A Text-to-Speech Platform for Variable Length Optimal Unit Searching Using Perceptual Cost Functions

Minkyu Lee, Daniel P. Lopresti, Joseph P. Olive

Bell Labs, Lucent Technologies,
600 Mountain Avenue, Murray Hill, NJ 07974, USA
{minkyul,dpl,jpo}@research.bell-labs.com

Abstract

In concatenative Text-to-Speech, the size of the speech corpus is closely related to synthetic speech quality. In this paper, we describe our work on a new corpus-based Bell Labs' TTS system. This encompasses large acoustic inventories with a rich set of annotations, models and data structures for representing and managing such inventories, and an optimal unit selection algorithm that accommodates a broad range of possible cost criteria. We also propose a new method for setting weights in the cost functions based on a perceptual preference test. Our results show that this approach can successfully predict human preference patterns. Synthetic speech using weights determined in this manner consistently demonstrates smoother transitions and higher voice quality than speech using manually set weights.

1. Introduction

In concatenative Text-to-Speech, there is a trade-off between corpus size and speech quality. Smaller corpus-based TTS systems, typically diphone-based, require more frequent unit concatenations, often resulting in speech quality degradation. For applications with limited resources (e.g. hand-held devices), TTS systems with a small set of diphone units are more appropriate. Research in this field is focused on better speech production models and signal processing methods to improve the quality of synthetic speech from the limited set of units.

On the other hand, TTS applications based on personal computers or larger computer systems can afford the large corpus-based approach, which have more potential for higher quality synthetic speech. In large corpus systems, units of any size, from diphone, triphone, or longer, can be used for synthesis. Longer units can better preserve the naturalness of the original speech. Moreover, the same unit can have multiple entries in the speech corpus, each entry recorded with different prosody and in different contexts. By selecting a unit whose prosody is already close to the target prosody, the burden of signal processing for prosody modification can be alleviated. Fewer concatenations and less signal processing enable large corpus TTS systems to produce smoother and more natural sounding synthetic speech.

Recent research issues in this area include speech corpus construction, appropriate unit-searching algorithms, etc. Nakajima and Hamada [1] suggested an automatic synthesis unit generation method using a statistical clustering technique. Allophones are determined and automatically generated from a labeled speech corpus. Hunt and Black [2] proposed a phoneme-based unit-searching algorithm, where the Viterbi searching algorithm is applied to select the best units. Two cost functions, target cost and concatenation cost, are defined. To determine the weights of sub-cost functions, two methods were proposed: weight space search and regression training. The algorithms find a set of weights which minimize the difference between the natural and synthetic speech. Donovan [3] constructed a large speech corpus based on decision-tree state-clustered hidden Markov models. Dynamic programming search is carried out to determine synthesis units, which will have prosody (duration, energy and pitch) such that the amount of quality degradation introduced by the signal processing module that follows (TD-PSOLA) is minimized. Breen and Jackson [4] dynamically generated a sequence of units based on a global cost. The costs are based on purely phonologically motivated criteria without reference to any acoustic features. The unit selection process makes no direct use of acoustic information.

In general, given the target specification from the front-end of the TTS system, a unit selection algorithm searches through the speech corpus to find a set of units that (1) connect well, (2) are from similar context, (3) are close to the target prosody specification, and (4) are as long as possible. However, the calculation of costs, especially determining the weights of the subcosts, is a very complicated issue.

In this paper, we describe our recent work on a new corpus-based Bell Labs' TTS system. Key features of this research platform include support for large acoustic inventories with a rich set of annotations, models and data structures for representing and managing such inventories, an optimal unit selection algorithm that accommodates a broad range of possible cost criteria, and an interactive tool to provide visualizations and encourage exploration.

We also propose a new method for unit-searching based on a perceptual preference test. The proposed algorithm is designed to find the weights in the cost functions in more systematic and meaningful way. The algorithm searches for a set of weights that can produce a ranking of renditions that is close to the perceptual test results. The downhill simplex method is used for the multi-dimensional search of the weights. A dissimilarity measure is proposed to evaluate the closeness of two rankings. In about 83 percent of the cases, the unit selection algorithm using the optimal set of weights chooses the same rendition that human listeners prefer. These results show that the proposed weight optimization algorithm can successfully predict the human preference pattern. Synthetic speech using the optimal weights consistently shows smoother transitions and higher voice quality than speech using manually determined weights.

