsylvenia, 1998.