Chapter 3[[1]](#footnote-1)

DEVELOPED TEXT TO SPEECH SYSTEM

“All life is a concatenation of ephemeralities.”

*Alfred Edward Kahn (1917–2010)*

3.1 introduction

In this chapter, we will describe our TTS system’s architecture through which we have attempted to overcome the shortcomings of the reviewed systems. The contribution of this chapter is twofold: on the one hand, we contribute to the enhancement of the TTS NLP module (front-end) through the use and the development of three components: a diacritization system, a G2P converter, and a module for pauses prediction. On the other hand, we contribute to the DSP module’s improvement through a novel approach for unit-selection based on an inventory of non-uniform units. A particular focus is given to the choice of the base acoustic units adopted for this task. Furthermore, we propose a set of contextual pre-selection filters to optimize the time complexity of the unit-selection algorithm by rejecting the database units that do not match the right and the left context of the target unit. Finally, we discuss the features we have adopted for the cost function calculation and the use of a phone penalty to maximize the cost of units likely to produce artifacts at the concatenation points.

punctuation provided with the input text. It is, therefore, necessary to address this limitation in this work by implementing a module that deals with implicit pauses (punctuation) and phrase breaks. These are addressed in Section 3.2.2.

Furthermore, building a phonetization module requires a thorough study of the most relevant phonological rules. In the literature, different rules are employed, including those related to classical Arabic (e.g., *Tajweed* rules). Yet, in this work, we only address the most important rules that comply with the MSA speaking style. The last implemented module of our TTS system front-end performs the extraction of linguistic and prosodic information, which we will use to build the feature vector of each phonetic unit.

3.2.[[2]](#footnote-2) *Diacritization*

Before investigating the choice of a diacritization system, we first need to define the criteria that we should consider for making this choice. The most crucial parameter of a diacritization tool from a TTS perspective is the level of diacritization, which ranges between full, half, or partial depending on the number of diacritic marks presented on a word (Mohamed Attia Mohamed Elaraby, 2000).

The first mode implies that all the letters are assigned appropriate diacritics. In the *half-diacritization* mode, only the morphological-independent letters are diacritized, while the diacritics of letters that depend on the syntactic analysis are dropped. For example, with the half diacritization, the word "?xDh" èY g @ (meaning "he took it") is diacritized as "?axaDh" èY g @. In the last mode, a single letter or a subset of letters are diacritized. For example, Sokun or vowels following long-vowels with a similar sound, as in "?xDu:h" èðY g @ which is partially diacritized as "?axaDu:hu" èð Y g @, since the omission of diacritics in such cases does not affect the pronunciation of the word, we assume that the full and partial modes are the most appropriate for a TTS application.

With these criteria in mind, we have filtered some diacritizers from many conducted surveys, which have guided our decision. In (Hamed and Zesch,

[[3]](#footnote-3)017) and (Fadel et al., 2019), empirical comparative studies were conducted on different available systems, among which are the relatively powerful tools MADAMIRA (Pasha et al., 2014), Farasa (Abdelali et al., 2016), and Mishkal (Zerrouki, 2014). Both works agree that Farasa outperforms MADAMIRA. Mishkal, however, has shown a higher error rate (using the Diacritic Error Rate (DER) and the Word Error Rate (WER) measures) in both studies. Besides, other diacritization tools are also available, for example, the ArabicDiacritizer1 and Ali-Soft2.

Yet, they were not considered for this decision due to the excessive amount of issues (e.g., limited input text size, integration incompatibilities).

For the reasons outlined above, we have decided to adopt the Farasa diacritization tool, which allows for a simple integration as a library in Java-based applications or through a RESTful API implemented for different programming languages.

3.2.2 *Pauses prediction*

In natural speech, besides the pauses forced by the punctuation marks, speakers tend naturally to incorporate breaks in an utterance. This phenomenon is increasingly perceived for long sentences, which generally lead the speaker to run out of breath. Besides, the speaker may introduce some pauses to express hesitation or emphasize a segment of the sentence. All of these scenarios characterize natural speech, which, when ignored, could lead to a monotonous speech or an ambiguity in the sentence meaning.

In the context of TTS, the automatic insertion of pauses is not less important than the other prosodic features such as pitch, intensity, and duration modeling. Pauses, along with the other features, contribute to the prosody, which is strongly correlated with the naturalness of the synthesized speech.

Overall, the pauses inserted by the TTS system are related to two kinds of breaks: punctuation-based pauses, also referred to as breath groups, and phrase boundaries or small breath groups. In this thesis, both types are addressed.

