TTSBOX 1.0 DOCUMENTATION

A Matlab toolbox for teaching Text-to-Speech synthesis

04/08/2004

Faculté Polytechnique de Mons

Thierry Dutoit
Faculté Polytechnique de Mons
TCTS Lab

Ave. Copernic	Ph:	+32 65 374774
Parc Initialis	Fax:	+32 65 374729
B-7000 Mons	Thierry.Dutoit at fpms.ac.be	
Belgium	http://tcts.fpms.ac.be/~dutoit	

TTSBOX 1.0 DOCUMENTATION

Prof. T. Dutoit, July 2004.

0. What is TTSBOX ?

As the Chinese proverb says : "Tell me and I'll forget. Show me and I'll remember. But involve me and I'll understand."

Text-to-Speech synthesis, however, is a complex combination of language processing, signal processing, and computer science. Students are therefore usually introduced to it in a top-down approach, emphasising problems to be solved and introducing solutions on paper, but with little real practice : designing a TTS takes too much time, and modifying one is usually impossible if you did not take part in its design (yet only if it was correctly documented). Apart from the FESTIVAL TTS system, which uses SCHEME as an interactive language for letting students play with TTS basics, no real "hands on" toolbox was available, especially for engineering students (who are most often familiar with MATLAB).

TTSBOX is a Matlab toolbox for teaching Text-to-Speech synthesis to undergraduate and graduate students. It was designed with the hope that it can help to increase the personal involvment of students in their TTS courses. I imagined it when teaching TTS in the EPFL post-graduate course in computer science "Language and Speech Engineering", and later involved my own graduate students at FPMs (Belgium) in its design.

TTSBOX performs the synthesis of Genglish (for "Generic English"), an imaginary language obtained by replacing English words by generic words. Genglish therefore has a rather limited lexicon, but its pronunciation maintains most of the problems encountered in natural languages. TTSBOX uses simple data-driven techniques (Bigrams, CARTs, NUUs) while trying to keep the code minimal, so as to keep it readable for students with reasonable MATLAB practice.

1. Synthesizing Genglish

Genglish is defined here as “English in which all the words belonging to open classes (i.e. classes of words whose number of elements is constantly expanding : verbs, nouns, adjectives, and adjectival adverbs) are replaced by a generic substitute” (see Table 1). Apart from these substitutions, the syntax and pronounciation of Genglish is assumed to be that of English. It will also be assumed, in order to avoid the need for a preprocessor, that Genglish has no abreviations, no arabic nor roman numbers, and no acronyms.

Table 1 Simple English-to-Genglish translation[1]

English	**Genglish**
Verb forms (except auxiliary forms of "be", "have", and "do")	Forms of "to gengle" [dʒɛŋgl]
Auxiliary forms of "be" and "have"	"is", "are", or "be"
Auxiliary forms of "do" , "can" and "must"	(omitted)
Adjectives, ordinals	"genglish" [gɛŋglɪʃ]
Nouns	"gengle" or "gengles" [gɛŋgl]
Adjectival adverbs	"gengly" [gɛŋglɪ]
Degree adverbs	(omitted)
Negation adverbs	(omitted)
Acronyms, proper names	"John"
Coordinators	"and"
Subordinators	"since"
Determiners, Numerals	"the"
Prepositions (except "of" and "to" + infinitive)	"on"
"of"	"of"
"to" (introducing an infinitive)	"to"
Pronouns (other than relative)	"it"
Relative pronouns	"which"

It is quite easy, for a human reader, to translate English into Genglish. The quotation at the beginning of this document, for instance, would look like this in Genglish:

> "[Gengling on the genglish gengle of John gengle] It is gengle gengle; gengle gengle is the gengle of gengling the gengle on the gengle…"

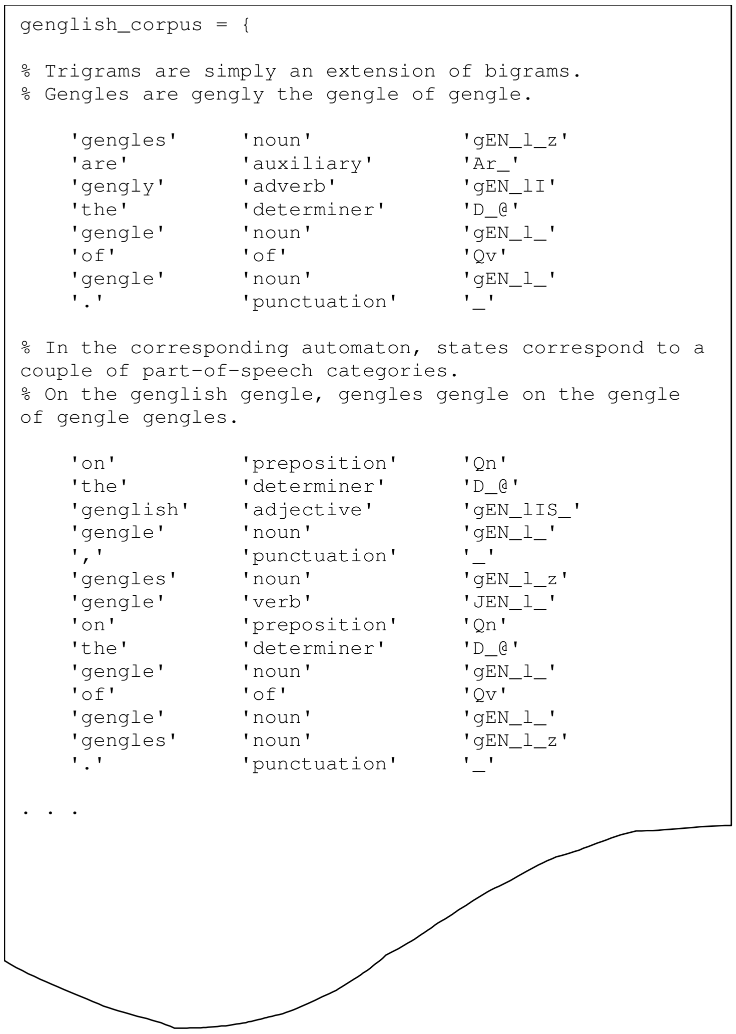
For a deeper examination of Genglish sentences, we have created a MATLAB corpus file, `genglish_load_corpus.m`, containing `genglish_corpus`, a set of 50 Genglish sentences (about 800 words) in which each word is listed with its spelling, part-of-speech category, and phonetization (Fig. 1). This file also contains a smaller test corpus, `genglish_test_corpus` (24 sentences; about 400 words), which wil be used later. Obtaining the first 10 words of `genglish_corpus` is easy:

```
» genglish_load_corpus;
» genglish_corpus(1:10,:)
ans =
    'gengles'     'noun'           'gEN_l_z'
```

[1] This is only a temptative description of Engligh-to-Genglish correspondance. English sentences whose words are not covered by this table should be … avoided.

```
'are'           'auxiliary'       'a__'
'gengly'        'adverb'          'gEN_lI'
'the'           'determiner'      'D_@'
'gengle'        'noun'            'gEN_l_'
'of'            'of'              'Qv'
'gengle'        'noun'            'gEN_l_'
'.'             'punctuation'     '_'
'on'            'preposition'     'Qn'
'the'           'determiner'      'D_@'
```

As we see, although Genglish is lexically much simpler than English (and practically looses most of its semantics), it maintains a lot of its phonetic, syntactic, and prosodic complexity. In particular, Genglish is lexically ambiguous (even more than English), since verbs and nouns can have the same spelling (but different pronunciations). Genglish is much simpler than English, in that it is a language with a *closed* lexicon: the list of its words is limited, once and for all. In contrast, the set of Genglish sentences is infinite. This will make our TTS design task much simpler, while keeping one of the major challenges of the synthesis of *open* lexicon languages : that of naturalness. At the end of the document, we provide a list of pointers to techniques for handling open languages as well.



```
genglish_corpus = {

% Trigrams are simply an extension of bigrams.
% Gengles are gengly the gengle of gengle.

    'gengles'     'noun'            'gEN_l_z'
    'are'         'auxiliary'       'Ar_'
    'gengly'      'adverb'          'gEN_lI'
    'the'         'determiner'      'D_@'
    'gengle'      'noun'            'gEN_l_'
    'of'          'of'              'Qv'
    'gengle'      'noun'            'gEN_l_'
    '.'           'punctuation'     '_'

% In the corresponding automaton, states correspond to a
couple of part-of-speech categories.
% On the genglish gengle, gengles gengle on the gengle
of gengle gengles.

    'on'          'preposition'     'Qn'
    'the'         'determiner'      'D_@'
    'genglish'    'adjective'       'gEN_lIS_'
    'gengle'      'noun'            'gEN_l_'
    ','           'punctuation'     '_'
    'gengles'     'noun'            'gEN_l_z'
    'gengle'      'verb'            'JEN_l_'
    'on'          'preposition'     'Qn'
    'the'         'determiner'      'D_@'
    'gengle'      'noun'            'gEN_l_'
    'of'          'of'              'Qv'
    'gengle'      'noun'            'gEN_l_'
    'gengles'     'noun'            'gEN_l_z'
    '.'           'punctuation'     '_'

. . .
```

Fig. 1 The Genglish corpus file (exerpt).

2. MORPHOSYNTACTIC ANALYSIS

As will be shown in Section 3 and 4, it is often impossible to correctly pronounce a sequence of words in natural languages without prior knowledge of their part-of-speech, as well as of their hierarchical organization into groups, which itself also depends on the sequence of part-of-speech involved.

Part-of-speech information can simply be obtained from a lexicon in many cases, but there are a large number of words (most of them frequently used), which can have distinct part-of-speech, depending the context in which they are used (think of "*record", "permit*", "*present", "answer*", or "*kind*", in English, and "*gengle*" in Genglish).

The morpho-syntactic module of a TTS system is therefore usually composed of :

- A morphological analysis module, responsible for proposing all possible part of speech categories for each word taken individually, on the basis of its spelling alone.
- A contextual analysis module considering words in their context. This module typically chooses, from the list of possible part-of-speech categories for each word, the one it is most likely to have it the given lexical context.
- Finally, a syntactic-prosodic parser, which finds the hierarchical organization of words into clause and phrase-like constituents that more closely relates to its expected intonational structure (see 2.3 for more details).

2.1 Genglish Pre-processing

Genglish contains no numbers, no abbreviations, no acronyms, no complicated proper names, and no unknown words. What is more, Genglish writers make no spelling mistake. This makes our first Genglish module an model of simplicity : the only task it has to perform is to segment incoming sentences into *tokens* (words and punctuations). We easily do this using the simple finite state machine embodied in the MATLAB `strtok` function, after making sure punctuations are not considered as separate tokens. The resulting function, `tts_preprocess_using_fsm`, is straightforward :

```
» tts_preprocess_using_fsm('Gengles are gengly the gengle of genle.')
ans =
    'gengles'
    'are'
    'gengly'
    'the'
    'gengle'
    'of'
    'genle'
    '.'
```

More information on real pre-processing issues can be found in [Dutoit 97, section 4.1]) or in [Sproat 98, chapter 3].

2.2. Morphological analysis of Genglish

Since Genglish is, by definition, a closed language, the possible part-of-speech categories of its words can advantageously be described in terms of a morphological *lexicon*, which provides a list of al words, associated with their possible part-of-speech categories.

