VoCo: Text-based Insertion and Replacement in Audio Narration

ZEYU JIN, Princeton University
GAUTHAM J. MYSORE, STEPHEN DIVERDI, and JINGWAN LU, Adobe Research
ADAM FINKELSTEIN, Princeton University

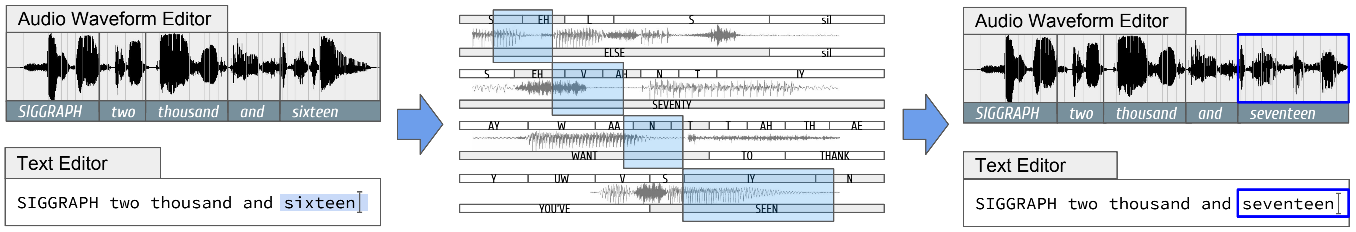

Fig. 1. Text-based editing provides a natural interface for modifying audio narrations. Our approach allows the editor to replace an existing word (or insert a new word) by typing, and the system automatically synthesizes the new word by stitching together snippets of audio from elsewhere in the narration. Here we replace the word *sixteen* by *seventeen* in a text editor, and the new audio is automatically stitched together from parts of the words *else, seventy, want* and *seen.*

Editing audio narration using conventional software typically involves many painstaking low-level manipulations. Some state of the art systems allow the editor to work in a text transcript of the narration, and perform select, cut, copy and paste operations directly in the transcript; these operations are then automatically applied to the waveform in a straightforward manner. However, an obvious gap in the text-based interface is the ability to type new words not appearing in the transcript, for example inserting a new word for emphasis or replacing a misspoken word. While high-quality voice synthesizers exist today, the challenge is to synthesize the new word in a voice that matches the rest of the narration. This paper presents a system that can synthesize a new word or short phrase such that it blends seamlessly in the context of the existing narration. Our approach is to use a text to speech synthesizer to say the word in a generic voice, and then use voice conversion to convert it into a voice that matches the narration. Offering a range of degrees of control to the editor, our interface supports fully automatic synthesis, selection among a candidate set of alternative pronunciations, fine control over edit placements and pitch profiles, and even guidance by the editors own voice. The paper presents studies showing that the output of our method is preferred over baseline methods and often indistinguishable from the original voice.

CCS Concepts: • **Human-centered computing → Interaction techniques**;

Additional Key Words and Phrases: audio, human computer interaction

ACM Reference format:
Zeyu Jin, Gautham J. Mysore, Stephen DiVerdi, Jingwan Lu, and Adam Finkelstein. 2017. VoCo: Text-based Insertion and Replacement in Audio Narration. *ACM Trans. Graph.* 36, 4, Article 96 (July 2017), 13 pages.
https://doi.org/10.1145/3072959.3073702

Author's addresses: Princeton Computer Science Department, 35 Olden Street, Princeton NJ 08540 USA; Adobe Research, 601 Townsend Street, San Francisco CA 94103 USA.

0730-0301/2017/7-ART96 $15.00
https://doi.org/10.1145/3072959.3073702

1 INTRODUCTION

Recorded audio narration plays a crucial role in many scenarios including animation, computer games, demonstration videos, documentaries, and podcasts. After narration is recorded, most of these applications require editing. Typical audio editing interfaces present a visualization of the audio waveform and provide the user with standard select, cut, copy and paste operations (in addition to low-level operations like time stretching, pitch bending, or envelope adjustment), which are applied to the waveform itself. Such interfaces can be cumbersome, especially for non-experts. Researchers have addressed this problem by aligning the waveform with a transcript of the narration, and providing an interface wherein the user can perform cut-copy-paste operations in the text of the transcript. Whittaker and Amento [2004] and Rubin et al. [2013] show that this form of interface significantly reduces search and editing time, and is preferred by users. State of the art video editing systems incorporate audio-aligned transcript editors to facilitate search and editing tasks [Berthouzoz et al. 2012; Casares et al. 2002].

While cut-copy-paste operations are supported, one aspect remains conspicuously missing from text-based audio editors: insertion. It is easy for a person to type a new word not appearing in the transcript, but it is not obvious how to synthesize the corresponding audio. Nevertheless, in many circumstances inserting a new word or phrase during editing would be useful, for example replacing a misspoken word or inserting an adjective for emphasis. It is possible to record new audio of just the missing word, but to do so requires access to the original voice talent. Moreover, even when the original narrator, microphone and acoustic environment are available for a new recording, it remains difficult to match the audio quality of an inserted word or phrase to the context around it. Thus the insertion is often evident in the edited audio, even while methods like that of Germain et al. [2016] can ameliorate such artifacts. Regardless, just as it is easier to type than to edit audio waveforms for cut and paste operations, it is also easier to type for insertion or replacement rather than record new audio.

This paper introduces a method for synthesizing a word or short phrase to be inserted into a narration, based on typing. The input is an existing recording coupled with a transcript, as well as the text and location of the word to be inserted. With the proliferation of voice-based interfaces, there now exist many high-quality text-to-speech (TTS) synthesizers. The challenge in our problem is to automatically synthesize the inserted word in a voice that sounds seamless in context – as if it were uttered by the same person in the same recording as the rest of the narration (the *target* voice). While it is possible to customize a speech synthesizer for a specific target, conventional approaches (e.g., Acapela Group [2016]) start from a large corpus of example recordings (e.g., 10 hours) and require many hours of human annotation. Our goal is to synthesize a word or short phrase (as opposed to a full sentence) that reasonably matches the target voice in the context into which it is inserted, but based on a much smaller corpus that lacks human annotation. Moreover, the synthesized voice should have *clarity* (be free of noticeable artifacts like popping or muffling) and *individuality* (sound like the target).

The key idea of our approach is to synthesize the inserted word using a similar TTS voice (e.g., having correct gender) and then modify it to match the target using *voice conversion* (making an utterance by one person sound as if it was made by another). Voice conversion has been studied for decades, and the idea of using voice conversion for TTS was proposed by Kain and Macon [Kain and Macon 1998] in the 1990s. However, only recently have voice conversion methods been able to achieve high individuality together with substantial clarity, for example the recent CUTE algorithm of Jin et al. [Jin et al. 2016]. Based on a triphone model of human voice, CUTE performs a dynamic programming optimization to find and assemble small snippets of the target voice from the corpus, such that when concatenated they resemble the TTS word. This approach has related ideas from computer graphics research, for example the *image analogies* method of Hertzmann et al. [2001] and the *HelpingHand* approach of Lu et al. [2012].

This paper introduces a method called VoCo that builds on CUTE and makes several improvements suitable for our problem: (1) it improves synthesis quality by introducing *range selection* to replace frame-level unit selection; (2) to accelerate the optimization for use in our interactive application, it introduces a new two-stage optimization approach (dynamic programming for selecting phoneme sequences, followed by *range selection* to choose audio frames that match those phonemes); (3) it introduces the notion of *exchangeable triphones* to achieve clarity with a smaller corpus (20-40 minutes) than earlier methods; (4) it optimizes matching the context of the insertion; and (5) for cases where the default output is unsatisfactory in quality or prosody, it supports interfaces by which novices and/or experts can improve the results – by choosing among a variety of alternative versions of the synthesized word, adjusting the edit boundaries and pitch profiles of the concatenated audio clips, and even adjusting the synthesis using the editor's own voice.

We present studies showing that words synthesized by VoCo are perceived as more natural than those produced by baseline methods (TTS and CUTE). Moreover, the studies show that VoCo produces output that, more often than not, is indistinguishable from the voice of the target speaker when inserted into a sentence. These studies are conservative in the sense that the goal of being indistinguishable from the target voice (for careful listeners) is stronger than the goal of being plausible or acceptable in edited narration (for casual listeners). Finally, we describe another study wherein an expert audio engineer painstakingly assembles snippets of the corpus to synthesize new words, and even this arduous process produces results that are audibly inferior those of VoCo.

2 RELATED WORK

Our approach offers several connections to work in **computer graphics**. The optimization over triphones described in Section 4.3 builds on the seminal *Video Rewrite* work of Bregler et al. [1997]. Likewise, research like that of Stone et al. [2004] and Levine et al. [2009] shows how natural gestures of animated characters can be driven by speech. Finally our approach is closely related to work that relies on optimization for example-based synthesis of textures over a scalar variable like time or arc-length [Lu et al. 2012; Lukáč et al. 2013].