3.2.2.1 *Punctuation-based pauses prediction*

Let’s consider the following example:

He said Bilal had got a bachelor.

The above sentence accepts two possible meanings:

1. He, said Bilal, has got a bachelor.
2. He said: Bilal has got a bachelor.

For these examples, we can perceive the crucial role of prosodic pauses indicated by punctuation. In a typical system, the text’s punctuation is the most reliable indicator of pause location (Huang et al., 2001).

It is to note that, at this stage, we assume that a normalization phase was previously carried out to resolve the abbreviations and special symbols that might be mistaken for a punctuation mark (e.g., periods and commas in the number 20.000 or 01,12).

The first task performed by our pauses prediction module is the replacement of each punctuation mark with its corresponding tag. With this in mind, we have defined a set of common pause tags depending on the average silence duration that they may require. This classification was based on a statistical study that we have conducted on a transcribed speech corpus to locate all the punctuation marks and calculate the average duration of each of them.

The speech corpus sample used for this study was extracted from the corpus built in this thesis (Chapter 5). The sample comprises 300 sentences delimited by a period, question mark, or exclamation mark, among which 188 declarative and 112 interrogatives with an average size of 12 words per sentence. The sentences were read by the speaker at an average rate of 13 phonemes/second. The study resulted in an analysis of a total of 1,013 punctuation marks. The results are shown in Table 3.1.

Based on these results, we set out to define a single tag for punctuation marks that have relatively close duration. Consequently, two pause tags were defined as follows:

*<SP>* for a short pause represented by the punctuation marks: commas, parentheses, colons, and all kinds of quotes. *<LP>* for a long pause represented by the punctuation marks: periods, ellipsis points, semicolons, and exclamation marks.

Apart from this, the tag *<QS>* is assigned to question marks as they convey essential information about the sentence prosody, which shouldn’t be mistaken with other punctuation marks.

Besides, based on a close analysis of the spectral properties and the *F*0 shape of the different sentence types, it appears essential to consider the kind of questions that accept "Yes" or "No" answers as a distinct type of interrogative sentences since they are characterized with a rising *F*0 at the end of the sentence, as shown in Figure 3.1. For this, the tag *<QSYN>* is particularly assigned to sentences from this class.

3.2.2.2 *Phrase boundary prediction*

Besides the pauses marked with punctuation marks, an additional pause type can occur within a breath group, i.e., between two punctuation marks, to delimit a group of grammatically compatible words. The process by which these group boundaries are defined is called "Phrasing." This is typically addressed in the literature by using rules-based techniques.

For the Arabic language, very few works have addressed the phrasing in TTS. For example, in (Baloul and Baudry, 2003), the author has formalized a set of rules that predict the position of pauses within a breath group based on a test corpus of 168 sentences. Through this corpus, the author has investigated the correlation between types of pauses and grammatical group boundaries. The study resulted in rules guided by the type and nature of the grammatical group (verbal, subject, direct, indirect) while taking into account the min and max number of syllables since or before the last or the next pause. Nonetheless, this model is far from being generalist considering the size and type of the analysis corpus. The same conclusion has been stated by the author of this work.

According to (Huang et al., 2001), these decision-tree-based systems can achieve up to 81% in pauses accuracy. On the other hand, higher accuracy was achieved with data-driven techniques based on a large amount of data annotated at different levels (POS, punctuation, etc.)(Braunschweiler and Chen, 2013; Sarkar and Rao, 2015).

In this work, before thinking of proceeding to the preparation of a test corpus, which is usually a costly task, to conduct a data-driven study, we launched several tests with available syntactic parsing tools such as MADAMIRA, with the intention of using them to get a first parsing level. However, the latter was considered very minimal (i.e., a word-level parsing), requiring the inclusion of other factors to group the sub-chunks under a single syntactic segment.

Apart from this, the Arabic RGB, a syntactic dependency parser, was evaluated for the same purpose. The tool is developed by the Natural Language Processing group at the Massachusetts Institute of Technology (MIT). It was adapted to the Arabic language by the same authors in collaboration with the QCRI and using data trained with the Farasa toolkit. The tool is distributed in a java opensource[[4]](#footnote-4)version or through the Demo page of the Farasa website [[5]](#footnote-5). The implemented approach is based on tensor decomposition and greedy decoding, (Zhang et al.,2014).