We therefore analyze our Genglish corpus to derive its morphological lexicon (Fig. 2).

This is done with a simple MATLAB script, `corpus_to_lexicons.m` (which creates other lexicons at the same time; see later). The resulting MATLAB variable, `genglish_morphlex`, is a cell array[2] of all possible Genglish words and their possible part-of-speech categories[3] :

```
» genglish_load_corpus;
» [genglish_morph_lex,genglish_pos_lex, genglish_graph_lex,
  genglish_phon_lex]=corpus_to_lexicons(genglish_corpus);
» genglish_morph_lex
genglish_morph_lex =
    ','           {1x1 cell}
    '.'           {1x1 cell}
    'and'         {1x1 cell}
    'are'         {1x1 cell}
    'be'          {1x1 cell}
    'gengle'      {2x1 cell}
    'gengled'     {2x1 cell}
    'gengles'     {2x1 cell}
    'gengling'    {1x1 cell}
    'genglish'    {1x1 cell}
    'gengly'      {1x1 cell}
    'is'          {1x1 cell}
    'it'          {1x1 cell}
    'john'        {1x1 cell}
    'of'          {1x1 cell}
    'on'          {1x1 cell}
    'since'       {1x1 cell}
    'the'         {1x1 cell}
    'to'          {1x1 cell}
    'which'       {1x1 cell}
```

```
genglish_morph_lex ={
',', {'punctuation'}
'.', {'punctuation'}
'John', {'propername'}
'and', {'coordinator'}
'are',{'auxiliary'}
'be',{'auxiliary'}
'gengle',{'verb'; 'noun'}
'gengled',{'verb'; 'participle'}
'gengles',{'verb'; 'noun'}
'gengling',{'participle'}
'genglish',{'adjective'}
'gengly', {'adverb'}
'is',{'auxiliary'}
'it', {'pronoun'}
'of', {'of'}
'on', {'preposition'}
'since', {'subordinator'}
'the', {'determiner'}
'to', {'to'}
'which', {'pronoun'}
};
```

Fig. 2 The (expanded) contents of the morphological lexicon of Genglish.

[2] In Matlab, it is easy to create arrays of items of various types (including arrays of arrays), by using curly braces '{}' in a straightforward way to index items (instead of the traditional '[]' for monotype arrays). Such curly braces-based indexed item are stored into Matlab *cells.*

[3] In this simple tutorial, we do not distinguish tenses, gender, or number in the list of part-of-speech categories, in order not to increase the number of possible categories.

As expected, the words *'gengle, 'gengles'*, and *'gengled'* get two possible tags. For *'gengle'*, for instance :

```
» genglish_morph_lex{7,2}
ans =
    'noun'    'verb'
```

`Corpus_to_lexicons` also provides the set of part-of-speech categories found in the corpus :

```
» genglish_pos_lex
genglish_pos_lex =
    'adjective'
    'adverb'
    'auxiliary'
    'coordinator'
    'determiner'
    'noun'
    'of'
    'participle'
    'preposition'
    'pronoun'
    'propername'
    'punctuation'
    'subordinator'
    'to'
    'verb'
```

Notice the number of elements in the second column of this lexicon results from the tags we have stored in our corpus, which was our own decision. The more categories we distinguish, the more information we will have later for phonetization and syntactic-prosodic grouping; on the other hand, the design of contextual analysis module will be harder.

We then create a MATLAB function implementing simple, functional (but by far non-optimal[4]) lexicon search, `lexicon_search.m`[5]. Finding the possible part_of_speech categories for the word *'gengled'*, for instance, is obtained by:

```
» pos=lexicon_search('gengled',genglish_morph_lex)
pos =
    'participle'   'verb'
```

This function is used by `tts_morph_using_lexicon.m` to obtain the possible part-of-speech categories for all the words in a sentence:

```
» pos_list=tts_morph_using_lexicon({'it';'gengled'},genglish_morph_lex)
pos_list =
```

[4] Fast lexical access is usually achieved by use of *hash-tables* or *tries* [Knuth 73].

[5] For clarity, we follow simple naming conventions for matlab script files :

1. No use of uppercase characters
2. The name of the files which are not related to a particular language always start with `tts_`; language-dependent files start with the name of the language (`genglish_` in our case)
3. Files names explicitely mention what the script does. This leads to long file names, but makes their use more comprehensive.

```
    {1x1 cell}    {1x2 cell}
» pos_list{:,:}   % forces MATLAB to output the content of all cells
ans =
    'pronoun'
ans =
    'participle'    'verb'
```

2.3 Contextual analysis of Genglish

We now want to be able to associate to each word the part-of-speech category it has *in the context in which it appears*. This obviously implies to make a decision among the possible part-of-speech categories proposed by the previous module for each word. One standard way of doing this is by using *n-grams.*

N-grams first appeared in the context of continuous speech recognition to estimate the probability of a sequence of words w_1, w_2, . . ., w_N in a given language. They are now widely used in computational linguistics, speech synthesis, and speech recognition because they combine simplicity and efficiency (see [Kupiec 92] for instance). As [Jelinek 93] points out: "That this simple approach is so successful is a source of considerable irritation to me and to some of my colleagues. We have evidence that better language models are obtainable, we think we know many weaknesses of the trigram model, and yet, when we devise more or less subtle methods of improvement, we come up short."

In the context of tagging a given sentence $\boldsymbol{W}$=(w_1, w_2, ..., w_N), we are looking for the best sequence of tags $\hat{\boldsymbol{T}}$ among all the sequences $\boldsymbol{T}$=(t_1, t_2, ..., t_N) chosen in the set of admissible tags {c_1, c_2, ..., c_M }:

$$\hat{\boldsymbol{T}} = \arg\max_{\boldsymbol{T}} P(\boldsymbol{T}|\boldsymbol{W}) \tag{1}$$

By Bayes's rule, this is equivalent to finding

$$\hat{\boldsymbol{T}} = \arg\max_{\boldsymbol{T}} \frac{P(\boldsymbol{T},\boldsymbol{W})}{P(\boldsymbol{W})} = \arg\max_{\boldsymbol{T}} \frac{P(\boldsymbol{W}|\boldsymbol{T})P(\boldsymbol{T})}{P(\boldsymbol{W})} \tag{2}$$

The denominator of (2) in clearly independent of $\boldsymbol{T}$ and can be ignored in the search for $\hat{\boldsymbol{T}}$.

The *n*-gram model for tagging simply makes the following assumptions (clearly inexact, but it turns out that they are quite useful):

1. The probability of a word given the past mostly depends on its tag.
2. The probability of a tag given the past mostly depends on the last *n-1* tags.

As a result:

$$\begin{aligned}
P(\boldsymbol{W}|\boldsymbol{T}) &= P(w_1,w_2,\ldots,w_N \mid t_1,t_2,\ldots,t_N) \\
&= P(w_1 \mid t_1,t_2,\ldots,t_N)P(w_2 \mid w_1,t_1,t_2,\ldots,t_N)\ldots P(w_N \mid w_1,\ldots,w_{N-1},t_1,t_2,\ldots,t_N) \\
&\approx \prod_{i=1}^{N} P(w_i \mid t_i) \\
P(\boldsymbol{T}) &= P(t_1,t_2,\ldots,t_N) \\
&= P(t_1)P(t_2|t_1)P(t_3|t_2,t_1)\ldots P(t_N|t_1,t_2,\ldots,t_{N-1}) \\
&\approx \prod_{i=1}^{N} P(t_i \mid t_{i-1},t_{i-2},\ldots,t_{i-n+1})
\end{aligned} \tag{3}$$

Hence

$$P(t_1,t_2,\ldots,t_N|w_1,w_2,\ldots,w_N) = \prod_{i=1}^{N} P(w_i \mid t_i)\ P(t_i \mid t_{i-1},\ldots,t_{i-n+1}) \tag{4}$$

It is then straightforward to see the problem in terms of a finite state automaton. In a bigram model, for example, *n* in (4) is set to one (i.e., it is assumed that the probability of a tag only depends on the previous tag). It is then easy to sketch a bigram, using a set of states which simply represent the part-of-speech categories considered by the grammar (one state per category). Each transition is associated with a *transition probability* P(c_i|c_j) (from state *j* to state *i*), which is the probability for a word of category c_j to be followed by a word of category c_i. If one assumes that the vocabulary is finite (as is the case for Genglish) with *L* the number of elements in its vocabulary, one can define, for each state and each word in the vocabulary, a state-dependent *emission probability* P(w_i|c_j), which represents the probability that category c_j appears as word w_i.

An example is given in Fig. 3, for a possible bigram automaton of Genglish. Emission probabilities are given in text boxes attached to states. In this particular case, a large number of emission probabilities have zero value (and are therefore not mentioned in the corresponding text boxes), since all Genglish words cannot appear with all possible part-of-speech categories. Transition probiblilites are attached to arcs. As opposed to emission probabilities, most transition probabilities exist *a priori*. [6]

[6] Such a probabilistic model with a finite number of states, each capable of emitting one symbol in a finite set, is actually often called a *discrete Hidden Markov model*.

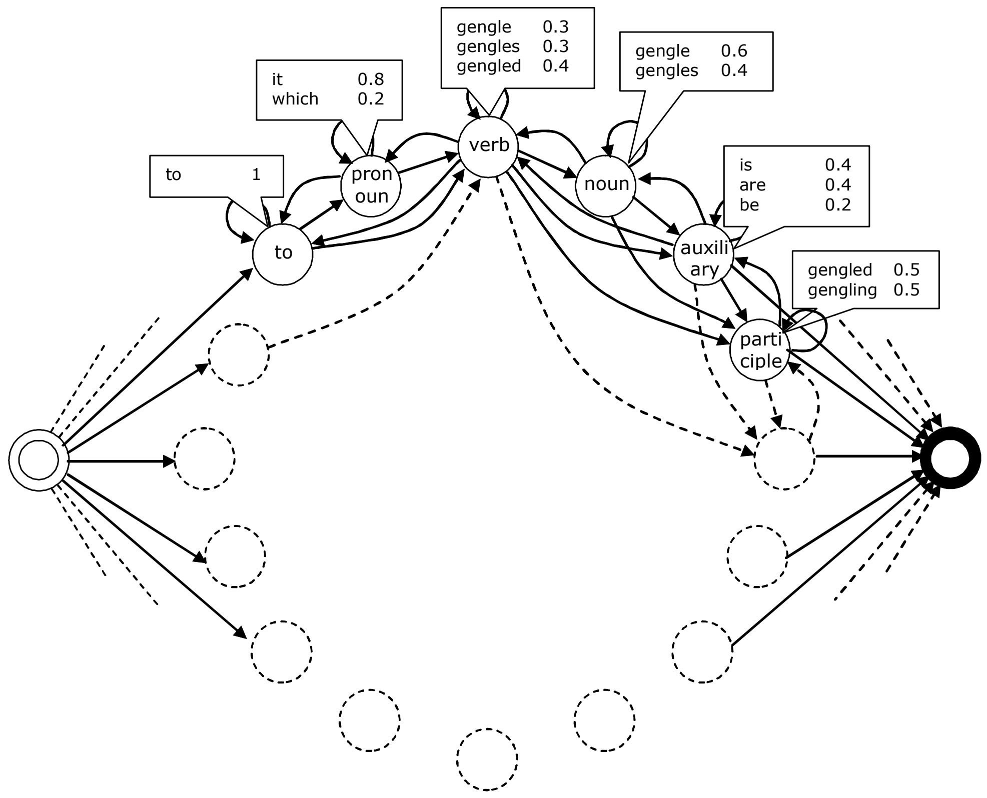


Fig. 3 A possible bigram automaton for Genglish (all states are supposed to be fully connected: only a few connections are shown; the white and black rings represent the initial and final states).