Audio editing tools such as Adobe Audition and Audacity allow experts to edit waveforms in a timeline interface. Audio manipulation is achieved by chaining low-level signal processing tools that require expertise and knowledge to use. However, such general-purpose editing tools typically do not have a specialized **speech processing** interface, as they are unaware of the linguistic properties of the signal. Therefore research-oriented speech editing tools such as Praat [Boersma et al. 2002] and STRAIGHT [Kawahara et al. 2008] allow a user to manipulate phonetic and linguistic aspects of speech in a timeline where audio is presented synchronously with other properties such as phonemes and pitches. Similar to this idea, recent research in this scope aims at connecting average users with sophisticated speech processing tools using text and a middle layer: Whittaker and Amento presents a voicemail editing interface that boosts productivity with text-based navigation and word-level copy-and-paste. The idea of using script to navigate a timeline is also employed by video authoring tools such as the work of Casares et. al. [Casares et al. 2002], and recent versions of commercial software such as Avid Media Composer. It is also adopted by video browsing and symmetrization tools such as Video Digest [Pavel et al. 2014] and SceneSkim [Pavel et al. 2015]. In speech editing, text is not only used for browsing but editing as well: Berthouzoz et al. [2012] created interview video editing tool that allows an editor to place cuts and pauses by cutting words and adding pauses in the transcript. They also show "suitablility" scores as colors in the transcript to indicate suitable points to place cuts. Later, Rubin et al. [2013] presented an editing tool for audio stories that support cut, copy and paste in text and selecting among different takes in the text. One important aspect of text-based speech editing, however, is missing from these approaches – word replacement and insertion, crucial for words that do not exist in the transcript. This is the main focus of our work.

Kain and Macon [1998] showed how to create new TTS voices with relatively little data using **voice conversion**. Typical voice conversion methods represent voice signals as parametric source-filter models in which the source approximates the vocal cords that generate periodic or noise signals and the filter mimics the vocal tract that shapes the source signal into the sounds we hear [Taylor 2009]. Then these methods explicitly model a conversion function

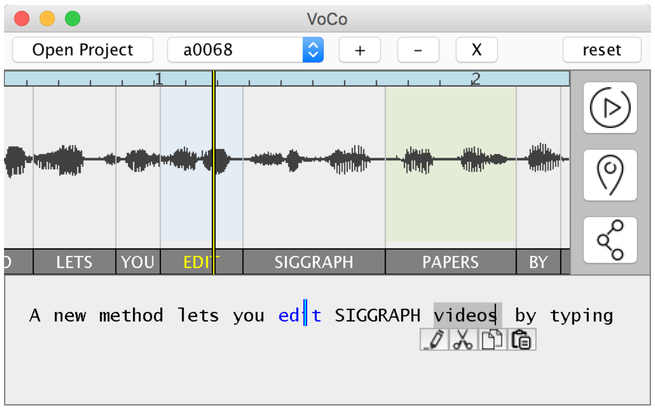

Fig. 2. Main Editing Interface. An editor can cut, copy and paste text in the transcript in order to effect changes in the narration. Our main contribution is that the editor can also replace existing words or type new words not in the narration (in this case replacing *papers* with *videos*) and our system will synthesize a result that matches the voice of the speaker.

from one voice (source) to another (target) in the parametric space. Given a new source (query), they apply the conversion function and then re-synthesize actual signals from the converted parametric model. The most classical conversion function is based on mapping over Gaussian Mixture Models [Stylianou et al. 1998; Toda et al. 2001]. Recent research has also explored other forms of probabilistic modeling [Toda et al. 2007a], matrix factorization [Aihara et al. 2014] and deep neural networks [Chen et al. 2014; Desai et al. 2009]. These approaches achieved very low distortion in the parametric space but the output signal contains muffled artifacts possibly not captured by the distortion measurement or introduced when re-synthesizing from parametric representations of MFCC [Desai et al. 2009] or STRAIGHT [Kawahara et al. 2008]. The artifacts tend to be perceptible when inserting a converted word between words of an existing speech signal, and thus unsuitable for our word insertion application.

In another thread of voice conversion research, algorithms concatenate segments of a target speaker's voice to approximate the query; a method called **unit selection** is used to select these segments such that they are close to the source in content and have a smooth transition from one segment to the next [Hunt and Black 1996]. Recent approaches in this this thread differ in how units are defined and selected: for example Fujii et al. [2007] segment speech into phonemes and concatenate these phonemes to form new content. Another thread of research performs frame-level unit selection (thereby increasing the number of units) to overcome the shortage of voice samples [Dutoit et al. 2007]. Wu et al. [2013] perform unit selection on exemplars, time-frequency speech segments that span multiple consecutive frames [Raj et al. 2010]. Although these methods produce words that are free from muffled artifacts, they tend to have new problems with unnatural pitch contours and noise due to discontinuities between units. The recent CUTE method Jin et al. [2016] further reduces distortion by a hybrid approach using phoneme-level pre-selection and frame-level unit selection. Our method VoCo builds on CUTE and has several improvements for better performance in word insertion.

3 EDITING INTERFACE

Our goal is to make editing narrations as simple as text editing for both novices and experts, while providing a range of customization options for motivated and advanced editors. To achieve this goal, our interface inherits the basic cut-copy-paste functionality of the speech editor of Rubin et al. [2013]. In addition our interface allows a user to type new words (via insert or replace) and then automatically synthesize and blend them into the audio context. A non-expert user can customize the synthesized words by selecting among a number of alternative results. More experienced editors can adjust the length, pitch and amplitude of the stitched audio snippets to further refine the synthesis.

3.1 Text-based Editor

To start, a user selects an audio recording and its corresponding transcript. Our system aligns the loaded audio recording with the transcript and builds a customized voice synthesizer based on the voice characteristics of the selected recording (Section 4). Then, the main editing interface launches, shown in Figure 2. The upper part visualizes the audio waveform segmented by individual words. The lower part displays the transcript and provides a text pad in which users can perform edits. When playing back the audio, two cursors appear in synchronized locations in both the audio and the text. A user can move the cursor by clicking on either the waveform or the text. Basic operations including *delete, cut, copy* and *paste* are allowed in the text editor. These operations will be reflected immediately in the audio visualizer. The primary contribution of this paper is that a user can also insert and replace words via typing; the synthesizer will then synthesize the new words in the voice of the person in the given recording and blend them into context as seamlessly as possible. An edit button will appear and can be clicked if the user chooses to customize the synthesis.

3.2 Alternative Syntheses

There are multiple ways to speak the same word, and therefore it is natural to provide a user with alternative synthesis results. As shown in Figure 3, the alternative results are organized in a list. Via radio buttons, the user can quickly listen to all the alternatives, and then optionally change modes (by a checkbox) to listen to how each alternative sounds in the context of the full sentence. Studies presented in Section 5 show that in general allowing the user to select among alternatives produces a more natural sounding result.

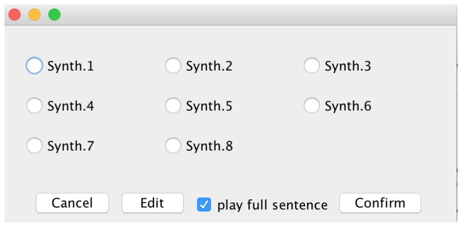

Fig. 3. Interface presenting alternative results. When the editor is not satisfied with the default synthesis, this interface allows them to explore and choose among several alternative pronunciations.

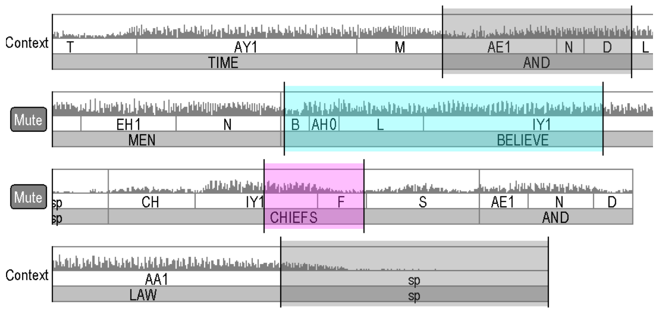

Fig. 4. More advanced users can have fine-grain control over the boundaries of audio snippets selected by our algorithm.

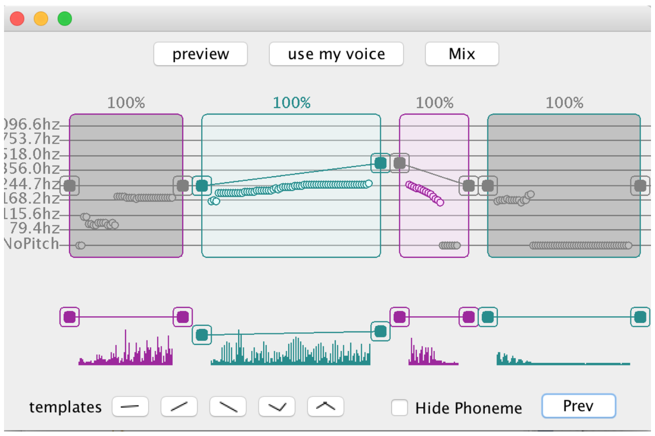

Fig. 5. Advanced editors can manually adjust pitch profile, amplitude and snippet duration. Novice users can choose from a pre-defined set of pitch profiles (bottom), or record their own voice as an exemplar to control pitch and timing (top).