The output of this tool was compared with the Stanford dependency parser for the Arabic language. As a result, the former seemed to correlate more with the locations of the prosodic boundaries (pauses). It was therefore adopted to serve as the first level of parsing. Figure 3.2 shows an example of the dependency parsing using the Arabic RGB tool provided by Farasa.

The dependencies are identified with *Head IDs* between the current word and the word on which it depends. In our pause prediction module, the output of the dependency parser is processed item by item (the items here refer to the words of the output sequence). Based on the *Head ID* of each item, the module decides whether or not a phrase boundary will be placed before the item. Indeed, a pause tag *<IP>*, for an Internal Pause, is only inserted if the current item is not dependent on the previous nor the following item, i.e., if the current item’s ID is different from its *Head ID +1* and *HeadID -1*.

Accordingly, using the same example presented in Figure 3.2, the output of the pause prediction module is as shown in Figure 3.3.

Moreover, based on our analysis of samples from audiobooks (a total of around 1.20 hours of speech), we have defined a set of *parsing words* after or before which the speaker usually introduces a short pause. Examples from these words are presented in Table 3.2. Post-break words refer to words occurring after a phrase boundary, and Pre-break words represent those occurring before a phrase boundary. Some words are redundant in both categories since they occur before and after a phrase boundary. The complete list of words is presented in Appendix II.

Considering these new rules, the decision regarding the placement of the *<IP>* tag will be conditioned by the defined set of words, as shown in Figure 3.4.

After running several tests with the implemented phrase boundary prediction module, we have noticed that the insertion of some *<IP>* tags was forced by the parsing words. These latter seem to perform well when punctuation is missed or when the output of the dependency parser causes the model to ignore a potential pause. An example of such cases is presented in Figure 3.5. The red <IP> tags are inferred by the RGB dependency parser. The blue tag is forced by the parser

3.2.3 *Grapheme-To-Phone*

As mentioned in Chapter 2, the G2P process is generally conducted either through a direct mapping or through a dictionary lookup for non-regular languages such as English and French, where pronunciation irregularities cannot be controlled through a finite set of rules. In contrast, for the Arabic language, being a phonemic language, the challenge of irregular pronunciations is less acute and is generally addressed through an exception dictionary of a finite number of words. However, another challenge is raised in the Arabic language; The pronunciation of a phoneme may vary depending on the context in which it occurs. These are mainly related to the co-articulation effects between sounds of the same word or between words of a given phrase. Fortunately, phoneticians and linguistics have already circumvented this problem in earlier works. This approach is referred to in the literature as rule-based phonetic transcription.

In this work, we have adopted the rule-based approach for building the G2P module to be incorporated in the front-end of our TTS system. Accordingly, we have collected a set of phonological rules from different works to make the rules set as comprehensive as possible, along with an exception dictionary. These are thoroughly described in this section.

3.2.3.1 *Phonetic transcription*

The rules reported in this section are mostly collected from a number of reference linguistic works (Al-ghamdi, Al-Muhtasib, and Elshafei, 2004; Alghmadi and Abdulaziz, 2003; El-Imam, 1989; Watson, 2007). However, as we stated earlier, the implemented set of rules is by no means exhaustive as we have only selected the most relevant rules that are commonly employed in MSA speaking style.

1. **Transcription of irregular words**

The first step in the letter-to-sound conversion is the processing of irregular words. These are defined as words whose pronunciation cannot be predicted from a direct one-to-one correspondence between letters and sounds, nor from the phonological rules.

The standard approach to deal with these exceptions in MSA is to build a lexicon of this class’s words along with their pronunciations. Therefore we have collected a list of irregular words from different sources to make it as exhaustive as possible. These are listed in Table 3.3.

1. **Grapheme-to-phoneme conversion** This step aims at performing the graphemeto-phoneme conversion that does not imply the use of phonological rules. Rather, the graphemes are inspected one by one and mapped to their corresponding phonemes, taking into account their context. The grapheme-tophoneme rules are implemented in the indicated order as follows: • Replacing the Arabic nunation mark called *Tanween* with its corresponding phoneme.

3. **Phoneme-to-phone conversion** The output of the previous step, which is a sequence of phonemes, is used as input to this module which performs the mapping from phonemes to phones. As discussed earlier in Section 2.3.1.2, this module uses a set of phonological rules collected from the literature (Al-ghamdi, Al-Muhtasib, and Elshafei, 2004; Alghmadi and Abdulaziz, 2003; Watson, 2007) to predict the actual pronunciation of a phoneme influenced by its context.