Trigrams are simply an extension of bigrams. In the corresponding automaton, states correspond to a couple of part-of-speech categories. The number of states of a trigram is therefore roughly the square of the number of states of a bigram.

Computing expression (4) requires the prior computation of all emission and transition probabilities. This can be done by counting appearings of words and tag combinations in a corpus. The corpus must be large enough for the estimates obtained by counting to be meaningful. Fortunately enough, since the vocabulary of Genglish is very small, a few pages of text are sufficient.

Computing bigram emission probabilities is easy in Genglish : the probability that category c_i emits word w_i is approximately[7] given by the number of times w_i appears as c_i , divided by the total number of words with part-of-speech category c_i :

$$P(w_i \mid c_j) \approx \frac{\#(w_i, c_j)}{\#(c_j)} \tag{5}$$

[7] Provided the corpus is large enough, the so-called *law of large numbers* allows us to estimate probabilities by counting.

Similarly, the bigram transition probabilitiy between categories c_j and c_i is approximately given by the number of times c_i appears after c_j , divided by the total number of words with part-of-speech category c_j :

$$P(c_i \mid c_j) \approx \frac{\#(c_i, c_j)}{\#(c_j)} \tag{6}$$

In order to compute these estimates on our Genglish corpus, we implement equations (5) and (6) in a MATLAB function, `corpus_to_bigrams.m`, which returns the emission and transition probabilities sketched in Fig. 3 :

```
» [emission_probs,transition_probs]=corpus_to_bigrams(genglish_corpus);
```

One can check, for example, that the last column of `emission_probs` only has three non-zero elements, which account for the fact that the last part-of_speech category of `genglish_pos_lex` (i.e., `verb`) can appear as three Genglish words : *"gengle", "gengled",* and *"gengles",* with estimated emission probabilities 0.4355, 0.1935, and 0.3710, respectively:

```
» emission_probs(:,15)
ans =
         0
         0
         0
         0
         0
    0.4792
    0.1667
    0.3542
         0
         0
         0
         0
         0
         0
         0
         0
         0
         0
         0
         0
```

Similarly, column 13 of `transition_probs` has only 3 non-zero elements, which accounts for the fact that `subordinator`, the thirteenth element of `genglish_pos_lex`, is only followed by `determiner`, `noun` and `pronoun` in the training corpus, with probabilities 0.6, 0.2, and 0.2, respectively :

```
» transition_probs(:,13)

ans =

         0
         0
         0
         0
    0.6000
    0.2000
         0
         0
         0
    0.2000
         0
         0
         0
         0
         0
```

In practice, though, one can never be sure to cover all possible cases in a corpus, however large it is. People typically address this problem by changing zeros into small, non-zero values, which will tend to restrain the algorithm from choosing very unliky paths, while avoiding the assumption of strict null probabilities. In our script we simply add 1e-8 to all probabilities[8].

Once emission and transition probabilities are estimated, obtaining the best sequence of tags for a given sentence reduces to selecting the best sequence of part-of-speech tags for the sentence, i.e., the one with highest probability (given the sequence of words and the bigram model). This corresponds to finding the best path in a lattice. As a matter of fact, while Fig. 3 shows a bigram automaton for all possible sentences of Genglish, the automaton reduces to a lattice for a given sentence (see Fig. 4).

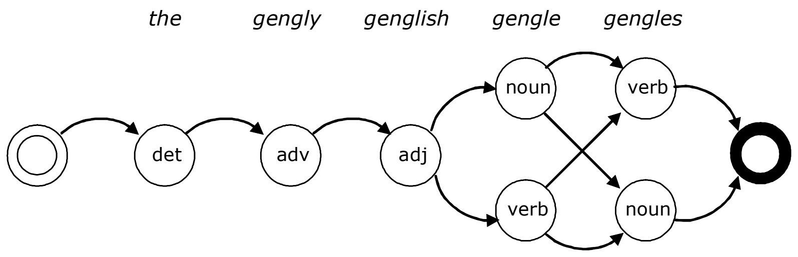


Fig. 4 An example of a lattice bigram for a simple Genglish sentence. Transition probabilities are associated to arcs. Each state is capable of emitting the word it refers to with a state-dependent emission probability.

One could obviously obtain the best path by first computing the probability of all possible sequences, and then retaining the one with highest probability. This is usually a time-consuming task[9], since the number of possible sequences of tags for a sentence is the product of the numbers of possible tags for all its words (in the simple example of Fig. 4, the number of paths is four). A more optimal approach would involve the use of the so-called Viterbi algorithm, which uses the *dynamic programming* principle (see Section 2.2 on dynamic time warping for pattern recognition). We leave this as an exercise, and rather

[8] This obviously prevents our "probabilities" from summing up to 1 (hence the numbers we have in our tables are no longer real probabilities). Many other, more accurate, means of handling zero probabilities exist. See Chen and Goodman (1998) for more information on so-called *smoothing* techniques.

create a brute-force Matlab function for finding all possible paths in a lattice, `lattice_get_all_paths.m`[10], and another file, `tts_tag_using_bigrams.m`, for using it in the context of our bigram model and obtaining the best tag sequence.

In order to check to correctness of our functions, we write a short script, `genglish_test_bigrams.m`, which runs the model on a few sentences (100 words) taken from both the training corpus and a smaller test corpus. We compute error rates (computed here as the ratio of the number of incorrect tags divided by the total number of words examined), and get 0.0190 in both cases : as expected the error rate is not zero for sentences in the training corpus (Genglish is indeed not adequatly modeled by bigrams), and the similar error rate for sentences never seen by the tagger proves its usefulness.

As a (working) exemple, let us take the Genglish equivalent of "A more optimal approach involves the use of the so-called Viterbi algorithm, which uses the dynamic programming principle.", i.e., "The gengly genglish gengle gengles the gengle of the gengled John gengle, which gengles the genglish gengle gengles.". Eight words in this sentence are morphologically ambiguous (each one having two possible tags), which gives 256 paths in the part-of-speech lattice. The correct tag sequence (`determiner adverb adjective noun verb determiner noun of determiner participle propername noun punctuation pronoun verb determiner adjective noun noun punctuation`) is correctly retrieved by the tagger :

```
» sentence={'the';'gengly';'genglish';'gengle';'gengles';'the';'gengle'
;'of';'the';'gengled';'john';'gengle';',';'which';'gengles';'the';'geng
lish';'gengle';'gengles'; '.'};
» possible_tags=tts_morph_using_lexicon(sentence,genglish_morph_lex);
» tags=tts_tag_using_bigrams(emission_probs,transition_probs,genglish_m
orphlex,genglish_pos,sentence,possible_tags)
tags =
    'determiner'
    'adverb'
    'adjective'
    'noun'
    'verb'
    'determiner'
    'noun'
    'of'
    'determiner'
    'participle'
    'propername'
    'noun'
    'punctuation'
    'pronoun'
    'verb'
    'determiner'
```

[9] In practice, after building a MATLAB function for enumerating all paths and computing their probabilities, we found the computational time became close to infinite when the number of paths was bigger than 256!

[10] This function, which is common to all discrete hidden Markov models, will also be used later, for searching for the best sequence of speech units given the sentence to synthesize.

```
'adjective'
'noun'
'noun'
'punctuation'
```

2.4 Syntactic-prosodic grouping of Genglish

Before starting to phonetize a Genglish sentence, and certainly before deciding of its intonation, it is important to organize words into some kind of hierarchical structure, in order to decide which groups of words which will be prosodically produced as single entities.

As a matter of fact, while isolated words receive stress on their *stressed syllable* (which can be obtained by dictinonary lookup or by rule, depending the language considered), not all word-level stressed syllables actually receive prosodic accentuation marks when words are grouped into sentences. Hence the need for some *syntactic-prosodic grouping* in a TTS system*.*

Although some authors identify several levels of hierarchy (sometimes confusingly termed as *prosodic words, breath groups, intermediate phrases, stress groups*, *intonational phrases, or breath groups*), many state-of-the art TTS systems actually distinguish only one level, which we shall term as *prosodic phrase,* and characterized by the fact that it includes only one accented syllable. It is important to understand that such an assumption imples that a speaker’s intonation on a given syllable only depends on the position of this syllable within its prosodic phrase, and on the position of this phrase within the sentence, plus of course on wether the syllable is tha accented syllable of the phrase or not.

Yet the crudest assumption is still to come : in many state-of-the-art TTS systems, prosodic phrases are identified with a rather trivial *chinks 'n chuncks* algorithm (after Liberman and Church, 1992). In this approach, a prosodic phrase break is automatically set when a word belonging the chunks group is followed by a word classified as a chink (or, in other words, a prosodic phrase is forced to be composed of the largest possible sequence of chinks, followed by the larget possible sequence of chunks; see Fig. 5). Chinks and chunks basically correspond to function and content words classes, with some minor modifications.

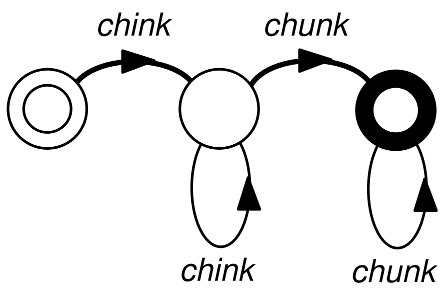


Fig. 5 A simple automaton for prosodic phrase using the chinks 'n chunks algorithm.

For the synthesis of Genglish, we will consider the following classes, defined in the `genglish_load_chinksnchunks.m` function :

Genglish chinks =	"and", "since", "the", "on", "of", "to", "it", "which", "gengled" (participle), "gengling", "is", "are"
Genglish chunks =	"gengle", "gengles", "gengled" (verb), "genglish", "gengly", "John", ",", "."

A short MATLAB file, `tts_phrase_using_chinksnchunks.m` implements our chinks 'n chunks module, which is then applied to the genglish corpus by the `genglish_test_chinksnchunks.m` script. This leads to the following typical phrases (shown here with the corresponding part-of-speech tags) :

'gengles' | *'are'* *'gengly'* | *'the'* *'gengle'* | *'of'* *'gengles'*
'noun' | 'auxiliary' 'adverb' | 'determiner' 'noun' | 'of' 'noun'

'it' *'is'* *'gengled'* | *'on'* *'gengling'* *'gengles'* | *'of'*
'gengles' |
'pronoun' 'auxiliary' 'participle' | 'preposition' 'participle' 'noun' | 'of'
'noun' |
'and' *'gengle'* *'gengles'* | *'on'* *'the'* *'gengle'*
'coordinator' 'noun' 'noun' | 'preposition' 'determiner' 'noun'

'the' *'gengly'* *'genglish'* *'gengle'* *'gengles'* | *'the'* *'gengle'* |
'determiner' 'adverb' 'adjective' 'noun' 'verb' | 'determiner' 'noun' |
'of' *'the'* *'gengled'* *'John'* *'gengle'* *', '* |
'of' 'determiner' 'participle' 'propername' 'noun' 'punctuation' |
'which' *'gengles'* | *'the'* *'genglish'* *'gengle'* *'gengle'*
'pronoun' 'verb' | 'determiner' 'adjective' 'noun' 'noun'

The last sentence phrasing, for instance, is obtained using tags obtained from the example in Section 2.2, by :

```
» phrases=tts_phrase_using_chinksnchunks(chinks,chunks,sentence,tags)'
phrases =
   1  1  1  1  1  2  2  3  3  3  3  3  3  4  4  5  5  5  5  5
```

in which numbers refer to the index of the phrase to which each word belong.