3.3 Manual Editing

To allow users with audio editing skills to customize the synthesis result, we introduce two advanced editing options which can be applied sequentially. Recall that the synthesized word is created by stitching together snippets of words from the target voice corpus. By clicking on the *Edit* button in the *alternative syntheses* window, a user is presented with a new window containing multiple audio tracks that depict these snippets. In Figure 4, a user can adjust the boundaries of the snippets, change the lengths and find new stitching points. After previous changes are confirmed, a second window appears (Figure 5) that allows the user to adjust the pitch contour of each snippet (shown as a graph of dots) either by dragging handles or by choosing one of a pre-selected set of pitch profiles shown at the bottom of the window. This window also allows the editor to adjust the amplitude and/or duration of each snippet by dragging the top or sides of the box that surrounds it. Finally, as an alternative to manually adjusting the pitch and timing, a user can speak into a microphone to *demonstrate* how a target word should be spoken. The target word will then be re-synthesized taking into account the users' pitch and timing features. Studies presented in Section 5 show that manual editing can further improve on words selected via alternative syntheses.

4 ALGORITHMS

Our word replacement and insertion tasks require synthesizing new audio based on typed text, a process often referred to as text-to-speech (TTS) synthesis. State of the art commercial TTS systems rely on a large corpus with a significant amount of manual annotation [Tokuda et al. 2013], neither of which is available in our setting. In order to build a *target TTS* synthesizer for an arbitrary target voice with a relatively small amount of data and no manual annotation, we perform voice conversion with the following procedure:

(1) **Corpus preparation** - We first align the target speaker's speech samples to the transcript using a *forced alignment* algorithm [Sjölander 2003]. This method converts the transcript to phoneme sequences and then establishes the mapping between phonemes and speech samples in time. This step is also used by existing text-based speech editors such as that of Rubin et al. [2013].
(2) **TTS selection and preparation** - Given the *target* speaker's speech recording and transcript, we first choose an existing synthetic voice, the *source* TTS, which sounds similar to the target. This is done by sampling several sentences from the target speaker's narration, and synthesizing the same content in a variety of TTS voices. Then we calculate the acoustic distances between the target and each of the TTS voices using Mel-Cepstral Distortion (MCD) [Kubichek 1993], a metric that measures spectral difference between two audio samples. We use the TTS voice with the lowest MCD distance to the target, as the source TTS voice.
(3) **Building a voice converter** - We build a voice converter based on parallel samples of the source and target voices. Since we have the target narration and transcript, we use the source TTS to synthesize the parallel speech in the source speaker's voice. We therefore do not need a pre-existing parallel speech corpus. Similar to the target voice, we segment the source voice into phonemes using *forced alignment*. The segmented audio and transcript (both source and target) serve as the training data for the proposed data-driven voice converter (Section 4.1 through 4.4).
(4) **Synthesis and blending** - Given a new word, we synthesize the corresponding audio using the *source* TTS, transform the result using the trained voice converter to make it sound like the target speaker, and finally insert the synthesized word into the given context using cross fade.

4.1 CUTE voice conversion

In this section we describe CUTE, the voice conversion method recently introduced by Jin et al. [Jin et al. 2016]. As noted above, we use voice conversion to convert our source voice (produced by TTS) to match our target voice. CUTE is a data-driven voice conversion approach that uses the source voice as a query and searches for snippets of the target voice that as much as possible sound similar to the query, while also matching at the stitch boundaries. CUTE first segments both the source and target audio into small, equally-sized windows called *frames*. It then needs to find the optimal sequence of frames of the target voice in order to perform the synthesis. It does so via an optimization that minimizes two costs: matching cost between the selected target and the source frames and concatenation cost between adjacent frames. This optimization can be performed via dynamic programming, much as drawing gestures are stitched together in the HelpingHand approach of Lu et al. [Lu et al. 2012]. For interactive performance, CUTE relies on *triphone preselection,* which limits the search space to a small number of candidate frames, and is crucial in preserving synthesis quality. In particular, rather than using the KNN algorithm as in HelpingHand, CUTE selects candidates that are phonetically consistent with the query, meaning they share a similar sequence of triphones. Sections 4.2–4.4 offer several quality and performance improvements over CUTE. Nevertheless, we briefly describe CUTE here as a baseline approach for voice conversion in our system:

1. **Frame-by-frame alignment**: First, we segment the audio into frames (in our implementation, the window size is 25ms and consecutive frames overlap by 5ms%). Next, we apply Dynamic Time Warping (DTW) to align the source frames with the target frames using a distance metric defined as the Euclidean distance between the corresponding MFCC features [Muda et al. 2010].

2. **Feature extraction and exemplars**: For both source and target voices, we extract the per-frame MFCC and F0 feature vectors. The F0 vectors representing the fundamental frequency (also referred to as *pitch*) are normalized to the target voice's F0 range using a logarithmic Gaussian normalized transformation [Toda et al. 2007a]. We concatenate the feature vectors of $2n + 1$ consecutive frames into exemplars [Wu et al. 2013]. These exemplars not only encode frequencies, but also temporal variation in frequencies.

Using the frame alignment from the first stage, we combine the source MFCC exemplars and target F0 exemplars into joint *matching exemplars*. Minimizing distance between the query and matching exemplars helps the synthesized voice sound like the query. Likewise, the target MFCC and F0 exemplars are combined into *concatenation exemplars*; minimizing the distance between neighboring concatenation exemplars encourages smoothness at stitch boundaries.

3. **Triphone preselection**: For performance, this step removes frames that are unlikely to be selected in the matching step. Based on phoneme segmentation, each frame is assigned a phoneme label. These labels are further converted to *triphones* – a phoneme and its two neighbors. For example, the triphones for the word "user" (phonemes: `Y UW1 Z ER0`) are

```
(st)_Y_UW1, Y_UW1_Z, UW1_Z_ER0, Z_ER0_(ed)
```

where `(st)` and `(ed)` label the beginning and the end of a sentence. For each query frame (source voice), frames (target voice) that share the same triphone label are selected as candidate matches. If there are no exact triphone matches, CUTE selects the frames in which either the first two or last two phonemes (called *diphones*) match with the query triphone. Likewise, if no diphones are available, candidates that only match the central phoneme (*monophone*) are selected. This way, frames that are most phonetically similar to the query are preserved for the next step.

5. **Matching**: In the speech synthesis literature, this matching step is called *unit selection* [Taylor 2009]. Based on the candidate table, the Viterbi algorithm [Forney 1973] is used to select one candidate frame per query frame such that the sum of two cost functions are minimized. The first is *matching cost*, defined as the Euclidean distance between the query exemplar and the candidate's matching exemplar. The second is *concatenation cost* designed to characterize the distortion introduced by concatenating two candidates. It is the product of three terms, *break cost*, *smoothness cost* and *significance cost*. The two cost functions together, aim to choose candidates that result in the least number of breaks and if there are breaks, they are in the least significant part of the synthesis and their transition is smooth.

6. **Concatenation**: Consider sequences of consecutive frames as audio snippets – we need to combine these snippets at their stitch points. We use the WSOLA algorithm of Roelands and Verhelst [1993] to find the overlapping areas between adjacent snippets where the waveforms are most similar to one another, and blend the snippets together using a cross-fade.

We have implemented CUTE for our application, and as shown by experiments described in Section 5, the quality of the synthesized result is often satisfactory. However, CUTE has several drawbacks in our context. First, CUTE is slow whenever matching triphones are not found in the preselection stage. In that case CUTE falls back on diphones or monophones, of which there is a much larger candidate pool, making the candidate table very large. Second, the CUTE algorithm searches for snippets that match at internal stitch boundaries, but does not optimize for blending an inserted word into its context – sometimes leading to audible artifacts. Third, CUTE has difficulty with accents. Suppose the source and target speakers use different pronunciations (both of which are valid) for the same word in the same context. When synthesizing the same word, CUTE cannot emulate the target speaker's accent because of mismatched phonemes between the query and the candidates. Finally, CUTE sometimes fails to synthesize natural sounding prosody. This is especially the case when we have limited training data, as the selected segments might have pitch discontinuities at the stitch boundaries.

In the remainder of this section we introduce several improvements to CUTE that address these limitations: use of *Exchangeable Triphones* allows for alternative pronunciations. *Dynamic Triphone Preselection* further minimizes the search space while preserving meaningful candidates. *Range Selection* replaces the traditional Unit Selection described above with a faster method for selecting audio snippets directly. Finally we introduce an approach for generating *Alternative Synthesis Results* that all tend to sound reasonable but emphasize different characteristics of the target voice.

4.2 Exchangeable Triphones

Recall that when CUTE cannot find an exact triphone match, it falls back on diphones or monophones, which leads to a much larger candidate set containing worse matches, which impacts quality and performance. Here we introduce two types of *exchangeable triphones* that expand the triphone matching to include similar triphones that lead to only a slightly different pronunciation, while maintaining a much smaller search space than the diphone fallback.

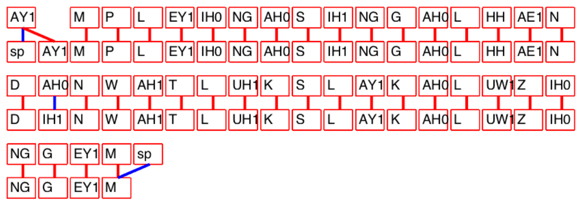
AY1 M P L EY1 IH0 NG AH0 S IH1 NG G AH0 L HH AE1 N
sp AY1 M P L EY1 IH0 NG AH0 S IH1 NG G AH0 L HH AE1 N
D AH0 N W AH1 T L UH1 K S L AY1 K AH0 L UW1 Z IH0
D IH1 N W AH1 T L UH1 K S L AY1 K AH0 L UW1 Z IH0
NG G EY1 M sp
NG G EY1 M

Fig. 6. Alignment of two phoneme sequences (source above and target below). Red lines indicate a match whereas blue lines show mismatching phonemes (and thus suggest potential accent mappings).