In MSA, the influence of sounds on each other results in different allophones of the same phonemes. The cases affected by this transformation are classified in MSA into two phenomena: Assimilation and Hamza rules. Based on a number of linguistic and phonological works(Al-ghamdi, Al-

Muhtasib, and Elshafei, 2004; El-Imam, 2004), we have collected and implemented these rules. Besides, we have included an additional set of rules which have demonstrated a noticeable improvement in the synthesis quality.

• **The assimilation**:

In MSA, the assimilation occurs in different cases (e.g., in a nasal context, before stop alveolar consonants, etc.). However, most Arabic assimilation rules are not compulsory in MSA; they are generally employed in classical Arabic and are referred to as *Tajweed* rules. Therefore, as far as we are concerned, we have merely implemented the (È@) assimilation rule, being mandatory in MSA. Nonetheless, we do not completely ignore the other assimilation cases; they are implicitly taken into account by the implemented unit-selection algorithm, which selects the target speech units with regards to their phonetic context (preceding and following diphones). We have implemented the assimilation rules as follows:

* If the definite article is followed by an alveolar or dental letter, also referred in Arabic phonology to *Sun Consonant* (SC) (see Table 3.4) and is phrase-initial:

/?al/ + SC → [?a] + SC

For example, Ò Ë@ /?aSSamsu/ → [SSamsu] (meaning "the sun").

* If the definite article is followed by a SC and is phrase-median or final:

/?al/ + SC → SC

For example, ÒË@ áÓ /mina ?aSSamsi/ → [mina SSamsi] (meaning "from the sun").

* If the definite article is followed by a *Moon consonant*(MC) (see Table 3.4) and is phrase-initial:

/?al/ + MC → [?al] + MC

For example, ÈA®ÜÏ@ /?alomaqa:l/ → [?almaqa:l] (meaning "the article").

* If the definite article is followed by a MC and is phrase-median or final:

/?al/ + MC → [l] + MC

For example, ÈA®ÜÏ@áÓ /mina ?alomaqa:l/ → [mina lmaqa:l] (meaning "from the article").

* **Hamza rules**:

/?i/) is generally addressed in the literature with few limited rulesDealing with the different pronunciation variants of Hamza (@ or @

that do not deal with all aspects of this phoneme. Thus, to make these rules as comprehensive as possible, we have defined additional rules based on the analysis of different speaker’s pronunciations. Accordingly, we have identified two main cases:

* 1. Hamza is pronounced if it is phrase-initial (indicated in the fol-

lowing rule with ˆ ) or preceded with a consonant:

ˆ /?i/ → [?i]

C + /?i/ → C + [?i]

For example, the word XAJ¯@ /?iqtis’a:d/ is pronounced [?iqtis’a:d] as it appears at the beginning of a phrase. Similarly, the word

XAJ¯B@ /?al?iqtis’a:d/ is pronounced [?al?iqtis’a:d] as it is preceded with the consonant /l/.

* 1. Hamza is dropped if preceded with a vowel:

V + /?i/ → V

For example, the word XAJ¯AK. /bi?iqtis’a:d/ is pronounced [bi?qtis’a:d] as it is preceded with the vowel /i/.

Note that the last rule includes the cases where the word is prefixed (e.g. /fa/, /ka/, /bi/, and /li/).

* **The confluence of two unvoweled letters**

In Arabic phonology, if a word ends with a *Sakin* which refers either to one of the long vowels /a:/, /i:/, or /u:/, or an unvoweled consonant (*Sokun*), and is followed by a word starting with a *Sakin*, usually the definite article /?al/ or Hamza /?i/, the first silent letter is omitted. Although this rule is broadly known and included in many works, it is not properly implemented, and thus many cases of this rule are generally not covered. Therefore, we have defined a set of rules through which we attempted to cover all the possible scenarios of the silent letters’ phenomenon.

**–** A long vowel (LV) is shortened if it occurs at the end of a word and is followed by a word starting with either Sokun or geminated consonant (CC) (usually resulting from the assimilation of the definite article /?al/). This rule is implemented as:

LV + CC → SV (CC is either a geminated consonant or a Sokun

consonant)

(For example, the wordmeaning "in the house"I). J.Ë@ú ¯ /fi: lbajti/ is pronounced [fi lbajti]

* + - * + The short vowel *Fatha* is inserted after an unvoweled /n/ if the latter occurs at the end of a preposition and is followed by a word starting with either Sokun or geminated consonant. This rule is implemented as:

Prep/n/ + CC → [na] + CC

For example, the word I J.Ë@áÓ /min lbajti/ is pronounced [mina lbajti] (meaning "from the house").