In Section 2, we present an overview of the system. The optimal unit selection algorithm is described in Section 3. Our cost functions, which include acoustic and concatenation components, are explained in Section 4. The downhill simplex method is briefly introduced in Section 5. A dissimilarity measure is described in Section 6. Finally, we present simulation results in Section 7 and our concluding remarks in Section 8.

2. System Overview

The current system is a research prototype consisting of:

1. A flexible, unifying data structure, the *annotated string*.
2. Efficient representation of acoustic inventories and target utterances in terms of annotated strings.
3. Software routines for manipulating annotated strings to support building and maintaining acoustic inventories.
4. C language implementation of the Viterbi dynamic programming algorithm for performing optimal unit selection.
5. An interactive system for exploring the results of unit selection from a large corpus, including visualizations for the search graph and a wide range of useful statistics.

Facilitating the management of acoustic inventories is a primary goal of our work. Previously such inventories were assembled and carefully tuned by hand. The shift to very large corpora, though, as well as a desire to support quickly retargetable, application-specific synthesis systems precludes such a time-consuming, labor-intensive process. Rather, we have developed a way of automating the construction of new inventories once the basic corpus has been annotated.

This paradigm centers around the *annotated string* data structure, which encodes the phonetic identity of each unit as well as an open-ended set of features extracted from the speech samples. An example of an annotated string fragment is shown in Figure 1. This data structure is complete in the sense that it includes everything we need to perform optimal unit selection for synthesis. Acoustic inventories, then, are simply collections of annotated strings. We have created a programming environment, embedded in a general-purpose scripting language, that allows for the manipulation of such inventories programmatically. Functions provided include converting between various formats, assembling and filtering collections of annotated strings, and cutting strings into smaller units based on rules that may invoke regular expression pattern matching, acoustic criteria, or a combination of both.

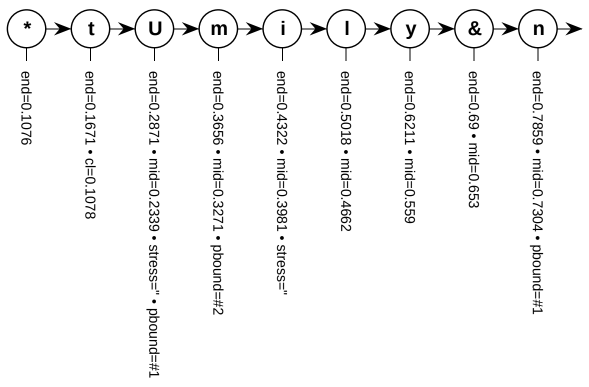
*
t
U
m
i
l
y
&
n
end=0.1076
end=0.1671 • cl=0.1078
end=0.2871 • mid=0.2339 • stress='' • pbound=#1
end=0.3656 • mid=0.3271 • pbound=#2
end=0.4322 • mid=0.3981 • stress=''
end=0.5018 • mid=0.4662
end=0.6211 • mid=0.559
end=0.69 • mid=0.653
end=0.7859 • mid=0.7304 • pbound=#1

Figure 1: An annotated string fragment.

As a result, it becomes possible to write simple programs that build and modify inventories, so that generating a new inventory now requires minutes of CPU time instead of days or weeks of manual effort. The output of this process is also designed to be compact and efficient; several hours of recorded speech yields an inventory requiring only a few megabytes of storage, and unit selection achieves real-time throughput. In this way, speech synthesis can be easily and inexpensively specialized to the application of interest.

Optimal unit selection is accomplished via Viterbi search. The problem is formulated as finding a minimum-cost path through a graph consisting of all possible ways of concatenating inventory units of any length to realize the target utterance. Since the inventories in question may be large, the graphs can easily consist of ten thousand or more nodes. While the current implementation is a research prototype and not yet optimized for speed, its performance is still quite reasonable, measuring on the order of seconds for an average input sentence. Beyond returning a minimum-cost synthesis path, the search code is also capable of outputting the complete computation graph, including all of the nodes and their associated costs, as well as all alternate optimal paths (often there is more than one). These can be explored by the user (e.g. a TTS researcher, or a field engineer designing an acoustic inventory for a specific application) using a graphical interactive system we have developed.