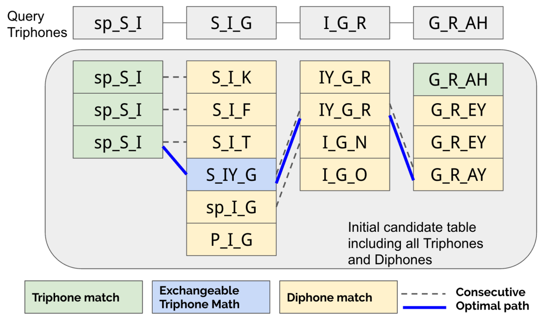

Fig. 7. Dynamic triphone preselection. For each query triphone (top) we find a candidate set of good potential matches (columns below). Good paths through this set minimize differences from the query, number and severity of breaks, and contextual mismatches between neighboring triphones.

Word-based Exchangeable Triphones come from a word-to-phoneme dictionary like the one used in the aforementioned forced alignment process [Sjölander 2003]. Some of the triphones in a word are exchangeable because alternative pronunciations of the word exist. For example the word "this" has two pronunciations, `DH_IH1_S` and `DH_AH0_S`, in the dictionary. So when we search for one of those triphones in our corpus, we will accept the other as a valid match as well.

Accent Exchangeable Triphones are not defined in dictionaries, but rather are discovered via the phoneme alignment component of the DTW-based source to target alignment process outlined above in the first stage of the CUTE algorithm (Section 4.1). In that stage, we discover cases where the source tends to utter one triphone whereas the target utters a different triphone, when saying the *same* text. Figure 6 shows an example of such phoneme alignment. In this example (third and fourth rows), we find that `D_AH0_N` and `D_IH1_N` are exchangeable. Therefore, just as with word-based exchangeable triphones above, we allow matches between these triphones when searching the corpus. This allows for a target speaker with a particular accent to be better matched with the more generic source TTS voice.

4.3 Dynamic Triphone Preselection (DTP)

When a query triphone has no matches in the corpus, the CUTE algorithm instead fills the candidate table with diphones or monophones as described above. This is somewhat ameliorated by the exchangeable triphones introduced in the previous section. Nevertheless, the problem can still occur. Since the complexity of the Viterbi algorithm scales quadratically with the number of candidates, having a large number of inferior candidates (matching only the diphones and monophones parts) is undesirable. Moreover, not all diphones are necessary. When the matching step imposes a large break penalty for snippet boundaries, only segments with the smallest number of breaks tend to be selected. Therefore we can remove candidates that might result in larger number of breaks from the candidate table. Essentially what we do is perform a dynamic programming optimization at the phoneme level, to reduce the set of candidates to only those triphone sequences with the optimal number of breaks. This hugely accelerates the frame-level search that follows (Section 4.4).

To construct an optimally compact *frame candidate table*, we first perform Dynamic Triphone Preselection, or DTP, at the phoneme level to obtain a *triphone candidate table*. We call audio frames spanning a phoneme, a *phoneme segment*. One triphone can have multiple matching phoneme segments. The goal of DTP is to select a small set of segments per query triphone such that the sequences of frames that have the minimal number of breaks are contained. In other words, this method finds optimal paths through phoneme segments that contain minimal number of breaks and then merges them into the candidate table.

The method is illustrated in Figure 7. For each query triphone q, we first initialize its candidate table with all matching Triphones, with exchangeable triphones also included. Similar to the triphone preselection step, when exact matches are unavailable, diphones and monophones are used instead. Between neighboring phonemes, some segments are consecutive (dotted links) in the original target speech and some are not. Non-consecutive phoneme segments will surely introduce a break in the matching step and thus should be minimized with priority. We also want to minimize the number of neighboring Triphone segments that do not match, e.g. `sp_S_I` and `P_I_G`, because they are likely to cause audible distortion when they are stitched together. Finally, we should also minimize the number of Diphones and Interchangeable Triphones because they are an approximation to the desired Triphone in the query. Putting all these criteria together, this problem is to minimize an energy function of matching cost (similarity between query Triphone and a candidate Triphone segment) and concatenation cost (whether there is a break and whether a segment matches its previous segment's Triphone). This is very similar to those in the matching step of CUTE and HelpingHand [Lu et al. 2012], and can be solved with the Viterbi Algorithm. Figure 7 shows an example optimal solution. Note that there could be multiple optimal paths, and all of them should be included. Finally, we retain only the triphone segments that are included in the optimal paths and omit the rest. The retained segments form the input to the *range selection* step that follows.

4.4 Range Selection

In the *matching* step of the CUTE algorithm, the preselected triphones are translated into corresponding frames and the optimal sequence of frames are selected using the unit selection. In this paper, we propose a new method, *Range Selection*, to replace Unit Selection. Instead of selecting individual frames and indirectly encouraging

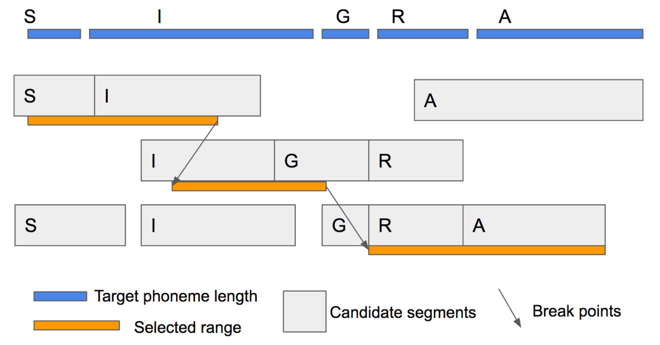

Fig. 8. Range selection objective illustrated. The blue bars are the query phonemes with length proportional to their duration. The gray wider bar are candidate segments; consecutive segments are visually next to each other. The orange bar depicts one possible selection of the candidates while the arrows show where stitching occurs between the selected range of segments.

continuity using a high concatenation cost, *Range Selection* selects ranges of consecutive frames directly by choosing their starting and ending boundaries so that the sequences sound similar to the query in terms of phonemes and pace and contain little distortion at stitching points (Figure 8).

Selecting a range of frames instead of individual ones offers several advantages: (1) it allows the definition of similarity at the sequence level and thus more sophisticated similarity measures such as Mel-Cepstral Distortion can be used; (2) it is an order of magnitude faster than frame level unit selection – by a factor of average number of frames per phoneme; and (3) it includes duration of phonemes as part of the cost function, offering explicit controls on the pace of the resulting synthesis – it is superior to the skip and repeat costs in unit selection, which do not have a direct influence on the duration of the synthesized signal.

Based on pre-selected phoneme segments, *Range Selection* finds a small set of subsequences from those segments or “range”, expressed by its starting and ending frame numbers. We use $\langle s, t\rangle$ to present a range from Frame s to Frame t. Since dynamic triphone preselection (Section 4.3) ensures that breaks will only occur once per phoneme, we can limit the number of ranges selected per phoneme to be at most two (e.g. Phoneme I in Figure 8). It means for each query phoneme, at most two candidate segments are considered, and the ranges we select from them are their subsequences.

Since only one break is allowed per phoneme, the break will occur in one of the two locations: (Case 1) inside a phoneme, e.g., the break inside Phoneme I in Figure 8; and (Case 2) between two phonemes, e.g., the break between Phoneme G and R in Figure 8. In Case 1, the range selected must cover the beginning of phoneme (first candidate of Phoneme I) because otherwise it will introduce one more break, violating the rule. Then it will transition to either another candidate in the same phoneme (second candidate of Phoneme I) or to the next phoneme (phoneme G transitioning to R). Note that transitioning to another candidate of the same phoneme means a second range is selected and the second range must extend to the end of the phoneme (second candidate of Phoneme I); otherwise an extra break will occur. In Case 2, there should be only one range selected inside the phoneme because if there are two, one more break is introduced.

To optimize for similarity, smooth transition and pace, we define an objective function in the following way. Let R_{ij} be the j-th candidate segment for Phoneme i chosen by DTP. Each segment can be represented with two numbers, the beginning and ending frame indices, $R_{ij} = \langle b_{ij}, e_{ij}\rangle = \{b_{ij}, b_{ij}+1, ..., e_{ij}\}$. Let variable $\mathcal{R}_{ik}$ be the k-th *selected range* for phoneme i, where $\mathcal{R}_{ik} = \langle s_{ik}, t_{ik}\rangle$. Define the set of ranges selected for Phoneme i as $\mathcal{R}_i = \{\mathcal{R}_{ik} \mid k \in (1, K_i)\}$ where $K_i \in 1,2$ is the number of ranges selected for Phoneme i. *Range Selection* minimizes the following function:

$$O_{rs} = \sum_{i=1}^{n} (\alpha S(q_i, \mathcal{R}_i) + \beta L(q_i, \mathcal{R}_i)) + \sum_{i=1}^{n} C_i + \sum_{i=2}^{n} \mathcal{D}_i \quad (1)$$

where q_i is the i-th query phoneme; Functions S and L measure similarity cost and the duration cost between the query phoneme and the selected ranges $\mathcal{R}_i$. Their weights are controlled by a constant value *alpha* and *beta*. Functions C_i and $\mathcal{D}_i$ are two types of concatenation costs that penalizes concatenating *ranges* that are dissimilar to one another at the boundaries [Conkie and Isard 1997]. C_i is used for a concatenation point in the middle of a segment (Case 1) and $\mathcal{D}_i$ for a concatenation point at the beginning (Case 2). Through our experiments, we found that balancing between concatenation, similarity and duration cost ($\alpha = 1$ and $\beta = 6$) is most likely to produce good results. The above optimization problem can be solved efficiently with dynamic programming. See the detailed description in the appendix.