Allthough some groups are clearly not optimal (see the first group of the previous example, for instance, which abruptly ends with a verb), flavours of this algorithm are often used in commercial TTS systems.

3. PHONETIZATION

The phonetization (or *letter-to-sound*, LTS) module is responsible for the automatic determination of the phonetic transcription of the incoming text. At first sight, this task seems as simple as performing the equivalent of a sequence of dictionary look-ups. From a deeper examination, however, one quickly realizes that most words appear in genuine speech with several phonetic transcriptions, many of which are not even mentioned in pronunciation dictionaries. Namely :

1. Pronunciation dictionaries refer to word roots only. They do not explicitly account for morphological variations (i.e. plural, feminine, conjugations, especially for highly inflected languages, such as French).
2. Some words actually correspond to several entries in the dictionary, generally with different pronunciations. This is typically the case of heterophonic homographs, i.e. words that are pronounced differently even though they have the same spelling, as for *'record'* (/rekɔːd/ or /rɪkɔːd/), constitute by far the most tedious class of pronunciation ambiguities. Their correct pronunciation generally depends on their part-of-speech and most frequently contrasts verbs and non-verbs.
3. Words embedded into sentences are not pronounced as if they were isolated. Their pronunciation may be altered at word boundaries (as in the case of phonetic liaisons), or even inside words (due to rhythmic constraints, for instance).
4. Finally, not all words can be found in a phonetic dictionary : the pronunciation of new words and of many proper names has to be deduced from the one of already known words.

Automatic phonetizers dealing with such problems can be implemented in many ways, often roughly classified as *dictionary-based* and *rule-based* strategies, although many intermediate solutions exist.

Dictionary-based solutions consist of storing a maximum of knowledge into a lexicon. Entries are sometimes restricted to morphemes, and the pronunciation of surface forms is accounted for by inflectional, derivational, and compounding morphophonemic rules. Morphemes that cannot be found in the lexicon are transcribed by rule. This approach has been often followed for the synthesis of English (see Levinson et al. 1993, with their morpheme lexicon of 43,000 morphemes).

Languages with more complex morphology tend to be used rule-based transcription systems, which transfer most of the phonological competence of dictionaries into a set of letter-to-sound (or grapheme-to-phoneme) rules. Rules are either found by experts, through trials and errors, or obtained automatically using corpus-based methods for deriving phonetic decision trees (see Damper 2001).

3.1 Corpus-based Genglish phonetization

Considering Genglish phonetization, the problems quoted above can be precisely identified :

1. Given its closed lexicon, the phonetization of morphological variations of Genglish can easily be addressed by storing each variation into a phonetic lexicon.
2. Only *"gengle"* and *"gengles"* are heterophonic homographs. Storing separate entries for each word *and* a given part-of-speech in our phonetic lexicon would solve the problem.
3. The only Genglish word whose phonetization could change when embedded in a sentence is *"is"*¸ which could be pronounced as *" 's"*. We will assume it is always produced in its full form.
4. Unknown words do not exist in Genglish.

Dealing with Genglish thus *a priori* offers a comfortable dictionary-based solution for phonetization. For tutorial reasons, however, we rather develop here a small corpus-based phonetizer, implemented as a decision tree trained on real data. Besides, this generic technique is increasingly used in multilingual TTS systems.

3.2 Decision trees

Decision trees describe how a given input can possibly correspond to specific outputs, as a function of some contextual factors. At each non-terminal node, there is a question requiring a yes or no answer about the value of the contextual factor associated with the node, and for each possible answer there is a branch leading to the next question Questions relate to contextual factors assumed to be useful. Terminal nodes, or *leaves*, are associated with a specific output.

Such decision trees can be automatically obtained by *classification and regression tree* (CART) training techniques (Brieman *et al*., 1984; see also Damper, 2001), which automatically allow the most significant contextual factor to be statistically selected using a greedy algorithm.[11]

In order to build a tree, one needs a *training set* composed of inputs (or *features*) associated with outputs (or *labels*). From this set, relations between feature values and outputs are used to determine predictors for these outputs. At start-up, all the training data is assigned to a first *parent* node. The tree is then built by recursively splitting the data in a parent node into subsets that form descendant or *child* nodes. Each node encodes the distribution of the training data in a given context. Central to CARTs is the node splitting algorithm, based on the minimization of the *entropy* of the training data. Entropy is an information theoretic concept that can be though of a measure of the "randomness" of the data. It is measured in *bits* and computed as the negative of the mean of the log-likelihood of all labels *l* in the set of admissible ones *L*:

$$H(L|\text{node}) = -\sum_{l \in L} P(l|\text{node}) \log_2 P(l|\text{node}) \qquad (6)$$

[11]A greedy algorithm makes optimal decisions at each step, without regard to subsequent steps. It aims to build a tree which is locally optimal, but very likely not globally optimal.

This value differs from node to node, as the distribution of labels at a node is influenced by all the choices that have been made based on features from the very first parent node. If, for instance, *L* contains 2^N labels seen as equally probable at a given node position, formula (6) reduces to $H(L|\text{node}) = -\log_2(2^{-N}) = N$ —precisely the number of bits required to encode any label *at that node position*. In case labels are not equally probable, which corresponds to a reduction of the "randomness" of *L, H(L)* decreases, and any label can be coded with less than *N* bits. Each splitting of a parent node, based on the partition c_i^j of the values of a contextual factor c_i into two subsets C_1^j and C_2^j, produces two child nodes and the resulting average entropy is given by:

$$H(L|\text{child}_1,\text{child}_2) = H(L|\text{child}_1)P(\text{child}_1) + H(L|\text{child}_2)P(\text{child}_2) \qquad (7)$$

where $P(\text{child}_1)$ and $P(\text{child}_2)$ are the probabilities of visiting the child nodes —the probabilities that c_i falls in C_1^j and C_2^j, respectively, given their parent node[12]. Indices *i* and *j* respectively refer to features (in the set of *a priori* useful ones, which has to be established in advance) and to partitions of their values. The key idea is that the best splitting $c_{i,best}^{j,best}$ is the one that maximizes the difference between the entropies before and after splitting. This difference is defined as the *average mutual information* $I(L, c_i^j)$ between the labels to predict and splitting c_i^j :

$$I(L,c_i^j) = H(L|\text{parent}) - H(L|\text{child}_1,\text{child}_2) \qquad (8)$$

It can be obtained by first examining each feature c_i and finding the partition $c_i^{j,best}$ that maximizes $I(L, c_i^j)$, and then maximizing $I(L, c_i^{j,best})$ over *i*:

$$c_i^{j,best} = \arg\max_j I(L,c_i^j) \qquad (9)$$

$$c_{i,best}^{j,best} = \arg\max_i I(L,c_i^{j,best}) \qquad (10)$$

The node splitting algorithm is iteratively applied on child nodes, and branches of the tree are stopped when maximum average mutual information falls below a threshold, in which case a further reduction of entropy is seen as not significant.

Let us consider the simple object sequence of Fig. 5, in which we would be asked to find a good predictor for the color of any of the objects, as a function of its shape and size, as well as of the shapes and sizes of its surrounding. Not an easy task at first sight…

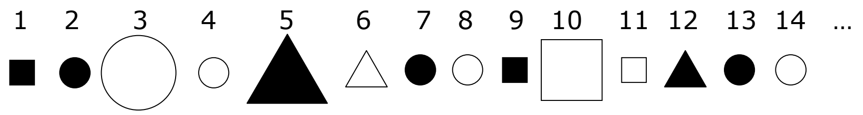


Fig. 5 A sequence of objects of various shapes, sizes and colors.

[12]As a matter of fact, these probabilities cannot be estimated *a priori*: they have to take account of the choices that have previously been made to lead to the parent node.

Let us then organize the information we have in the form of features (*SH(n)* : the shape of the *n*th object, *S(n)* : the size of the *n*th object) and outputs (*C(n)* : the colour of the *n*th object), (Table 2)

Table 2 Output corresponding to given features

C(n)	*S(n)*	*SH(n)*	*S(n-1)*	*SH(n-1)*	*S(n+1)*	*SH(n+1)*	…
Black	Small	Square	-	-	Small	Circle	…
Black	Small	Circle	Small	Square	Big	Circle	…
White	Big	Cricle	Small	Circle	Small	Circle	…
White	Small	Circle	Big	Circle	Big	Triangle	…
…	…	…	…	…	…	…	…

We are now left with the search for a question which tends to split our initial set of objects (in which colour seems random) into two child sets with minimal randomness (Fig 6). The entropy of a set is simply computed in this case as :

$$H = - P(\text{Black}) \log_2[P(\text{Black})] - P(\text{White}) \log_2[P(\text{White})] \qquad (8)$$

Since there are as many black objects as white objects in the initial set, its entropy is given by -(1/2 x -1) -(1/2 x -1) = 1 bit. After trying all sorts of possible questions on contextual factors, one finds that "*Is the shape of the previous object identical to that of the current object*" is a question which effectively predicts the object's colour (white if yes, black otherwise) : this question splits the initial set into two sets with null entropy, thereby maximizing the information gain in a single step.

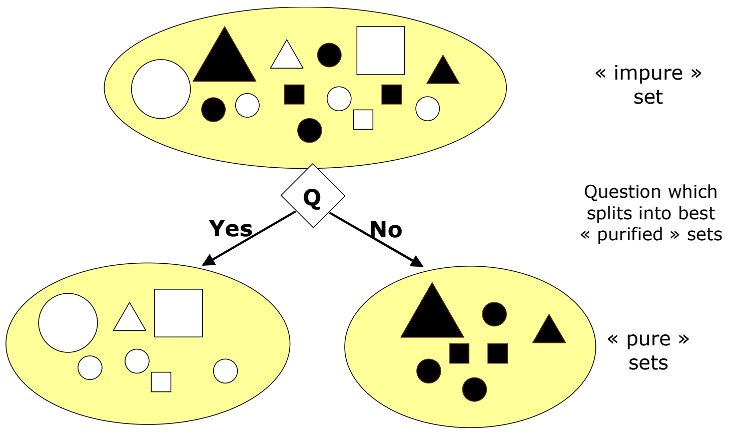


Fig. 6 Splitting a set in which colour seems randomly attributed to objects into subsets in which colour is no longer random.

3.2 Phonetic decision trees for Genglish

In the framework of automatic phonetization, the features used in the decision tree are simply the letter being currently phonetized, the letters on the left and right of the current letter, and the part-of-speech of speech of the current word (so as to handle heterphonic homographs). Outputs are phonemic symbols. (Fig. 7).