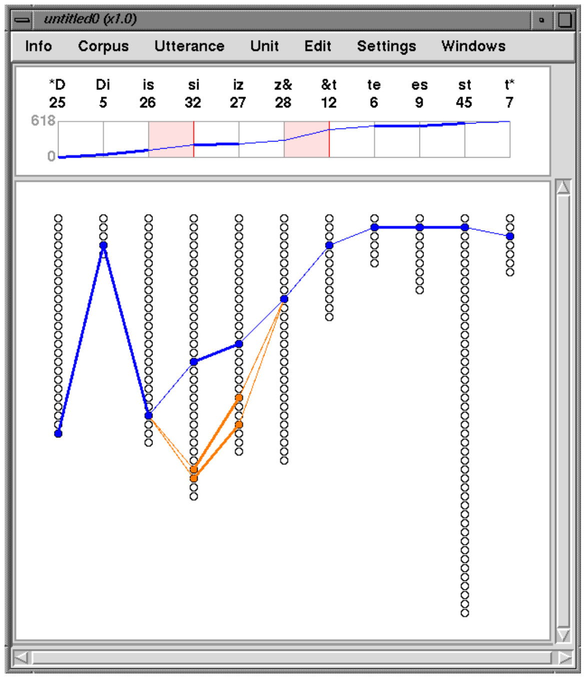

Figure 2: Screen snapshot of the graphical tool for visualizing TTS unit selection results.

Figure 2 shows a screen snapshot of our tool for viewing and manipulating synthesis results. The lower portion of the screen shows the search graph. Each node corresponds to a specific diphone from the current inventory (diphones are the shortest fundamental unit in our system, but units of any length may arise in general); as can be seen, there are 45 instances of the "st" diphone in the corpus in question (a small inventory was used to generate this particular example to keep the figure legible – the full-size corpus we normally use has 254 instances of "st"). The default optimal path is represented by the dark edges, with the thicker edges (e.g. "*Dis") representing contiguous units from the inventory (i.e. no concatenation is necessary). Two alternate optimal paths are shown by the lighter colored edges beneath the default path. The upper portion of the screen shows the target utterance and the accumulated cost proceeding from left to right. The tool itself is coded in Tcl/Tk, a well-known scripting language designed for building user interfaces.

3. Unit Selection

After the target phone sequence and its prosody are determined by the text analysis module, the entire speech corpus is searched to find the optimal units for synthesis. The basic strategy is to select the longest possible candidate at each step, provided its acoustic features and phonetic environment match well with those of the target. The former include mean F0, slope of F0, duration, and accentuation, while the latter refers to the types of the surrounding phones. An *acoustic cost* is calculated to evaluate the compatibility between the candidate unit and the target. If two units from different contexts need to be concatenated, the location of the best connection is determined by searching for a point where the spectral distance is minimal. The spectral distance at this point is used to assess a *concatenation cost* between the two units. Finally, using these cost functions and Viterbi search, an optimal sequence of units realizing the target utterance is selected.

Our implementation of the dynamic programming algorithm makes use of a number of sophisticated data structures so that large search graphs can be handled efficiently. This process runs as a server; inventories and target utterances are loaded dynamically. The two cost functions have access to all of the attributes encoded in the annotated string data structure, and incorporating new costs is straightforward. Figure 3 charts the computation times on an SGI O2 workstation (200 MHz MIPS R5000 CPU, 64 MB RAM) for a run that processed 1,104 utterances from the transcript of a recent U.S. Supreme Court case. The corpus consisted of 1,587 recorded sentences drawn from an unrelated source. Despite the fact that these search graphs can be quite large (averaging over 8,500 nodes) and neither the inventory nor the code is optimized yet, the average time for unit selection is less than 10 seconds.