4.5 Alternative Syntheses

While the default range selection method uses predefined α and β values in our experiments, alternative syntheses can be produced by using different combinations of α and β. Since α is the weight of similarity, the higher the value of α, the closer to the query the synthesis sounds will be, in both pitch and timbre. A higher α is more likely to lead to stitching distortion while a lower α is likely to lead to a smoother synthesis result since it is less restricted in terms of pitch and timbre. Similarly, higher β makes the duration of the synthesis closer to the query while lower β is less restrictive in terms of duration. Since range selection is effecient, we can quickly produce a grid of results, varying α and β in log space (Figure 9).

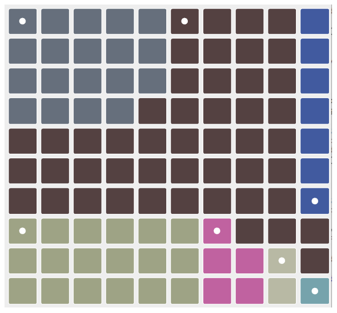

Fig. 9. Alternative syntheses. A grid of results produced by varying values of α and β for range selection. Each cell represents one alternative, and they are color coded such that similar results have the same color.

Though there are many of cells in the grid, only a handful sound unique. We group synthesis results using the same phoneme segments and color-code the grid by group. Inside each group, we retrieve a representative result by choosing the one with the minimal number of breaks. The most representative nodes in Figure 9 are marked with dots. These comprise the alternatives presented to the user, as shown in Figure 3.

5 EXPERIMENTS AND RESULTS

This section describes some synthesis results, as well as several studies wherein we asked subjects to evaluate the quality of these results. The audio files used in these experiments can be found at our project web page: http://gfx.cs.princeton.edu/pubs/Jin_2017_VTI/

5.1 Synthesis Examples

In this section we show some example results from our speech synthesis algorithm to demonstrate how our proposed method achieves the goals of contextual smoothness and having as few breaks as possible. The CMU Arctic dataset [Kominek and Black 2004] is used in these experiments.

Figure 10 shows a first example, where we synthesize the word “purchase” and insert into a recording where it is surrounded by silence. Multiple copies of the word “purchase” exist in the corpus, but because they are all chained with neighboring context words they would be unsuitable for simply copying into a silent context. Instead, our algorithm finds a different combination of audio pieces where PUR is from “pursue” (preceded by silence) and ASE is from the ending of “this” (followed by silence). Between those fragments, the middle part of “speeches” connects them. Note that because of exchangeable triphones, Z and S are allowed to be matched.

The next example is a homophone. The query is “prophet” without context. The result we obtained has no break at all. It is one solid piece of the word “profit”. This demonstrates our approach not only finds exact word matches if they exist, it also finds homophones if

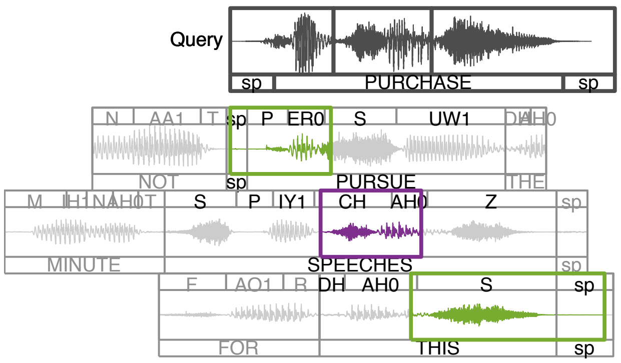

Fig. 10. Synthesis example: the word “purchase” placed in a silent context (not audibly linked to neighboring words). The query is above, segmented by where breaks occur in the synthesis result. The text beneath it shows the spans of words. The following three chunks of waveform show the snippets of audio that are concatenated during synthesis (green, purple, green). For visualization, the query waveform is stretched or compressed to match the length of chunks below.

they fit. The two lines below this paragraph show the query and the piece(s) used to synthesize that query. The first row is the query’s phoneme decomposition. Then the row below it shows information about a piece, including sentence ID, frame numbers, phonemes and words contained in this piece.

```
Query: P R AA1 F AH0 T               (prophet)
u0518: (334-407) P|R|AA1|F|AH0|T   (profit)
```

The third example is alternative synthesis. When putting the word “poison” between “the” and “of the strange vegetation”, we obtain alternative combinations of pieces that have different prosody and styles:

```
Query: P OY1 Z AH0 N               (poison)
COMBINATION 1:
u0277: (359-448) P|OY1|Z|AH0|N   (poisonous)
COMBINATION 2:
u0277: (359-402) P|OY1           (poisonous)
u0412: (107-120) Z               (is)
u0519: (243-277) Z|AH0|N         (is in)
COMBINATION 3:
u0141: (309-354) P|OY1           (pointing)
u0020: (073-121) Z|AH0|N         (clubs and)
```

Although the second alternative combination has the largest number of breaks, it sounds most natural within the context. It is selected in the sampling method because it has competitive pitch continuity with other alternative syntheses. Also note that we select segments across words if there is no silence detected between them.

5.2 Mean Opinion Score Test

We conducted two experiments in Technical Turk to evaluate our methods. The first one is a Mean Opinion Score (MOS) test [Machado and Queiroz 2010] that asks subjects to rate the quality of the inserted synthetic words. The second one is an identification test where subjects will tell whether they think a sentence has been edited or not. Four voices, two male and two female, from the CMU Arctic dataset, are used to create test samples. We used the first 800 utterances (40 minutes) for training and the remaining 332 utterances for word replacement. We randomly choose 44 words that spans 4 to 11 phonemes from 44 different sentences. For each sentence, we remove the chosen word, synthesize the word and insert it in the same location to create a recording with one word altered. We synthesize the same word using various other methods for comparison. There are 6 conditions where sentences are made:

- **Synth.** We use the source TTS voice to synthesize the word and put into context.
- **CUTE.** Based on our word synthesis framework, we use CUTE instead of VoCo for the voice conversion.
- **Auto.** Our method using pre-defined α and β values in range selection.
- **Choose.** We manually choose one synthesis from a number of alternatives (up to 16), if it improves on Auto above.
- **Edit.** We use the editing interface to further refine the synthesis, if it improves on Auto/Choose.
- **Real.** the actual recording without modification.

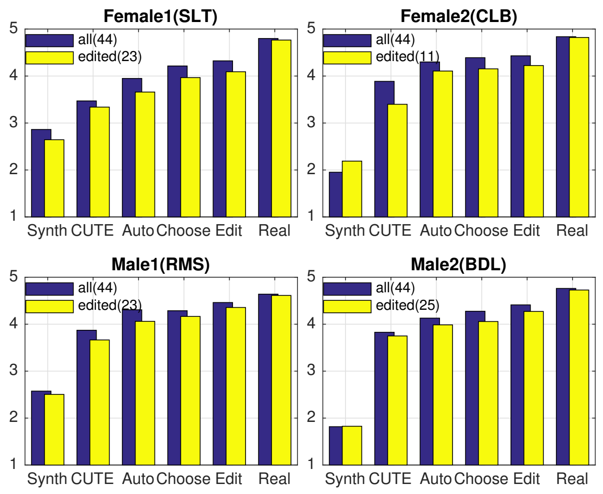

Fig. 11. These mean opinion score tests show that VoCo synthesis results are generally perceived as higher quality than those of baseline methods, scored on a Likert scale from 1=*bad* to 5=*excellent*.

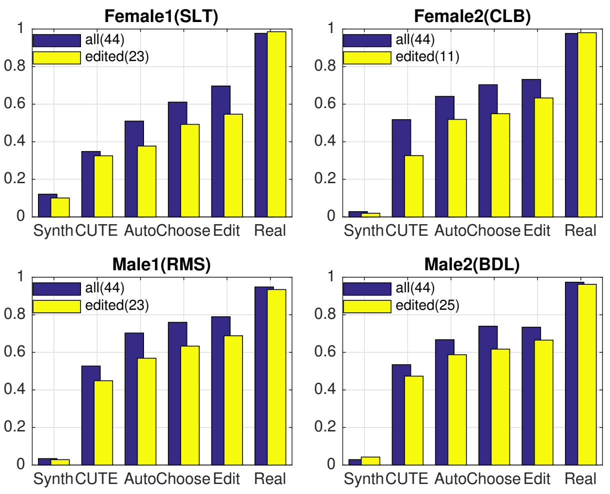

Fig. 12. Identification tests show that VoCo synthesis results are more likely identified as original recordings than other baseline methods, shown here as a fraction of people who identify the recordings as original.