Phonetic transcriptions are given for each word in the Genglish training corpus (the one we have used for n-grams), in such a way that each letter in the word gets its phonetic symbol (including the null symbol '_' if needed). For practical reasons, the phonetic symbols used in the corpus are chosen so that each phoneme gets a single phonetic character[13] (Table 3).

Table 3 Phonetics of Genglish, using in-house phonetic alphabet

'gengle'	'JEN_l_' (verb) 'gEN_l_' (noun)	'John'	'JQ_n'
'gengles'	'JEN_l_z' (verb) 'gEN_l_z' (noun)	'and'	'End'
'gengled'	'JEN_l_d'	'since'	'sIns_'
'gengling'	'JEN_lIN_'	'the'	'D_@'
'is'	'Iz'	'on'	'Qn'
'are'	'a__'	'of'	'Qv'
'be'	'bi'	'to'	'tU'
'genglish'	'gEN_lIS_'	'it'	'It'
'gengly'	'gEN_lI'	'which'	'w_IC_'

Using a MATLAB script, `corpus_to_phonetic_corpus.m`, we reorganize this data to create `genglish_phonetic_corpus`, a specific phonetic corpus (Table 4) with the same format as that of Table 2:

```
» genglish_phonetic_corpus=corpus_to_phonetic_corpus(genglish_corpus);
» genglish_phonetic_corpus(1:3,:)
ans =
gg__enN
Ee_gngN
NngeglN
```

[13] The best practice would have been to follow the recommendations of SAMPA (Speech Assessment Methods Phonetic Alphabet), which is increasingly used : http://www.phon.ucl.ac.uk/home/sampa/home.htm. But it would have forced us to use several phonetic characters for some phonemes.

Table 4 First lines of the phonetic corpus : phonemes *P(n)* corresponding to a given letter *L(n)* and contextual factors (2 letters on the left, *L(n-1)* and *L(n-2)*, 2 letters on the right, *L(n+1)* and *L(n+2)*and Verb/Non-verb distinction *V/N*)

P(n)	*L(n)*	*L(n-2)*	*L(n-1)*	*L(n+1)*	*L(n+2)*	*V/NV*
g	g	-	-	e	n	N
E	e	-	g	n	g	N
N	n	g	e	g	l	N
g	g	e	n	l	e	N
…	…	…	…	…	…	…

We then train a cart tree on our phonetic corpus, using Using a MATLAB script, `cart_train.m`, a simple and elegant[14] MATLAB implementation of the principle exposed in section 3.2, and use `cart_print.m` to see the results (see also Fig. 8) :

```
» genglish_phonetic_cart=cart_train(genglish_phonetic_corpus);
» cart_print(genglish_phonetic_cart)
```

```
if 2=_ if 4=e if 1=g if 6=N out=g
                     else   out=J
              else   if 1=h out=_
                     else   if 1=b out=b
                            else   out=r
       else   if 4=n if 1=o out=Q
                     else   if 1=i out=I
                            else   out=E
              else   if 1=t if 4=h out=D
                            else   out=t
                     else   if 1=o if 3=t out=U
                                   else   out=Q
                            else   if 3=_ if 1=i out=I
                                          else   if 1=J out=J
                                                 else   if 1=a out=A
                                                        else   if 1=s out=s
                                                               else   out=w
                                   else   if 1=n out=n
                                          else   if 1=f out=v
                                                 else   if 1=s out=z
                                                        else   if 1=e out=i
                                                               else   out=_
else   if 1=n if 4=g out=N
              else   out=n
       else   if 1=l out=l
              else   if 2=l if 3=e if 1=d out=d
                                   else   out=z
                            else   out=S
                     else   if 2=t out=@
                            else   if 1=i out=I
                                   else   if 1=d out=d
                                          else   if 1=y out=I
                                                 else   if 1=c if 2=h out=C
                                                               else   out=s
                                                        else   out=_
```

An interesting part of the tree is precisely its first nodes, which transcribe initial ‘g’ :

```
if 2=_ if 4=e if 1=g if 6=N out=g
                     else   out=J
```

[14] The MATLAB implementation is recursive, accounting for the fact that building a tree from its top is similar to buiding a tree from any of its internal nodes.

The tree first examines the second context feature, *L(n-2)*, and compares it to '_' (which, if true, indicates that *L(n)* is the initial character of a word). The tree thereby first examines the pronunciation of the first character of each word in its top left branch. It then immediately examines the pronunciation of words beginning with *'ge'* (by checking that *L(n+1)* is 'e' and *L(n)* is *'g'*), which as we know contain the major heterophonic homography of Genglish : [g/J] for the Verb/Non-verb distinction (the 6th feature). The tree handles this by assigning [g] to all non verb forms, and [J] to verbs. (Fig 8). This clearly shows that it has learnt about the importance of the V/NV tag.

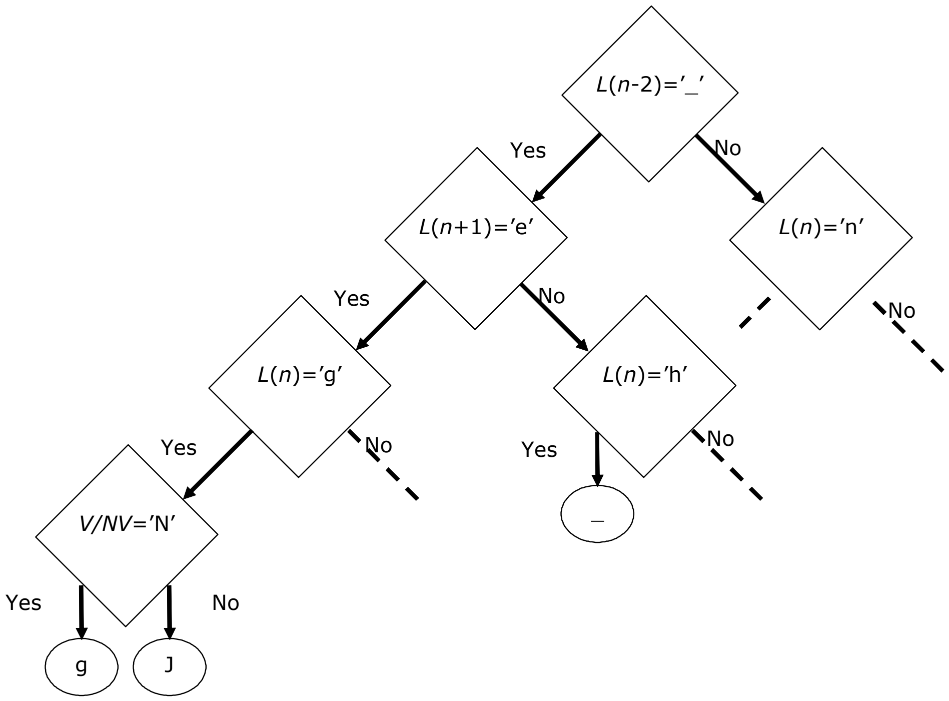


Fig. 7 Top nodes of the cart tree trained on our Genglish phonetic corpus.

Using a phonetic cart for assigning a phoneme to a character in context is easy. A short MATLAB function does the job, recursively : `cart_run.m`. This function is used iteratively by `tts_phonetize_using_cart.m` to produce the phonetization of all the characters of a word or sentence (and the 'verbose' option makes it print the nodes it follows):

```
>>tts_phonetize_using_cart({'gengles'},{'noun'},genglish_phonetic_cart,'verbose')
2=_ 4=e 1=g 6=N out=g
2=_ 4~=e 4=n 1~=o 1~=i out=E
2~=_ 1=n 4=g out=N
2~=_ 1~=n 1~=l 2~=l 2~=t 1~=i 1~=d 1~=y 1~=c out=_
2~=_ 1~=n 1=l out=l
2~=_ 1~=n 1~=l 2~=l 2~=t 1~=i 1~=d 1~=y 1~=c out=_
2~=_ 1~=n 1~=l 2=l 3=e 1~=d out=z
word=gengles   pos=noun   phonemes=gEN_l_z
```

We test this function on our complete Genglish test corpus (taking the part-of-speech information for each word from the corpus, not from *n*-gram tagging), using `genglish_test_cart.m`, and find no error[15].

4.4. PROSODY GENERATION

4.1 Prosodic information

The term *prosody* refers to certain properties of the speech signal which are related to audible changes in pitch, loudness, syllable length. Prosodic features have specific functions in speech communication (see Fig. 8). The most apparent effect of prosody is that of *focus* : some pitch events make a syllable stand out within the utterance, and indirectly the word or syntactic group it belongs to, will be highlighted as an important or new component in the meaning of that utterance.

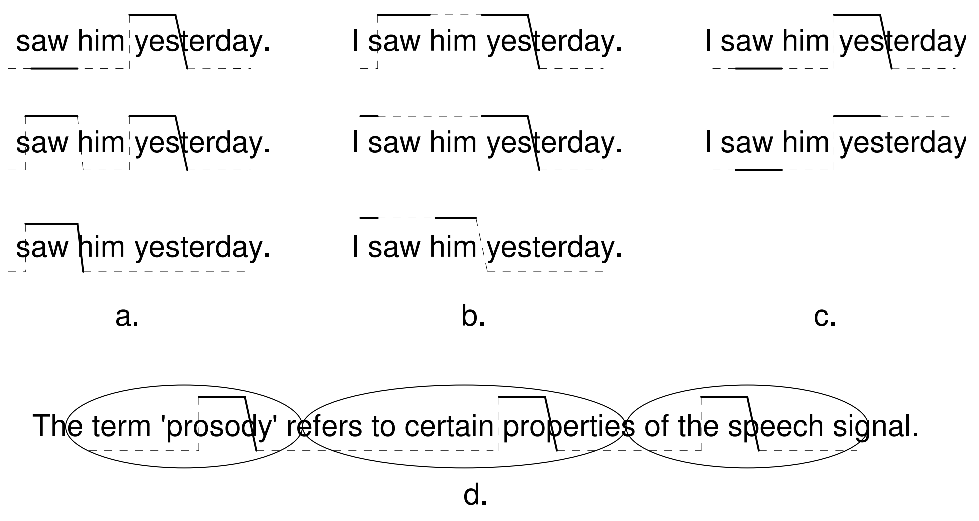


Fig. 8 Different kinds of information provided by intonation (lines indicate pitch movements; solid lines indicate stress).
a. Focus or given/new information;
b. Relationships between words (saw-yesterday; I-yesterday; I-him);
c. Finality (top) or continuation (bottom), as it appears on the last syllable;
d. Segmentation of the sentence into groups of syllables.

Although maybe less obvious, prosody has more systematic or general functions. Prosodic features create a segmentation of the speech chain into groups of syllables, or, put the other way round, they give rise to the grouping of syllables and words into larger chunks, termed as *prosodic phrases* and already mentioned in section 2.3. Moreover, there are prosodic features which suggest relationships between such groups, indicating that two or

[15] Given the extreme simplicity of Genglish phonetization, this is to the least desirable.

more groups of syllables are linked in some way. This grouping effect is hierarchical, although not necessarily identical to the syntactic structuring of the utterance.