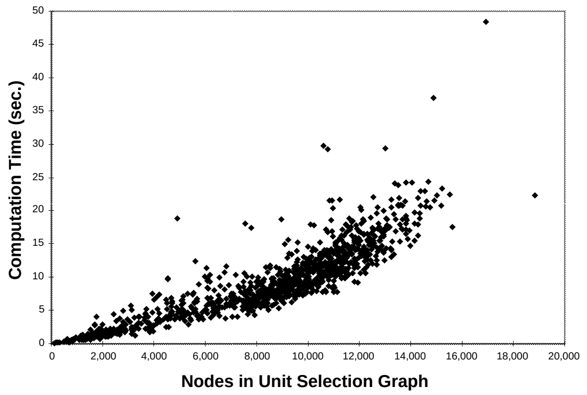

Figure 3: Computation time as a function of the number of nodes in the unit selection graph.

While listening tests are, of course, the most important subjective criterion, one simple quantitative measure of the effectiveness of unit selection is its ability to choose units of differing lengths (and, in particular, longer units where appropriate). The histogram in Figure 4 shows the number of times units of a given length were used in the synthesis experiment just described. Somewhat more than half of the units were longer than the minimum, 2, which corresponds to the basic diphone. Over 500 units were length 7 or longer, with the average being 2.9. Since the test utterances were derived from a different "genre" than the acoustic inventory, it seems likely that even better results could be obtained using a corpus targeted to a specific application of interest.

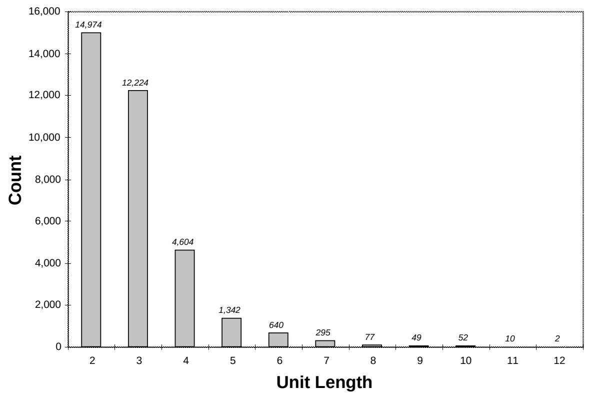

Figure 4: Histogram showing the number of times units of a given length were used (length 2 corresponds to diphones).

There is no doubt that proper unit selection has a critical impact on the quality of the resulting synthetic speech. Since the cost functions are weighted sums of sub-costs, the weights control the individual contributions of the various sub-costs to the total. They should also reflect the relative sensitivity of a feature to quality degradation when signal processing is applied to modify the feature. Hence, determining an appropriate set of weights for the cost functions is a fundamental problem.

4. Cost Functions

We denote the target specification provided by the front-end of a TTS system as T=$\{T_i \mid i = 1 \cdot \cdot P\}$ for an input sentence of P phonemes. Each target phoneme T_i is further annotated with M_i features, i.e., T_i=$\{t_{ij} \mid j = 1 \cdot \cdot M_i\}$. Candidates from the speech corpus are also annotated with M features, i.e., U_i=$\{u_{ij} \mid j = 1 \cdot \cdot M_i\}$.

The cost functions consist of an acoustic cost and a concatenation cost. The acoustic cost is a measure of the compatibility between a candidate unit and its target specification. A small acoustic cost means that the candidate has features that are close to the target specification. The acoustic cost between T_i and a candidate U_i can be calculated as

$$A_i = \sum_{k=1}^{M_i} w_k^a \cdot c_k^a(t_{ik}, u_{ik}). \quad (1)$$

The functions $c_k^a,\ k = 1 \cdots M_i$, are sub-cost functions for the acoustic feature elements, and each sub-cost function is weighted by w_k^a. The concatenation cost is a measure of the compatibility between two consecutive units. Similarly, a small concatenation cost will result in smooth concatenation. The connection cost between the $(i-1)^{th}$ unit and i^{th} unit is

$$C_i = \sum_{k=1}^{M_i} w_k^c \cdot c_k^c(u_{(i-1)k}, u_{ik}). \quad (2)$$

The functions $c_k^c,\ k = 1 \cdots M_i$, are sub-cost functions for the feature elements, which will be explained in Section 4.1. Each sub-cost function is weighted by w_k^c. Total cost is the sum of the acoustic cost and the connection cost:

$$C = \sum_{i=1}^{P} (A_i + C_i) \tag{3}$$

A set of candidate units that minimize the total cost for an input sentence can be found by Viterbi searching algorithm.