In each MOS test, we present a subject with a recording drawn randomly from 44 sentences and 6 conditions above. They are informed that each sentence has one word corrupted but restored by a computer program. They are asked to listen and rate the quality of the restored sentence on a Likert scale: 1 = bad (very annoying), 2 = poor (annoying), 3 = fair (slightly annoying), 4 = good (perceptible but not annoying) and 5 = excellent (imperceptible, almost real). They can play the recording multiple times.

On Amazon Mechanical Turk, we launched 400 HITs, 100 per voice. At the beginning of each HIT, we use a warmup question to get the subject familiar to the voice they are about to hear, where they are asked to choose a word heard in a long sentence. In each HIT, a subject is presented with 32 different sentences in which 24 of them are made of 4 instances from the above 6 conditions. From a held-out set of sentences, we add 4 more instances of the "Real" condition and 4 more cases of badly edited "Synth" condition to validate that the subject is paying attention and not guessing randomly. For the data to be retained, the subject may make at most one mistake on these validation tests, by either rating < 3 on "Real" examples or > 3 on "Synth" examples. The sentences and conditions are globally load-balanced using a database, so that we cover every sentence-condition combination a similar number of times.

In the end, we collected 365 valid HITs (out of 400 issued). For each condition per voice, we average all the ratings to produce the bar plot shown in Figure 11. The blue bars show average rating of all sentences while the yellow bars only include sentences that were manually edited (because it improved on the automatic method). From the blue bars, we can see that the MOS is ascending in the order of Synth, CUTE, Auto, Choose, Edit and Real. The average scores for Auto, Choose and Edit are both over "Good" and they significantly outperform CUTE that resides in the range of "fair". For the male voice "RMS" specifically, the rating is very close to "Real" example. From the results on Edited examples, we can see all methods generally have reduced rating because these examples are hard to synthesize and thus require editing. We also notice that "Edit" consistently outperform "Choose", which indicates that manual editing is useful in improving quality for some sentences.

5.3 Identification Test

In the Identification test, a subject is presented one sentence drawn from the six conditions described above (Synth, CUTE, Auto, Choose, Edit and Real) and informed that there may or may not be one word edited using a computer. The task is to identify if the recording is original or edited. In this experiment, we released 440 HITs on Mechanical Turk, 110 for each voice. These HITs incorporate the same warm-up question as in the MOS test, as well as the same procedure relying on held-out sentences for validation with at most one mistake allowed. Each HIT contains 24 actual test sentences drawn randomly from the six conditions, together with 8 validation sentences. The sentences and conditions are load-balanced such that no sentence is repeated during a HIT, and globally all pairs of sentence and condition are covered roughly uniformly.

For this experiment we collected 391 valid HITs in total (out of 440 issued). For each method in each voice, we compute the percentage of subjects who think the recordings they heard are original. As with the MOS tests, we show both the results for all sentences as well as just the edited sentences in Figure 12. From the blue bars (all sentences), we observe that the likelihood of a synthesis being identified as original ascends in the order: Synth, CUTE, Auto, Choose, Edit. The chance that Auto is confused with an actual recording is on average 57% for the female voices and 68% for the male voices which is significantly higher than those of CUTE (43% and 53%). Based on Fisher's exact test, Auto improves on CUTE with $p << 1 \times 10^{-16}$, indicating that our fully-automatic approach performs significantly better than a state of the art method for this task. Likewise Choose improves over Auto with $p < 2{\times}10^{-5}$,

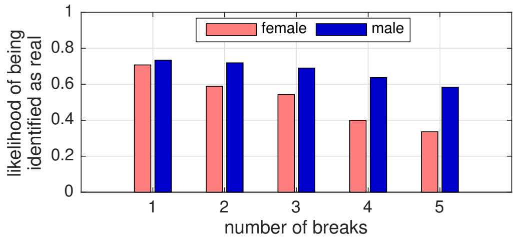

Fig. 13. Identification test results as a function of number of breaks in the synthesis. For female voices, the result degrades significantly as the number of breaks increase while in male voices, the trend is less strong.

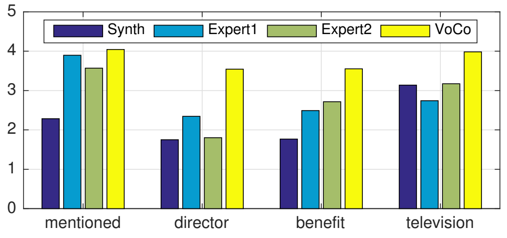

Fig. 14. Mean opinion scores for experts and algorithms replacing words in sentences with synthesized words: *mentioned, director, benefit* and *television*. VoCo outperforms the experts, while Synth provides a baseline.

meaning that choosing among alternatives improves the results, at least in some cases. We also find that Edit improves on Choose ($p < 0.027$ using Fisher exact test), with significance even though manual editing is used only in difficult cases where choosing among alternatives does not yield adequate results.

For the Edit condition, the chance of being confused with actual recordings is 71% for female voices and 76% for male voices. This performance gap between female and male voices is reduced with alternative syntheses and manual editing. Note, however, that this gap is not observed at all in the MOS test, where our method performs equally well on female and male voices. We speculate that this difference is observed because males voices generally sound noisier than female voices, and the noise obscures discontinuities introduced by piecing together audio snippets. Evidence supporting this hypothesis is revealed in Figure 13. The probability of a synthesized word being identified as real is plotted as a function of the number of breaks points in synthesizing a word using the fully automatic method. For female voices there is a strong descending trend whereas the trend is less strong for male voices.

5.4 Human synthesis

VoCo makes it possible for non-experts to insert and replace words in audio narration by typing. But does it make editing easier and/or generate better results for audio experts? To answer these questions, we asked two audio experts to manually create new words and insert them into a sentence using software tools with which they were familiar. One expert chose to work with the open-source audio editor Audacity[1] while the other used commercial application Adobe Audition.[2] Four words – *mentioned, director, benefit* and *television* – were were picked for the task because they contain non-trivial numbers of phonemes and there were sufficient triphones to synthesize them in the corpus. The experts were presented with 80 seconds of recordings, selected so as to ensure that there were abundant material to create those specific words. We used only 80 seconds of recordings to narrow the search space and thereby save human search time, thus measuring an upper bound of human performance. The actual human performance on a real corpus that contains tens of minutes of recordings should be considerable worse than what is demonstrated in our experiment. For comparison, we also synthesized these words using VoCo and the Synth voice as a baseline, each based on the same corpus.

We clocked the time the experts spent to create each new word: Expert 1 spent 15-27 minutes and Expert 2 spent 30-36 minutes per word. For comparison, VoCo only takes about one minute to process the 80-second corpus, and less than a second to synthesize each word. We also performed an MOS test on Mechanical Turk for the manually-edited words as well as two automatic methods – VoCo and Synth. There are 16 unique tests in total (4 words crossed with 4 conditions), and each unique test was shown twice per HIT (which contained 32 tests in total, presented in random order). Since the subject answered each question twice, we were able to reject data from subjects whose answers were inconsistent (more than 2 points apart on a question). Thus we collected 48 valid HITs, the results of which are shown in Figure 14. From the MOS scores, we can see that VoCo outperforms human editing in all four cases. Also the results of the generic synthesizer were considered no better than those of the experts, since they are in a voice that's slightly different from the voice used in the sentence. This study suggests that VoCo makes word insertion more efficient and effective than manual editing by audio experts.

6 CONCLUSION AND FUTURE WORK

This paper describes an algorithm that supports text-based editing of audio narrations by augmenting the relatively straightforward cut-copy-paste operations with the more challenging operations insert and replace. Our studies suggest that when synthesized words are inserted in the context of a spoken sentence, the modified sentence is often perceived as indistinguishable from other sentences spoken in the same voice. With alternative syntheses and manual editing, the "authenticity" of the synthesized word is further strengthened. Moreover, even in cases where subjects can recognize the sentence as edited, it would still have acceptable quality for many applications. In cases where a new word is desired (or needed), the only alternative approaches are to re-record the audio, or to recruit an expert audio engineer to painstakingly assemble the word (more than 20 minutes per word on average, according to our study, to yield words that were of inferior quality to those produced by VoCo). Therefore, we believe the proposed system offers a valuable improvement over the state of the art.

Nevertheless, there are several limitations in this approach that suggest areas for future work. First, our ultimate goal is to synthesize words with such consistently high quality that they will be rarely or never recognizable as edited. Our approach relies on MFCC and pitch features for determining compatible areas of waveform to

[1]http://www.audacityteam.org/
[2]http://www.adobe.com/products/audition.html

be stitched together. We believe the factor that could improve the quality of our synthesized words would be to design or find features that were optimized specifically for our problem.

Second, our method relies on a hand-crafted energy function to select an "optimal" synthesis result, but this function is not well correlated with human perception. We believe it may be possible to design a more suitable energy function for our problem via a data-driven approach that may even be voice-specific.

Finally, our method can currently only synthesize a word or short phrase. This limitation mainly stems from the challenge of synthesizing a voice with natural prosody over longer sequences of words. Since the prosody of generic TTS systems is generally unnatural, our approach inherits this defect. One idea we would like to investigate is the use of the audio editor's voice as the query, instead of the generic TTS voice. The editor could enunciate the query with a particular prosody, and this would guide the synthesized result. The main challenge is that in this scenario there is no aligned synthetic voice to guide the voice conversion process towards the target voice. However it may be possible by applying many-to-one voice conversion to the editor's voice [Toda et al. 2007b]. An alternate approach might build on the recent WaveNet work of van den Oord et al. [2016] which shows that deep learning can effectively model human prosody by learning directly from audio samples.