It is thus clear that the prosody we produce draws a lot from syntax, semantics, and pragmatics. This immediately rises a fundamental problem in TTS synthesis : how to produce natural sounding intonation and rhythm, without having access to these high levels of linguistic information? The tradeoff that is usually adopted when designing TTS systems is that of '*acceptably neutral*' prosody, defined as the default intonation which might be used for an utterance out of context. The key idea is that the "correct" syntactic structure, the one that precisely requires some semantic and pragmatic insight, is not essential for producing such acceptably neutral prosody. In other words, TTS systems focus on obtaining an acceptable segmentation of sentences and translate it into the continuation or finality marks of Fig. 9.c. They often ignore the relationships or contrastive meaning of Fig. 9.a and 9.b., which require a higher degree of linguistic sophistication.

4.2 Prosody as a by-product of unit selection in a large speech corpus

The organization of words in terms of prosodic phrases can be used to compute the duration of each phoneme (and of silences), as well as their intonation (this is what we do when reading the phrases obtained in section 2.3). This operation, however, is not straightforward. It requires the formalization of a lot of phonetic or phonological knowledge on prosody, which is either obtained from experts or automatically acquired from data with statistical methods.

One way of achieving this is by using *linguistic models* of intonation as an intermediate between syntactic-prosodic parsing and the generation of acoustic pitch values. The so-called *tone sequence theory* is one such model. It describes melodic curves in terms of relative *tones*. Following the pioneering work of Pierrehumbert for American English, tones are defined as the phonological abstractions for the target points obtained after broad acoustic stylization. This theory has been more deeply formalized into the *ToBI (Tones and Break Indices)* transcription system (Silverman et al. 1992). Mertens (1990) developed a similar model for French.

How the *F0* curve is ultimately generated greatly on so-called *acoustic models* of intonation. A typical approach is that of using Fujisaki's acoustic model, which describes F0 curves as the superpositions of phrase and accent curves. The analysis of timing and amplitude of these curves (as found in real speech) in terms of linguistic features (tones, typically) can be performed with statistical tools (see Möebius et al. 1993, for instance). Several authors have also recently reported on the automatic derivation of F0 curves from tone sequences, using statistical models (Black and Hunt 1996) or corpus-based prosodic unit selection (Malfrère et al. 1998).

Similarly, two main trends can be distinguished for phoneme duration modeling. In the first one, durations are computed by first assigning an *intrinsic* (i.e., *average*) *duration* to phonemes, which is further modified by successively applying rules combining co-intrinsic and linguistic factors into additive or multiplicative factors (for a review, see van Santen 1993). In a second and more recent approach mainly facilitated by the availability of large speech corpora and of computational resources for generating and analyzing these corpora, a very general duration model is proposed (such as CARTs). The model is automatically trained on a large amount of data, so as to minimize the difference between the durations predicted by the model and the durations observed on the data.

A still more recent trend is … not to compute F0 or duration values at all! In this case, prosody is obtained as a by-product of *unit selection* from a large speech corpus, using phonetic features (such as current and neighbouring phonemes), as well as linguistic features (such as stress, position of the phoneme within its word, position of the word withing its prosodic phrase, position of the prosodic phrase within the sentence, part-of-speech tag of the current word, etc.) to find a sequence of speech segment (or unit) taken from the speech corpus, whose features most closely match the features of the speech unit to be synthesized. This is the approach we will follow in this document. More on this in section 5.

5. CONCATENATIVE SYNTHESIS

Commerical TTS systems currently employ one main category of techniques for speech signal generation: *concatenative synthesis*, which attempts to synthesize speech by concatenating *acoustic units* (e.g., half-phonemes, phonemes, diphones, etc.) taken from natural speech.

This approach has resulted in significant advances in the quality of speech produced by speech synthesis systems over the past 15 years. In contrast to previous synthesis methods (known as *synthesis by rule*, or *formant synthesis*), the concatenation of acoustic units avoids the difficult problem of modeling the way humans generate speech. However, it also introduces other problems: the choice of the type of acoustic units to use, the concatenation of acoustic units that have been recorded in different contexts, and the possible modification of their prosody (intonation, duration)[16].

Word-level concatenation is impractical because of the large amount of units that would have to be recorded. Also, the lack of coarticulation at word boundaries results in unnaturally connected speech. Syllables and phonemes seem to be linguistically appealing units. However there are over 10000 syllables in English and while there are only 40 phonemes, their simple concatenation produces unnatural speech because it does not account for coarticulation.

5.1 Diphone-based synthesis

In contrast, *diphones* are currently used in many concatenative systems. A diphone is a speech segment which starts in the middle of the stable part (if any) of a phoneme, and end in the middle of the stable part of the next phoneme. Diphones therefore have the

[16] Notwithstanding the compression of the unit inventory using a speech coding technique : concatenative synthesis techniques tend to require large amounts of memory. We do not examine this problem here.

same average duration as phonemes (about 100 ms), but if a language has *N* phonemes, it typically has about *N*² diphones[17], which gives a typical diphone database size of 1500 diphones (about 3 minutes of speech, i.e. about 5Mb for spech sampled at 16Khz with two bytes per sample). Some diphone-based synthesizers also include multi-phone units of varying length to better represent highly coarticulated speech (such as in /r/ or /l/ contexts).

For the concatenation and prosodic modification of acoustic units, *speech models* are used. They must provide a parametric form for acoustic units which makes it possible to modify their local spectral envelope (for smoothing concatenation points) and their pitch and duration, without introducing audible artifacts. There has been a considerable amount of research effort directed at the design of adequate speech models for TTS. Linear prediction (LP) has been used first (Markel et al. 1976) for its relative simplicity. However, the buzziness inherent in LP degrades perceived voice quality. Other synthesis techniques based on pitch synchronous waveform processing have been proposed such as the Time-Domain Pitch-Synchronous-Overlap-Add (TD-PSOLA) method (Moulines et al. 1990). TD-PSOLA is currently one of the most popular concatenation methods. Although TD-PSOLA provides good quality speech synthesis, it has limitations which are related to its non-parametric structure: spectral mismatch at segmental boundaries and tonal quality when prosodic modifications are applied on the concatenated acoustic units. An alternative method is the MultiBand Resynthesis Overlap Add (MBROLA) method (Dutoit 1997, Chapter 10) which tries to overcome the TD-PSOLA concatenation problems by using a specially edited inventory (obtained by resynthesizing the voiced parts of the original inventory with constant harmonic phases and constant pitch). Both TD-PSOLA and MBROLA have very low computational cost. Sinusoidal approaches (e.g., Macon 1996) and hybrid harmonic/stochastic representations (Stylianou 1998) have also been proposed for speech synthesis.

5.2 Automatic unit selection synthesis

An extension of these strategies called *automatic unit selection* (Hunt and Black 1996) has recently been introduced, and opened new horizons to speech synthesis. Given a phoneme stream and target prosody for a utterance, this algorithm selects, from a very large speech database (1-10 hours typically, or 150-1500 Mb), an optimum set of acoustic units (typically isolated diphones or sequences of contiguous diphones) which best match the target specifications (Fig. 9). For every *target unit* required (typically, every dihpone to be synthesized), the speech database proposes lots of *candidate units*, each in a different context (and in general not exactly in the same context as the target unit). When candidate units cannot be found with the correct prosody (pitch and duration), prosody modification can be applied. Candidate units usally do not concatenate smoothly (unless a sequence of such candidate units can be found in the speech database, matching the target requirement), some smoothing can be is applied. Recent synthesizers, however, tend to avoid prosodic modications and smoothing, which sometimes create audible

[17] A bit less in practice : not all diphones are practically encountered in natural languages.

artefacts, and keep the speech data as is (thereby accepting some distrosion between the target prosody and the actual prosody produced by the system, and some spectral envelope discontinuities).

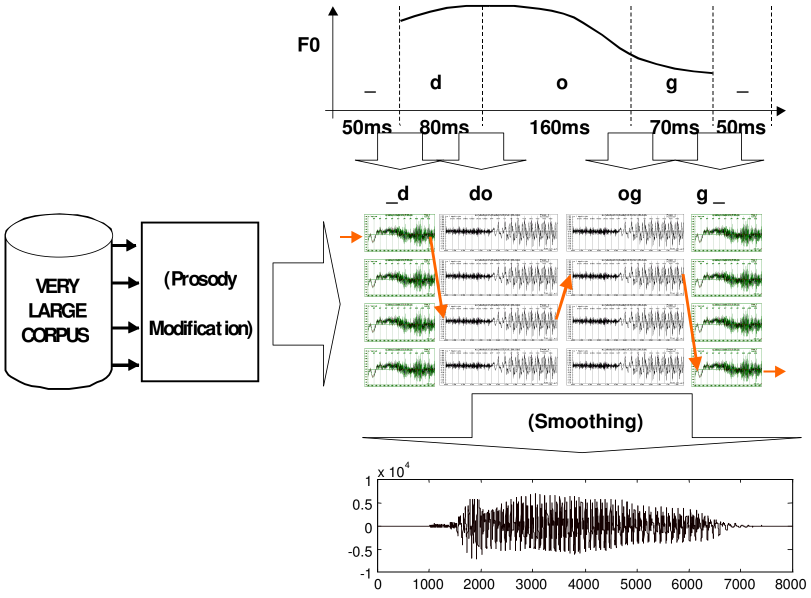


Fig. 9 A schematic view of a unit selection speech synthesizer. The prosody modification and smoothing modules have been mentioned between parentheses, since they are not always implemented.

The biggest challenge of unit selection synthesis lies in the search for the "best" path in the candidate units lattice. This path must minimize two costs simultaneously : the overall *target cost,* defined as the sum of elementary target costs between each candidate unit chosen and the initial target unit, and the overall *concatenation cost*, defined as the sum of elementary concatenation costs between successive candidate units. The elementary target cost is typically computed as a weighted sum of differences between the linguistic features of units. The elementary concatenation cost is usually computed as a weighted sum of acoustic differences between the end of the left candidate unit and the beginning of the right candidate unit (Fig. 10).

Ideally, target costs should reflect the acoustic (or even perceptual) difference between target units and candidate units. Since the acoustics of the target are not available (the synthesizer is precisely in charge of producing them), only linguisitic features can be used for estimating the segmental difference. An estimation of the difference between target and candidate prosody (i.e., the supra-segmental difference) can be achieved through tones (which implies to predict target tones and compare them to unit tones) or through pitch and duration values (this, however, implies to have predicted the pitch and duration of targets). On the other hand, ideal concatenation costs should reflect the perceptual discontinuity (as opposed to the acoustic one) between successive candidate units. These issues are still open.