The weights, w_k^a and w_k^c, control the individual contribution of the sub-costs to the total cost. If the weight for pitch mismatch is larger than the other weights, the unit-searching algorithm will try to find a set of units whose pitch is close to the target pitch, neglecting other features. The weights also determine the relative sensitivity of a feature to the quality degradation when signal processing is applied to modify the feature. For example, if the ratio of the weight for pitch mismatch to the weight for stress mismatch is $\frac{1}{50}$, it implies that pitch modification by 50 Hz is equivalent (in terms of the quality degradation) to synthesizing a stressed vowel using a non-stressed unit.

Unfortunately, determining the weights is not a simple task. The proposed algorithm is designed to find the weights in more systematic and meaningful way.

4.1. Sub-cost functions

For the calculation of the sub-costs, c_k^a and $c_k^c,\ k = 1 \cdots M_i$, we use acoustic features as well as phonetic features of units. Acoustical features include pitch, duration, RMS energy, and the first three formants frequencies. Phonological features used are accent and phonemic context.

Most signal processing algorithms including PSOLA suffer degradation of the original speech quality as the pitch is modified. The larger the modification factor, the more severe the degradation which will be introduced into the synthetic speech. Concatenating two units that have different pitch frequencies generates a distortion due to the pitch mismatch [6]. Thus, a sub-cost function for pitch frequency is defined as the absolute pitch difference between units. The quality degradation due to duration and energy modification is relatively minor but are not negligible. Thus, we also define sub-cost functions for duration and energy in similar forms as that of pitch function. Spectral discontinuity between concatenating units is measured by a normalized Euclidean distance of the first three formants as:

$$c^c_{spectral}(\mathbf{F}^1, \mathbf{F}^2) = \sum_{i=1}^{3} \frac{|F_i^1 - F_i^2|}{w_i}, \tag{4}$$

where w_1=50, w_2=100, and w_3=180. The first three formant frequencies are estimated by the phone dependent formant estimation method described in [7].

Another sources of quality degradation are the mismatch of stress and phonetic environment. Using a stressed unit for synthesizing an unstressed vowel, or vice versa, does not produce very good speech quality. The acoustic cues for stressed/unstressed vowel is not yet fully understood. We use a binary cost function, i.e., when both the target and candidate are stressed or unstressed, the sub-cost value is zero. Otherwise, the value is set to one. Similarly, if the surrounding phonemes of the candidate are in the same class as the surrounding phonemes of the target, the sub-cost is zero. Otherwise, the sub-cost value is set to one. The phoneme classes are categorized by the place of articulation and manner of each phoneme.

An additional sub-cost function is the continuity sub-cost function. The weight for continuity sub-cost function is usually very big in order to encourage the algorithm to find longer units.

5. Weight Optimization – Downhill Simplex Method

A schematic diagram of the proposed system is shown in Figure 5. The goal is to optimize the sub-cost weights such that the unit-searching algorithm behaves as human listeners do.

The test words consists of a set of single-syllable words, W=$\{W_i \mid i = 1 \cdot \cdot N\}$, where N is the number of words. For each word W_i, multiple renditions R_i=$\{r_{ij} \mid j = 1 \cdot \cdot n_i\}$ are synthesized using various candidate units, which have different prosody and are from different context. The number of renditions n_i depends on the number of candidates in the speech corpus.