A DETAILS OF THE RANGE SELECTION ALGORITHM

Range Selection (Section 4.4) aims to find a small set of consecutive frames (*ranges*) that are similar to the query and then combine them together to obtain the final synthesis result. In this section, we present the algorithmic details and more precise mathematical definitions about *Range Selection*. To recap, a range , denoted as $\langle s, t\rangle$, is defined as a sequence of frames starting from index s to t. The j-th candidate segment of phoneme i is $R_{ij} = \langle b_{ij}, e_{ij}\rangle$. Our objective is to select ranges $\mathcal{R}_{ik} = \langle s_{ik}, t_{ik}\rangle$ that minimizes Equation 1. We denote the collection of ranges selected for i-th phoneme as $\mathcal{R}_i = \{\mathcal{R}_{ik} \mid k \in (1, K_i)\}$, assuming there are K_i ranges selected for that phoneme ($K_i \in \{1, 2\}$).

In Equation 1, the concatenation costs C_i and $\mathcal{D}_i$ are defined as follows:

$$C_i = \begin{cases} 0 & \text{if } K_i = 1 \\ C(t_{i1}, s_{i2}) & \text{if } K_i = 2 \end{cases} \qquad \mathcal{D}_i = C(t_{i-1,K_{i-1}}, s_{i1})$$

where function $C(t, s)$ represents the distortion transitioning from Frame t to Frame s. In this paper, we define $C(t, s)$ as the Euclidean distance between the exemplar feature of Frame t to that of Frame s. For other cost functions in Equation 1, we define similarity cost S as Mel-Cepstral Distortion and the duration cost $L(r_1, r_2)$ as the log of the ratio between the length of r_1 and the length of r_2.

Because we limit the number of *ranges* per phoneme to be at most two, there are two types for each phoneme (1) choose two ranges, one starts from phoneme boundary (we call it `Pre`) and the other ends at a phoneme boundary (we call it `Post`). (2) choose only 1 range, starting and ending in the same phoneme segment; we call its starting point `Start` and ending point `End`.

This allows us to use dynamic programming to solve the general optimization defined in equation 1 efficiently. Let n_i be the number of candidates for phoneme i. For a segment $\langle b_{ij}, e_{ij}\rangle_{j=1\ldots n_i}$, the only valid *ranges* set for each query phoneme i should belong to one of the following two sets: $\texttt{Pre}(i) \times \texttt{Post}(i) \equiv \{\{r_{pre}, r_{post}\} | r_{pre} \in \texttt{Pre}(i), r_{post} \in \texttt{Post}(i)\}$ and $\texttt{Start}(i) \otimes \texttt{End}(i) \equiv \{< b, e > | j \in [n_i], b \in \texttt{Start}(i,j), e \in \texttt{End}(i,j), b < e\}$ where,

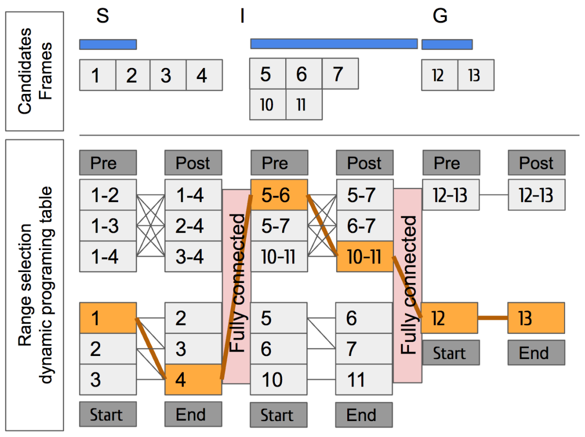

Fig. 15. Range selection algorithm illustrated. Above: the frames selected by dynamic triphone preselection, numbered by frame indices. Below: the dynamic programming table, where each box shows the frame-range it represents; the connections between boxes show dynamic programming dependencies. The orange boxes and connections show an example solution, wherein frame ranges 1-4, 5-6, 10-11 and 12-13 are selected.

$$\texttt{Pre}(i) = \{\langle b_{ij}, b_{ij} + k\rangle | j \in [n_i], k \in [e_{ij} - b_{ij}]\}$$
$$\texttt{Post}(i) = \{\langle b_{ij} + k, e_{ij}\rangle | j \in [n_i], k \in [e_{ij} - b_{ij}]\}$$
$$\texttt{Start}(i,j) = \{b_{ij}, b_{ij} + 1, \ldots, e_{ij} - 1\}$$
$$\texttt{End}(i,j) = \{b_{ij} + 1, b_{ij} + 2, \ldots, e_{ij}\}$$
$$\texttt{Start}(i) = \bigcup_{j=1}^{n_i} \texttt{Start}(i,j), \quad \texttt{End}(i) = \bigcup_{j=1}^{n_i} \texttt{End}(i,j)$$

For example, if a segment contains frame 1 through 3, then the above four sets are: $\texttt{Pre} = \{\langle 1,2\rangle, \langle 1,3\rangle\}$, $\texttt{Post} = \{\langle 1,3\rangle, \langle 2,3\rangle\}$, $\texttt{Start} = \{1,2\}$ and $\texttt{End} = \{2,3\}$. And the valid *ranges* are: $\texttt{Pre} \times \texttt{Post} = \{\{\langle 1,2\rangle, \langle 1,3\rangle\}, \{\langle 1,2\rangle, \langle 2,3\rangle\}, \{\langle 1,3\rangle, \langle 1,3\rangle\}, \{\langle 1,3\rangle, \langle 2,3\rangle\}\}$ and $\texttt{Start} \otimes \texttt{End} = \{\{\langle 1,2\rangle\}, \{\langle 1,3\rangle\}, \{\langle 2,3\rangle\}\}$. Because we are looking at one phoneme here ($i = 1$), we omit i from the equation. We can assume these sets are ordered for simplicity.

Now the task becomes selecting a subset of these valid *ranges* to minimize the objective function. First we prepare a minimal cost table: for each phoneme i, we create a note for each of the elements in the above 4 sets, $\texttt{Pre}(i)$, $\texttt{Post}(i)$, $\{\texttt{Start}(i,j)\}_j$ and $\{\texttt{End}(i,j)\}_j$. Let $F(\texttt{Pre}, i, j)$ be the corresponding frames of j-th element in set $\texttt{Pre}(i)$ and $M_{\texttt{pre},i,j}$ be its minimal cost. We do the same for `Post`, `Start` and `End`. Then we can derive the following dynamic programming algorithm that selects a transition path through the table that combines *ranges* to obtain a minimal cost M:

For `Start` and `Pre`, their preceding *ranges* are always from the previous phoneme. Therefore, the minimal cost M is defined to be the smallest of all their preceding *ranges*' minimal costs M plus concatenation cost C. If the *ranges* are consecutive, the concatenation cost is 0. Here is the complete mathematical definition.

$$
\begin{aligned}
M_{\texttt{Pre},i,j} = \min(& \\
& \min_t\{M_{\texttt{Post},i-1,t} + C(F_{\texttt{Post},i-1,t}, F_{\texttt{Pre},i,j})\}, \\
& \min_t\{M_{\texttt{End},i-1,t} + C(F_{\texttt{End},i-1,t}, F_{\texttt{Pre},i,j})\}) \\
M_{\texttt{Start},i,j} = \min(& \\
& \min_t\{M_{\texttt{Post},i-1,t} + C(F_{\texttt{Post},i-1,t}, F_{\texttt{Start},i,j})\}, \\
& \min_t\{M_{\texttt{End},i-1,t} + C(F_{\texttt{post},i-1,t}, F_{\texttt{Start},i,j})\})
\end{aligned}
$$

For each `Post` *range*, when combined with a preceding `Pre` *range*, they form a valid range choice for the current phoneme. Then we can use the corresponding frames to determine the similarity cost and the duration cost. If `Pre` and `Post` are not consecutive, then there will be a concatenation cost. Therefore,

$$
\begin{aligned}
M_{\texttt{Post},i,j} = \min_t\{M_{\texttt{Pre},i,t} & + \alpha S(q_i, \{F_{\texttt{Pre},i,t}, F_{\texttt{Post},i,j}\}) \\
& + \beta L(q_i, \{F_{\texttt{Pre},i,t}, F_{\texttt{Post},i,j}\}) \\
& + C(F_{\texttt{Pre},i,t}, F_{\texttt{Post},i,j})\}
\end{aligned}
$$

in which the definition of α, β, S and L follows the same definition used in Equation 1. Similarly, combining an `End` *range* with a `Start` *range* forms a valid *range* choice that defines the similarity and duration cost. Since there is only one *range* selected, there is no concatenation cost. Therefore,

$$
\begin{aligned}
M_{\texttt{End},i,j} = \min_t\{M_{\texttt{Start},i,t} & + \alpha S(q_i, \{\langle F_{\texttt{Start},i,t}, F_{\texttt{End},i,j}\rangle\}) \\
& + \beta L(q_i, \{\langle F_{\texttt{Start},i,t}, F_{\texttt{End},i,j}\rangle\})\}
\end{aligned}
$$

Using back trace, we can extract the selected *ranges* that produce the minimal cost. Figure 15 shows an example selection result. When combined, the final selected frames form two consecutive: segments: 1-6 and 10-13.