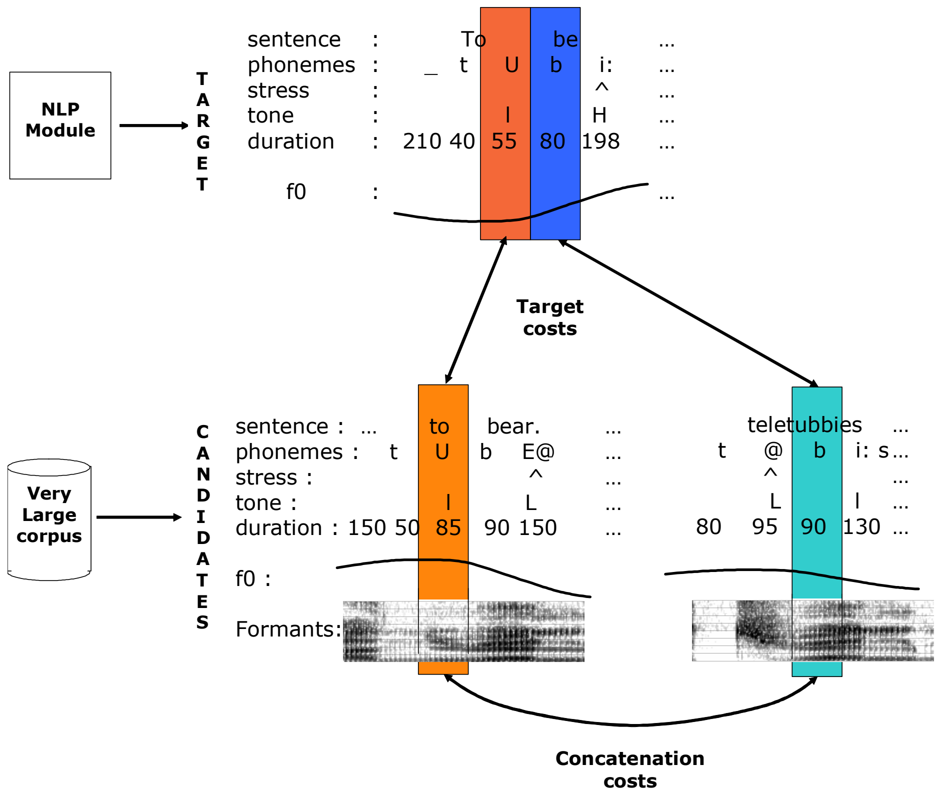

NLP Module
TARGET
sentence : To be ...
phonemes : _ t U b i: ...
stress : ^ ...
tone : l H ...
duration : 210 40 55 80 198 ...
f0 : ...
Target costs
Very Large corpus
CANDIDATES
sentence : ... to bear. ... teletubbies ...
phonemes : t U b E@ ... t @ b i: s ...
stress : ^ ... ^ ...
tone : l L ... L l ...
duration : 150 50 85 90 150 ... 80 95 90 130 ...
f0 :
Formants:
Concatenation costs

Fig. 10 Elementary target and concatenation costs. (Top : Target units, defined by their linguistic and prosodic features, computed by the NLP module. Bottom : candidate units found in the speech database, defined by their linguistic, prosodic, *and* acoustic features)

Unit selection tends to avoid as many concatenation points as possible, by selecting the largest sequences of consecutive units in the database, since this greatly dicreases the overall concatenation cost : the elementary concatenation cost between successive candidate units in the database is simply set to zero.

5.3 Unit selection synthesis of Genglish

Let us now develop a small but efficient unit selection-based Genglish synthesizer.

The first step is to record a speech corpus, which should be large enough for all sorts of phonetic sequences to appear in various linguistic contexts, so that the resulting speech segments are available in various prosodic forms. In order to make things simple, we have recorded the same 50 sentences as those stored in our Genglish (text) corpus and stored them in files named 1.wav, 2.wav, …, and 50.wav.

Segmenting speech into phonemes is not an easy task, even when phonemes are know (in which case this operation is termed as *alignment*). Done by hand, it takes forever; done by machines, it is never completely reliable. We have used an HMM-based text-to-speech alignement system developed at TCTS Lab (Malfrère *et al.*, 2003)[18] and produced corresponding .seg files, the content of which is easy to understand: each line mentions a start, an end (in sample), and a phoneme name. Alignment, however, is conditioned by the degree of correspondence between the assumed phonemic transcription (sometimes calle the *orthepic* transcription) and the actual list of phonetic units produced. The gap between these two worlds is undoubtedly the most difficult to bridge. Differences are mostly caused by coarticulation, which cannot be taken into account in phonemic transcriptions. Good examples can be found in 'it gengles' (46.wav), whose /t/ basically disappears and causes the occlusion of the succeeding [J], or in 'to gengle' (26.wav) whose /U/ gets neutralized into [@]. Phonemic-phonetic mismatches also result from personal or local speaking styles, as mostly obvious in the way speakers set pauses in ther speech (for obtaining meaningfull .seg files, we have inserted pauses in the phonetic transcription of sentences after listening to the recordings). To a larger extent, performing text-to-speech alignement rises the issue of the phonetic set to use. In our segmentation, for instance, we have decided to transcribe 'gengled' into /JENgld/ and not /JENg@ld/*,* so as to avoid having to handle very short [@] "phonetic" units*.*

In order to check the resulting segmentation and correct if when necessary, we have used the WAVESURFER tool developed at the Centre for Speech Technology (CTT) at KTH in Stockholm (which reads the same .seg files). As an example, Fig. 11 shows the segmentation information for 1.wav.

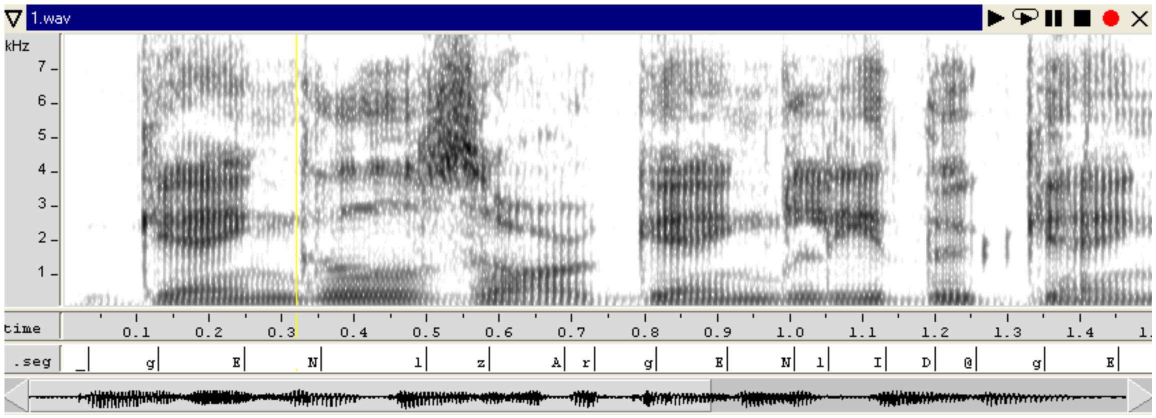


Fig. 11 A typical Genglish sentence (1.wav) and its phoneme segmentation, as shown by WAVESURFER.

From this segmented speech corpus, we have built a speech unit database, in which we have stored, for each available unit, the minimum information needed to compute its match to a given phonemic target. They are : its phoneme, previous phoneme, next

[18] But any other available tool could have been used, including … no tool at all.

phoneme, the index of the the part-of-speech (pos) of the current word, the index of the current prosodic phrase (within the current sentence), the number of prosodic phrases on the right (until the end of the sentence), the index of the current word (within the current prosodic phrase), the number of words on the right (until the end of the current prosodic phrase), the index of the sentence containing the phoneme (related wav file names are given by this index), and the start and the end sample for the current phoneme in the related wav file. This operation is achieved by the `corpus_to_speech_corpus.m` file; the first units in the database can easily be listed :

```
» speech_corpus= corpus_to_speech_corpus (genglish_corpus);
» speech_corpus(1:3,:)
ans =
    '_#gn1310'    [1]    [   0]    [ 533]
    'g#En1310'    [1]    [ 533]    [2069]
    'EgNn1310'    [1]    [2069]    [3982]
```

in which the information related to the first unit should be understood as follows : phoneme [_] preceded by nothing and followed by [g], in a noun, in the first of 4 phrases (there are 3 phrases after the current one), in the first word of the phrase (which contains only one word : no word on the right), can be found in file 1.wav, from sample 0 to 533.

As can be seen from the previous example, the first member of the unit data (a string in our case), termed as the linguistic context features of the unit, does not explicitly refer to prosody : neither stress nor target tones[19], nor even target intonation or duration values are used here. This choice was deliberately made for keeping our unit selection as simple as possible. It even makes sense for the synthesis of natural languages, as shown by the recently developed LiONS TTS system (Beaufort & Colotte, 2004; see also http://www.multitel.be/TTS/layout.php? page=LiONS).

We then design two functions to let us query this database : one for creating a list of formatted phonetic targets from the list of words, tags, phrases, and phonemes : `tts_set_targets.m`, and the other for finding best matches between a list of targets and units in the database : `tts_select_units.m`. The first function naturally formats targets in the same form as units (assigning them the same linguistic context feature set), except the acoustic information (in our case, the name of the wav file and the start/end sample) is missing : this is precisely what we are looking for. The second first checks for available diphones in the speech unit database matching the target diphones. It prunes the list downto a maximum of 10 units per diphone (so as to accelerate the search) and implements a Viterbi algorithm for finding the best sequence of units, the one that minimizes an overall selection cost.

The target cost is defined in a very crude way : it is 1 if the linguistic context features of unit and target match, and 0 otherwise. In other words : we have given the same weight to all linguistic features. The concatenation cost simply complies with the requirement

[19] For the sake of simplicity, we do not even refer to syllables in TTSBOX, and therefore explicitly avoid using the notion of stress. This, however, sometimes leads to some strange stress assignment.

mentioned in 5.2 : it is set to 0 for consecutive units, and to 1 for others. No acoustic distance is computed.

As expected, trying our unit selection on targets taken from units in the database simply returns the units themselves :

```
>> genglish_load_corpus;
>> speech_corpus=corpus_to_speech_corpus(genglish_corpus);
>> units=tts_select_units(speech_corpus,speech_corpus(1:26,:),'verbose')’
units =
     1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25
```

Last but not least, we use a simple MATLAB function, `tts_concatenate_using_xorr.m`, which extracts selected diphones from the speech corpus and assembles them into synthetic speech. It is well known that this operation tends to produce audible mismatches (namely, spectral amplitude mismatches, pitch mismatches, amsn phase mismatches; see Dutoit, 1997, Chap. 10). A minimalist treatment of phase mismatches is implemented here, by computing the cross-correlation between units to assemble, and slightly shifting the second one accordingly (so as to maximize cross-correlation around the concatenation point). Taking care of the other types of mismatches would require the more advanced frequency-domain synthesis systems mentioned in 5.1.

6. GENGLISH TEXT-TO-SPEECH SYNTHESIS AND BEYOND

Having designed all the functions mentioned above, and having tested them on our Genglish test corpus, we create the `tts_run.m` function, which performs Genglish text-to-speech synthesis. The essential content of this function speaks for itself :

```
function speech=tts_run(sentence_chars,verbose)
   sentence=tts_preprocess_using_fsm(sentence_chars);
   possible_tags=tts_morph_using_lexicon(sentence,morph_lex);
   tags=tts_tag_using_bigrams(emission_probs,transition_probs,morph_lex,pos_lex,
        sentence,possible_tags);
   phrase_indices=tts_phrase_using_chinksnchunks(chinks,chunks,sentence,tags);
   phonemes=tts_phonetize_using_cart(sentence,tags,phonetic_cart,verbose);
   target_sequence=tts_set_targets(sentence,tags,phrase_indices,phonemes);
   units=tts_select_units(speech_corpus,target_sequence,verbose);
   speech=tts_concatenate_using_xorr(speech_corpus,units);
```

As explained in the previous section, asking for a sentence in the speech corpus most aften results in the original recording (which is to the least desirable)[20]. Asking for other sentences produces natural audio results, although some discontinuities can be heard.