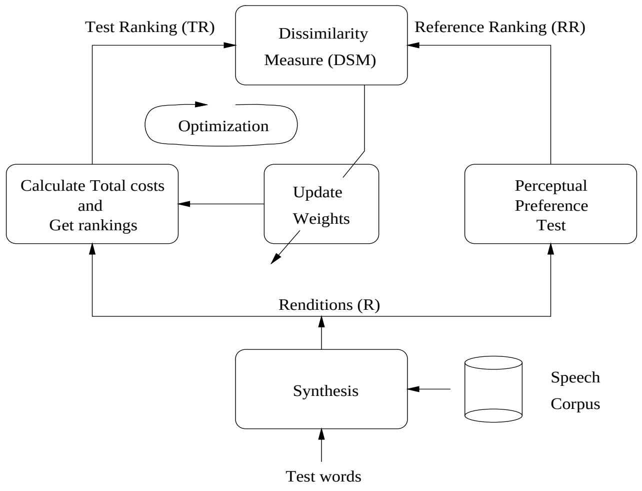

Figure 5: A schematic diagram of the proposed algorithm.

Listening tests are performed on the set of renditions. Listeners are asked to rank their preference of the renditions R_i for each word W_i. The ranking will be referred as the reference ranking. The total reference rankings RR=$\{RR_i \mid i = 1 \cdot \cdot N\}$ represent the perceptual preference of human listeners.

Meanwhile, using a set of heuristically chosen initial weights, the total cost of each of the n_i renditions in R_i are calculated (see Section 4). Then, the renditions are ranked according to the total costs. It is referred to as the test ranking. The total test rankings for all the words, TR=$\{TR_i \mid i = 1 \cdot \cdot N\}$ are delivered to the next block, where a dissimilarity measure of the two rankings, RR and TR, is calculated.

The dissimilarity measure is the average distance between the test and reference rankings for all N words (see Section 6). Then, the downhill simplex method [5] is used to adjust the weights to minimize the dissimilarity measure. When the dissimilarity measure is minimized, the weights produce a ranking that is similar to the reference ranking.

The optimization of weights is performed by the downhill simplex method. The downhill simplex method is a multidimensional minimization algorithm [5]. It is simple and provides quick solutions that minimize a function because it requires only the function evaluation, not the calculation of the derivatives.

A simplex is the geometric figure in N dimension, where N is the number of independent variables to be optimized. It consists of N+1 points, all the line segments connecting the points, and the polygonal faces. In 2-dimensions a simplex is a triangle. In 3-dimensions, it is a tetrahedron. Figure 6(a) shows a tetrahedron. The downhill simplex optimization algorithm evaluates the function at N+1 points in order to find a point where the function value is maximum (highest cost) and minimum (lowest point). Then, the algorithm moves the highest cost point to the opposite side of the simplex (Figure 6(b)), scale (Figure 6(c), or contract it (Figure 6(d)). It may do a multiple contraction (Figure 6(e)) along all dimensions toward the low point. The procedure repeats until a termination condition is satisfied.

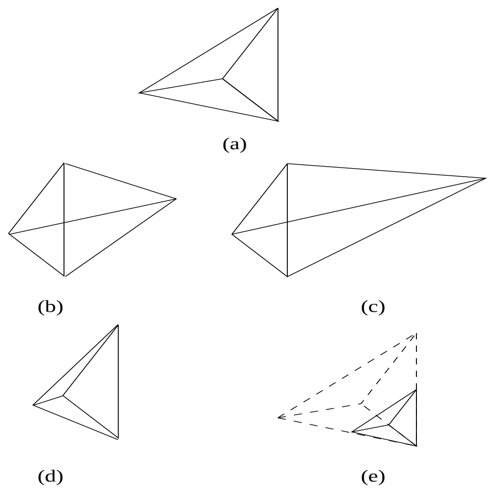

Figure 6: A simplex(a) and its possible variations for a step in the downhill simplex method. (b) reflection, (c) reflection and expansion, (d) contraction along one dimension from the high point, and (e) multiple contraction.

6. Dissimilarity Measure of Rankings

The downhill simplex method evaluates a function of N dimensional inputs in order to find the inputs that minimize the function. The inputs to the function are the weights of the sub-cost functions. The function value is calculated by the following procedures:

1. Given the weights, the total cost is calculated for each rendition for a test word.
2. The ranking of the renditions can be obtained by sorting the total costs in ascending order.
3. Calculate the dissimilarity measure between the ranking obtained from the previous step and the reference ranking from the perceptual test.
4. Repeat the above steps for all test words.
5. Get the average of the dissimilarity measures of all test words.