Range Selection is more efficient than Unit selection. Suppose each query phoneme lasts m frames and has n candidates. Unit selection has a complexity of $O(mn^2)$ per phoneme with the Viterbi algorithm. In *Range selection*, however, the list of candidates per phoneme contain $O(n)$ rows and only 2 columns (Figure 15). Therefore, the complexity of *Range selection* per phoneme is $O(n^2)$. Because of the efficiency, we can mass produce selection results for all possible values of α and β, which leads to alternative Syntheses (Section 4.5).

REFERENCES

Acapela Group. 2016. http://www.acapela-group.com. (2016). Accessed: 2016-04-10.

Ryo Aihara, Toru Nakashika, Tetsuya Takiguchi, and Yasuo Ariki. 2014. Voice conversion based on Non-negative matrix factorization using phoneme-categorized dictionary. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2014)*.

Floraine Berthouzoz, Wilmot Li, and Maneesh Agrawala. 2012. Tools for placing cuts and transitions in interview video. *ACM Trans. on Graphics (TOG)* 31, 4 (2012), 67.

Paulus Petrus Gerardus Boersma et al. 2002. Praat, a system for doing phonetics by computer. *Glot international* 5 (2002).

Christoph Bregler, Michele Covell, and Malcolm Slaney. 1997. Video Rewrite: Driving Visual Speech with Audio. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '97)*. 353–360.

Juan Casares, A Chris Long, Brad A Myers, Rishi Bhatnagar, Scott M Stevens, Laura Dabbish, Dan Yocum, and Albert Corbett. 2002. Simplifying video editing using metadata. In *Proceedings of the 4th conference on Designing interactive systems: processes, practices, methods, and techniques*. ACM, 157–166.

Ling-Hui Chen, Zhen-Hua Ling, Li-Juan Liu, and Li-Rong Dai. 2014. Voice conversion using deep neural networks with layer-wise generative training. *Audio, Speech, and Language Processing, IEEE/ACM Transactions on* 22, 12 (2014), 1859–1872.

Alistair D Conkie and Stephen Isard. 1997. Optimal coupling of diphones. In *Progress in speech synthesis*. Springer, 293–304.

Srinivas Desai, E Veera Raghavendra, B Yegnanarayana, Alan W Black, and Kishore Prahallad. 2009. Voice conversion using Artificial Neural Networks. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2009)*.

Thierry Dutoit, Andre Holzapfel, Matthieu Jottrand, Alexis Moinet, J Prez, and Yannis Stylianou. 2007. Towards a Voice Conversion System Based on Frame Selection. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2007)*.

G David Forney. 1973. The Viterbi algorithm. *Proc. IEEE* 61, 3 (1973), 268–278.

Kei Fujii, Jun Okawa, and Kaori Suigetsu. 2007. High-Individuality Voice Conversion Based on Concatenative Speech Synthesis. *International Journal of Electrical, Computer, Energetic, Electronic and Communication Engineering* 1, 11 (2007), 1617 – 1622.

François G. Germain, Gautham J. Mysore, and Takako Fujioka. 2016. Equalization Matching of Speech Recordings in Real-World Environments. In *41st IEEE International Conference on Acoustics Speech and Signal Processing (ICASSP 2016)*.

Aaron Hertzmann, Charles E Jacobs, Nuria Oliver, Brian Curless, and David H Salesin. 2001. Image analogies. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. ACM, 327–340.

Andrew J Hunt and Alan W Black. 1996. Unit selection in a concatenative speech synthesis system using a large speech database. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 1996)*. 373–376.

Zeyu Jin, Adam Finkelstein, Stephen DiVerdi, Jingwan Lu, and Gautham J. Mysore. 2016. CUTE: a concatenative method for voice conversion using exemplar-based unit selection. In *41st IEEE International Conference on Acoustics Speech and Signal Processing (ICASSP 2016)*.

Alexander Kain and Michael W Macon. 1998. Spectral voice conversion for text-to-speech synthesis. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 1998)*. 285–288.

Hideki Kawahara, Masanori Morise, Toru Takahashi, Ryuichi Nisimura, Toshio Irino, and Hideki Banno. 2008. TANDEM-STRAIGHT: A temporally stable power spectral representation for periodic signals and applications to interference-free spectrum, F0, and aperiodicity estimation. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2008)*. 3933–3936.

John Kominek and Alan W Black. 2004. The CMU Arctic speech databases. In *Fifth ISCA Workshop on Speech Synthesis*.

Robert F. Kubichek. 1993. Mel-cepstral distance measure for objective speech quality assessment. In *Proceedings of IEEE Pacific Rim Conference on Communications Computers and Signal Processing*. 125–128.

Sergey Levine, Christian Theobalt, and Vladlen Koltun. 2009. Real-time Prosody-driven Synthesis of Body Language. *ACM Trans. Graph.* 28, 5, Article 172 (Dec. 2009), 10 pages.

Jingwan Lu, Fisher Yu, Adam Finkelstein, and Stephen DiVerdi. 2012. HelpingHand: Example-based Stroke Stylization. *ACM Trans. Graph.* 31, 4, Article 46 (July 2012), 10 pages.

Michal Lukáč, Jakub Fišer, Jean-Charles Bazin, Ondřej Jamriška, Alexander Sorkine-Hornung, and Daniel Sýkora. 2013. Painting by Feature: Texture Boundaries for Example-based Image Creation. *ACM Trans. Graph.* 32, 4, Article 116 (July 2013), 8 pages.

Anderson F Machado and Marcelo Queiroz. 2010. Voice conversion: A critical survey. *Proc. Sound and Music Computing (SMC)* (2010), 1–8.

Lindasalwa Muda, Mumtaj Begam, and Irraivan Elamvazuthi. 2010. Voice recognition algorithms using mel frequency cepstral coefficient (MFCC) and dynamic time warping (DTW) techniques. *arXiv preprint arXiv:1003.4083* (2010).

Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. 2016. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499* (2016).

Amy Pavel, Dan B. Goldman, Björn Hartmann, and Maneesh Agrawala. 2015. SceneSkim: Searching and Browsing Movies Using Synchronized Captions, Scripts and Plot Summaries. In *Proceedings of the 28th annual ACM symposium on User interface software and technology (UIST 2015)*. 181–190.

Amy Pavel, Björn Hartmann, and Maneesh Agrawala. 2014. Video digests: A browsable, skimmable format for informational lecture videos. In *Proceedings of the 27th annual ACM symposium on User interface software and technology (UIST 2014)*. 573–582.

Bhiksha Raj, Tuomas Virtanen, Sourish Chaudhuri, and Rita Singh. 2010. Non-negative matrix factorization based compensation of music for automatic speech recognition. In *Interspeech 2010*. 717–720.

Marc Roelands and Werner Verhelst. 1993. Waveform similarity based overlap-add (WSOLA) for time-scale modification of speech: structures and evaluation. In *EUROSPEECH 1993*. 337–340.

Steve Rubin, Floraine Berthouzoz, Gautham J. Mysore, Wilmot Li, and Maneesh Agrawala. 2013. Content-based Tools for Editing Audio Stories. In *Proceedings of the 26th Annual ACM Symposium on User Interface Software and Technology (UIST 2013)*. 113–122.

Kåre Sjölander. 2003. An HMM-based system for automatic segmentation and alignment of speech. In *Proceedings of Fonetik 2003*. 93–96.

Matthew Stone, Doug DeCarlo, Insuk Oh, Christian Rodriguez, Adrian Stere, Alyssa Lees, and Chris Bregler. 2004. Speaking with Hands: Creating Animated Conversational Characters from Recordings of Human Performance. *ACM Trans. Graph.* 23, 3 (Aug. 2004), 506–513.

Yannis Stylianou, Olivier Cappé, and Eric Moulines. 1998. Continuous probabilistic transform for voice conversion. *IEEE Transactions on Speech and Audio Processing* 6, 2 (1998), 131–142.

Paul Taylor. 2009. *Text-to-Speech Synthesis*. Cambridge University Press.

Tomoki Toda, Alan W Black, and Keiichi Tokuda. 2007a. Voice Conversion Based on Maximum-Likelihood Estimation of Spectral Parameter Trajectory. *IEEE Transactions on Audio, Speech, and Language Processing* 15, 8 (2007), 2222–2235.

Tomoki Toda, Yamato Ohtani, and Kiyohiro Shikano. 2007b. One-to-many and many-to-one voice conversion based on eigenvoices. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2007)*. IV–1249.

Tomoki Toda, Hiroshi Saruwatari, and Kiyohiro Shikano. 2001. Voice conversion algorithm based on Gaussian mixture model with dynamic frequency warping of STRAIGHT spectrum. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2001)*. 841–844.

Keiichi Tokuda, Yoshihiko Nankaku, Tomoki Toda, Heiga Zen, Junichi Yamagishi, and Keiichiro Oura. 2013. Speech Synthesis Based on Hidden Markov Models. *Proc. IEEE* 101, 5 (May 2013), 1234–1252.

Steve Whittaker and Brian Amento. 2004. Semantic Speech Editing. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI 2004)*. 527–534.

Zhizheng Wu, Tuomas Virtanen, Tomi Kinnunen, Engsiong Chng, and Haizhou Li. 2013. Exemplar-based unit selection for voice conversion utilizing temporal information. In *INTERSPEECH 2013*. 3057–3061.