Try for instance :

```
>> genglish_init_tts;
>> tts_run('The gengle of John is genglish on the gengle','results');
```

[20] In some case, however, the TTS system chooses to assemble units from several recordings. The explanation for this is left as an exercise (see Section 4.8).

In a beta version of the system, asking for irregular sentences would stop the execution of the program, due to some target diphones which had no match in the unit database, as it is the case for diphone [nQ] in the following example :

```
>> tts_run('John on John','verbose');
```

This was a very typical drawback of early unit-selection-based TTS systems. The solution adopted here is the one which was used in the early days of NUU synthesis : that of using another diphone, whose phonetic content is not too different from the target. In our case, we simply impose the first phoneme to be the same as in the target. A more efficient approach would be to use broad phonetic classes and compute real phonetic distances between diphones (thereby replacing missing phonemes by phonetically close candidates, as in [On] replaced by [Om]). In today's NUU systems, a (complete) diphone database is also often provided, so as to ensure that at least one instance of each diphone is available.

One should not conclude that the limited size and complexity of our Genglish TTS system adds to the credibility of the odd quotation which started this document! As a matter of fact, a number of issues have not been treated here, while they are required for handlingin natural languages. What is more, most of the concepts used in this document have only been used in a crude way, and many refinements could be adopted to face these new issues. Let us mention, for instance:

- Pre-processing: how to handle numbers, acronyms, abbreviations, uppercase titles ?
- Morphological analysis: how to handle out-of vocabulary words, spelling mistakes ?
- Contextual analysis: how do we go "up from trigrams"? (Jelinek, 1991). Multigrams have proved to lead to improvements (Deligne & Bimbot, 1995) Which tags to consider? Tags, indeed, have a direct impact on the segmentation into syntactic-prosodic phrases. Tagging 'which' as 'relative' instead of 'pronoun', for instance, would certainly help phrasing. Increasing the number of tags, however, implies having a larger corpus for training n-grams.
- Phonetization: it was assumed that the grapheme/phoneme association in the corpus was known, while in practice one has to set it.
- Phrasing: it is clear that even Genglish is not correctly phrased using chinks and chunks. More elaborate models will also use several levels of grouping: the first will be acoustically marked by the use of pauses, the second will simply result in melodic variations.
- NUU synthesis: we have not considered the issue of corpus design, of which the quality of synthetic speech greatly depends. This is still more true when the speech corpus is made as small as possible, as the natural distribution of units has to be maintained artificially in the corpus. We also have not investigated the importance of weights in the definition of target and concatenation costs. There is no doubt that many concatenation problems could be avoided by performing better unit selection.
- We have not mentioned how to assess synthetic speech quality (both in terms of naturalness and intelligibility) assessment (see for instance Boeffard & d'Alessandro, 2002).

7. DO IT YOURSELF

We would like to conclude this presentation by suggesting a number of possible ways to expore TTS synthesis by extending or modifying our functions.

1. Implement a more realistic pre-processing by using the MATLAB's `regexp` function for handling regular expressions. Try to create a small pre-processor which would easily detect non-Genglish sentence (MATLAB currently outputs an error in this case) and possibly correct it.
2. Bi-gram tagging currently makes systematic mistakes on "gengles", considering them as nouns part of long noun groups. How to implement a rule such as : "there must be at least one verb in the incoming sentence"?
3. Try to obtain better phrasing by avoiding static chunk|chink decisions: let the same word could be seen as chink or chunk depending its nearest neighbour.
4. Try to estimate prosodic phrasing using bigrams (thereby avoiding predefined chinks and chunks). This will imply tagging the speech corpus in terms of phrase breaks, and try to deduce such phrase breaks from the part-of-speech of words. A variant is to obtain phrase breaks by training CARTs.
5. Compute a density of (real) concatenations (per phoneme) using automatically generated Genglish.
6. Make statistics on speech unit usage and try to prune the speech corpus without too much affecting audio results (or concatenation density).
7. Change brute-force n-gram into viterbi-based n-gram (this is an excellent exercise for writing a Viterbi algorithm) and compare their CPU load.
8. Create a Genglish generator using Genglish n-grams
9. Find sentences for which n-grams fail and understand why.
10. Modify selection weights (especially on phrase and word counts) and find sentences for which this has an effect.
11. Try to enrich unit features, by using syllabic stress (for the target cost) or pitch (for the concatenation cos), for instance.
12. Create a Genglish-like word generator, having the same graphotactics (i.e. character sequence statistics) as Genglish, and submit new words to the phonetizer.
13. Why do some sentences taken from the corpus (like 'The gengle of gengles of the gengle is gengly gengly the gengle of the gengle of gengles of the gengle') produce a sequence of units that are not consecutive in the speech corpus? (unlike another sentence from the corpus, like 'Gengles are gengly the gengle of gengle.') ?
14. …

8. REFERENCES

ALLEN, J., S. HUNNICUT, & D. KLATT, (1987). *From Text To Speech, The MITALK System*, Cambridge University Press: Cambridge.

BADIN, P., BAILLY, G., RAYBAUDI, M., AND SEGEBARTH, C. A three-dimensional linear articulatory model based on mri data. In Proceedings of the Inte*rnational Conference on Speech and Language Processing*, volume 2, pages 417-420, Sydney, Australia, november 1998.

BLACK A.W. AND A.J. HUNT. 1996. "Generating F0 contours from ToBI labels using linear regression". *Proceedings of the International Conference on Speech and Language Processing (ISCLP'96)*, 1385-1388. Philadelphia, USA.

BLACK. A.W., and P. TAYLOR. (1997). "Festival Speech Synthesis System : system documentation", *Human Communication Research Centre Technical Report HCRC/TR-83*.

BOITE, R., BOURLARD, H., DUTOIT, T., HANCQ, J., LEICH, H., (2000). *Traitement de la Parole*, 2nd Edition, Presses Polytechniques Universitaires Romandes: Lausanne.

CHEN S. F., GOODMAN J. T. (1998), "An Empirical Study of Smoothing Techniques for Language Modeling", Technical Report TR-10-98, Computer Science Group, Harvard University.

BEAUFORT R., COLOTTE V., (2004). « Synthèse vocale par sélection linguistiquement orientée d'unités non-uniformes : LiONS », Proceedings of JEP 2004.

BOEFFARD, O., and C. d'ALESSANDRO, (2002). "Synthèse de la parole", in *Analyse, synthèse et codage de la parole, traitement auomatique du langage parlé 1,* J. Mariani, Editor, Paris : Hermès, Lavoisier.

DAELEMANS, W., and A. VAN DEN BOSCH, (1993), "TabTalk: Reusability in data-oriented grapheme-to-phoneme conversion", *Proceedings of Eurospeech 93,* Berlin, pp. 1459-1462.

DAMPER, R.I., Ed., (2001), *Data-Driven Techniques in Speech Synthesis*, Kluwer Academic Publishers: Dordrecht.

DELIGNE, S., and F. BIMBOT, (1995), "Language Modeling by Variable Length Sequences : Theoretical Formulation and Evaluation of Multigrams", *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing 95¸* vol. 1, pp. 169-172.

DUTOIT, T., (1997). *An Introduction to Text-to-Speech Synthesis*, Kluwer Academic Publishers: Dordrecht.

FLANAGAN, J.L., ISHIZAKA, K., SHIPLEY, K.L, (1975). "Synthesis of Speech from a Dynamic Model of the Vocal Cords and Vocal Tract". *Bell System Technical Journal,* 54, 485-506.

JELINEK, F., (1991), "Up from Trigrams!", *Proceedings of Eurospeech 91*, Genova, vol. 3, pp. 1037-1040.

KLATT, D. H., (1980). "Software for a Cascade/Parallel Formant Synthesizer", *Journal Acoustical Society of America, 67*, pp. 971-995.

KNUTH, D., (1973). *The Art of Computer Programming*, vol. 2, Addison-Wesley: Reading, MA.

LIBERMAN, M.J., and K.W. CHURCH, (1992), "Text Analysis and Word Pronunciation in Text-to-Speech Synthesis", in *Advances in Speech Signal Processing*, S. Furui, M.M. Sondhi, eds., Dekker, New York, pp.791-831.

LINDBLOM, B.E.F., (1989). "Phonetic Invariance and the Adaptive Nature of Speech", in B.A.G. Elsendoorn and H. Bouma eds., *Working Models of Human Perception*, Academic Press, New-York, pp. 139-173.

MACON, M.W. 1996. "Speech Synthesis Based on Sinusoidal Modeling", *Ph.D. Dissertation*, Georgia Institute of Technology.

MALFRERE, F., T. DUTOIT ad P. MERTENS. 1998. "Automatic prosody generation using suprasegmental unit selection". *Proceedings of the 3rd ESCA/IEEE International Workshop on Speech Synthesis,* 323-328. Jenolan Caves, Australia.

MALFRERE, F., DEROO, O., DUTOIT, T., RIS, C. 2003. "Phonetic alignement : speech-synthesis-based versus Viterbi-based", Speech Communication, vol. 40, n°4, pp. 503-517.

MARKEL, J.D. and A.H. GRAY. 1976. *Linear Prediction of Speech*. New York, NY: Springer Verlag.

MOEBIUS, B., M. PAETZOLD and W. HESS. 1993. "Analysis and Synthesis of German F_0 Contours by Means of Fujisaki's Model". *Speech Communication,* 13, 53-61.

MOULINES, E. and F. CHARPENTIER. 1990. "Pitch Synchronous waveform processing techniques for Text-To-Speech synthesis using diphones". *Speech Communication*, 9, 5-6.

SONDHI, M.M., SCHROETER, J., (1997). "Speech production models and their digital implementations", in *The Digital Signal Processing Handbook*. New York, NY: CRC and IEEE Press.

SPROAT, R., Ed., (1998). *Multilingual Text-to-Speech Synthesis – The Bell Labs Approach,* Kluwer Academic Publishers: Dordrecht.

STYLIANOU, Y. 1998. "Concatenative Speech Synthesis using a Harmonic plus Noise Model". *Proceedings of the 3rd ESCA Speech Synthesis Workshop*, 261-266. Jenolan Caves, Australia.

van SANTEN, J.P.H., (1993), "Timing in Text-to-Speech Systems", *Proceedings Eurospeech 93*, Berlin, 1993, pp. 1397-1404.

ACKNOWLEDGMENTS

I would like to thank some of my Masters students at FPMs, who contributed to the tools mentioned in this document in several ways : Mathieu Jospin et Grégory Lenoir, who initiated the Matlab programming of simple CART trees, and Julien Hamaide and Stéphanie Devuyst, who worked on the n-gram tagger (and designed part of the Genglish training and test copora). I am also indebted to Laurent Couvreur, who segmented the Genglish speech corpus using the HMM/ANN tools at TCTS Lab. Last but not least, I would like to express my best thanks to Milos Cernak at Slovenská Technická Univerzita (Bratislava, Slovakia) for his work on NUU synthesis of Genglish (and for pushing me to finish this project !).