The average dissimilarity measure is the value of the function. A set of sub-cost weights that minimize the average dissimilarity measure will most likely lead to a unit-searching algorithm that behaves as the human listeners participating in the perceptual test do.

The dissimilarity measure between any two rankings should satisfy the following conditions:

Condition 1: The dissimilarity measure is a function of any two rankings, D(r1, r2), where r1 is a reference ranking and r2 is a ranking whose dissimilarity to the reference ranking is to be calculated. The value should be proportional to the total distance (or notches) that every element in r2 have to move in order to position itself at the same rank in r1.
For example, let's assume that there are 4 elements (denoted as A, B, C and D) in the rankings. The distance between a reference ranking “ABCD” and a test ranking “BCAD” is 1(one notch down for B) + 1(one notch down for C) + 2(two notches up for A) + 0(no movement for D) = 4. Of course, if the two rankings are identical, the distance is zero. Since the dissimilarity measure is relative to the reference ranking r1, the measure is not symmetric, i.e., D(r1,r2) $\neq$ D(r2,r1).

The above condition, however, may lead to an ambiguous situation, where different rankings have the same distance. For example, d(“ABCD”,“BCAD”) and d(“ABCD”,“ACDB”) are both 4, although the latter is more meaningful in the sense that it has the right element at the first place. When there is a tie, we must give a lower dissimilarity measure to a ranking that matches higher ranked elements correctly. In the example, “ACDB” should have a lower dissimilarity measure than “BCAD”. We need an additional condition for the clarification.
Condition 2: When two rankings are tied, the ranking that matches higher ranked elements should get a lower dissimilarity measure.

A distance metric that satisfies the conditions can be calculated by binary tree searching. The pseudocode for the procedure of calculating the dissimilarity measure is:

```
initialize *distance* to zero
foreach *entry* in array T
    find the location of *entry* in array R
    set *locR* equal to the index of *entry* in array R
    set *locT* equal to the index of *entry* in array T
    compute a *localdistance* by $locR \cdot fact(n-locT-1)$
    add *localdistance* to *distance*
    remove *entry* from *R*
end
normalize the distance by multiplying by 100/(*fact*(n)-1)
```

R and T are arrays of the reference and test rankings, respectively. The number of entries of the ranking is n. The index of the first *entry* in an array is zero and the function *fact*(x) = $1 \cdot 2 \cdot \cdots \cdot x$. The distance value is from 0 to *fact*(n)-1. Since the maximum distance depends on the number of entries of the rankings, the distance is normalized by *fact*(n)-1. The normalized distance is the dissimilarity measure, which ranges between 0 (the best match) to 100 (the worst match)

7. Experimental Results

The test database consists of six single-syllable words as in Table 1. Only single-syllable words were used so that listeners can pay attention to the single concatenation in the middle of a vowel.

Each word has a different number of renditions, which depends on the number of available candidates in the speech corpus. To synthesize a word “bob”, four diphones, i.e., |*-b|, |b-o|, |o-b|, and |b-*| are required. In the speech corpus, two candidates of |b-o| diphones, two of |o-b| diphones, and a |b-o-b| triphone were available. Therefore, a total of 5 different com-

Table 1: *Optimization Results.*

word	# renditions	reference	result	Dis.Msr.
bob	5	AEDCB	ACDBE	12.5
big	7	ABCEDFG	ABFECGD	1.57
bag	5	ABECD	ADEBC	16.67
bought	7	AGEFDBC	ABCGFED	11.46
bed	7	GBAEFCD	AEDBCGF	35.43
book	7	ACGBEDF	AFDGCBE	13.92
mean				15.26

binations of the available units were synthesized [1]. The 5 renditions have the same phonemic sequence "bob", but different units were used for each rendition. Residual PSOLA algorithm was used for synthesis. The original pitch and duration were modified according to the target prosody. However, the spectral envelope was not smoothed at the concatenation point. The number of renditions for each word is shown in Table 1.

The reference rankings of the renditions were obtained by perceptual preference test. The test was one-interval, two-alternative forced-choice experiment. Two randomly chosen renditions were presented to listeners. The listeners were asked to choose one that sounds better. For a test word with n_i different renditions, there are $\frac{n_i!}{(n_i-2)!\cdot 2!}$ possible pairs of renditions. All the pairs are presented to the listeners in random order. The results are collected to get the final rankings for all of the test words. The reference rankings for all the test words are shown in the third column of Table 1.

Finally, the downhill simplex optimization was performed and the resulting weights were used for calculation of the total cost. Roughly 83 percent of the cases (5 out of 6 words), the unit selection algorithm chose the same rendition that human listeners preferred. The average dissimilarity measure was 15.26 on the scale of 0 to 100. The final rankings are shown in the fourth column of the Table 1. The results showed that the weight optimization process was able to find a set of weights that could successfully predict the human preference pattern.

We used the optimal weights for selecting units for longer sentences. Synthetic speech using weights determined in this manner consistently demonstrates smoother transitions and higher voice quality than speech using manually set weights.

8. Conclusions

In this paper, we have presented an overview of the new corpus-based Bell Labs' TTS system. Centered around a unifying data structure, the annotated string, and making use of the Viterbi dynamic programming algorithm for optimal unit selection under open-ended cost criteria, our goal for this work has been to achieve both the generality needed to support research investigations as well as the ability to target the system to specific applications. Our experiments show that unit selection is fast enough to be practical even when using a large corpus, and that longer units arise often enough to make a significant difference in the quality of the synthesized speech.

We have also proposed a new method for unit-searching based on a perceptual preference test. The proposed algorithm is designed to find the weights in a more systematic and meaningful way. The algorithm searches a set of weights that can produce rankings of renditions that are close to the perceptual test results. The downhill simplex method is used for the multi-dimensional search of the weights. A dissimilarity measure is proposed to evaluate the closeness of two rankings, one from the perceptual test and the other from the algorithm.

In about 83 percent of the cases, the unit selection algorithm using the optimal set of weights chose the same rendition that human listeners preferred. These results show that the optimization of weights combined with the dissimilarity measure can successfully predict the human preference pattern.

[1] Concatenation of |*-b| and one of the following combinations $\{$ A:$|$b-o$|_1$ + $|$o-b$|_1$, B:$|$b-o$|_2$ + $|$o-b$|_1$, C:$|$b-o$|_1$ + $|$o-b$|_2$, D:$|$b-o$|_2$ + $|$o-b$|_2$, E:$|$b-o-b$|_1\}$ followed by |b-*|.

9. References

[1] Shinya Nakajima and Hiroshi Hamada, "Automatic generation of synthesis units based on context oriented clustering," in *Proceedings of the IEEE International Conference on Acoustics and Speech Signal Processing-88*, New York, NY, 1988, IEEE.

[2] Andrew J. Hunt and Alan W. Black, "Unit selection in a concatenative speech synthesis system using a large speech database," in *Proceedings of the IEEE International Conference on Acoustics and Speech Signal Processing-96*, Munich, 1996, IEEE, vol. 1, pp. 373–376.

[3] Robert Edward Donovan, *Trainable speech synthesis*, Ph.D. thesis, University of Cambridge, Cambridge, UK, 1996.

[4] A. P. Breen and P. Jackson, "Non-uniform unit selection and the similarity metric within bt's laureate tts system," in *Proceedings of the Third ESCA Workshop on Speech Synthesis*, Jenolan Caves, Australia, 1998, ESCA/IEEE.

[5] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery, *Numerical Recipes in C - The Art of Scientific Computing*, Cambridge University Press, 1992.

[6] Thierry Dutoit, *An introduction to text-to-speech synthesis*, Kluwer Academic, Dordrecht; Boston; London, 1997.

[7] Minkyu Lee, Jan van Santen, Bernd Möbius, and Joseph Olive, "Formant tracking using segmental phonemic information," in *Proceedings of the European Conference on Speech Communication and Technology (Eurospeech)*, Budapest, Hungary, 1999, ESCA.