* + - * + The short vowel *Damma* [u] is inserted after an unvoweled /m/ if the latter occurs at the end of a plural word (pronoun or verb) and is followed by a word starting with either Sokun or geminated consonant. This rule is implemented as:

Plural/m/ + CC → [mu] + CC

For example, the word øQåJ.Ë@ÑêË /lahum lbuSra:/ is pronounced [lahumu lbuSra:] (meaning "For them are good tidings").

* + - * + If none of the previous rules apply, and a consonant (C) occurs at the end of a word followed with a word starting with either Sokun or geminated consonant, the short vowel Kasra [i] is inserted at the end of the first word. This rule is implemented as:

C + CC → C + [i] + CC

For example, the word H.AJºË@Y g /xuD lkita:ba/ is pronounced [xuDi lkita:ba] (meaning "take the book").

It is important to note that these rules are compiled in the mentioned order.

3.2.4 *Labeling*

In a typical unit-selection based system, the output of the front-end of a unitselection TTS system is a sequence of linguistic and prosodic feature vectors that will be passed as an input to the back-end block, which uses these parameters to calculate the join cost between each target unit and its corresponding candidate unit.

The extraction of this features vector generally comes after a syllabification (in case of syllables units) or a segmentation phase, resulting in a sequence of tokens, each of which will be subsequently labeled with its corresponding linguistic and prosodic features vector. In this thesis, we refer to this concept as the *labeling*.

3.2.4.1 *Phonetic segmentation*

The segmentation of a phonetic utterance consists of splitting the text into a sequence of phonetic units that match the units available in the speech database. Basically, this step should be performed after defining the base units. Nonetheless, since this decision will be discussed in the DSP bloc (Section 3.3.2), it seems necessary to briefly describe the base units adopted in this work based on which the segmentation will be conducted. In this work, two types of segments are favored:

* The **lemma-like unit** refers to the sequence of phones corresponding to a lemma stripped of its last short/long vowel (e.g., [Dahaba] → [Dahab]

(bic graphemeI.ëX , meaning "he went")) or final phone /h/ that corresponds to the Ara-è in case of feminine nouns (e.g., [?imka:niyyah] → [?imka:niyy]

(éJ K A¾Ó@ , meaning "the ability")). This modification of the basic Arabic lemma

aims at favoring the coverage of several words, including those in inflected forms.

* The **transition segment** is simply the sequence of phones from the last phone of a lemma-like unit to the first phone of the next lemma-like unit.

Basically, conducting a lemma-like-based segmentation implies extracting the lemmas corresponding to each word of the input text using a lemmatization tool.

For this, we have adopted the AlKhalil2 lemmatizer (Boudchiche et al., 2017). This decision was motivated by the appealing characteristics of this tool compared to other available systems. We note especially its availability and its acceptable accuracy. Another important feature that has motivated our choice is the diacritized output provided by this tool which is lacking in the other evaluated systems.

The lemma-like based segmentation is conducted as follows:

* AlKhalil lemmatizer takes the input text (as a sequence of graphemes) and outputs an array of words and their corresponding unique diacritized lemma tags in Arabic graphemes and Buckwalter scheme. If the system fails to find the lemma of a word, it returns a null for the word in the matter.
* The output of the lemmatizer is first converted from Buckwalter to SAMPA alphabet. The resulting sequence of SAMPA phonemes is then converted to a sequence of context-sensitive phones using our phonetization module (Section 3).
* In accordance with the type of the base unit, the phonetic sequence of each lemma resulting from the previous step is stripped of its final short vowel or final phone /h/.
* The final step before the segmentation consists of checking if the lemmalike resulting from the previous step is a sub-string of the word to which it is assigned. If it is, the segmentation module splits the word in matter into a lemma-like and clitics (e.g. [fa?imka:niyya:tuhu] → [fa] [?imka:niyy] [a:tuhu] (éJ K A¾ÓA ¯ , meaning "his ability")). Otherwise, if the lemma is not a substring of the word, it will be ignored by the segmentation module.
* Besides, words for which the lemmatizer did not return any lemma (null value) are also ignored by the segmentation module.
* Once the lemma-like boundaries are defined, the last step in the segmentation is to concatenate the phone sequences extending from the last phone of a lemma-like to the first phone of the next lemma-like unit. This step aims at defining the transition segment boundaries.

Besides, to avoid concatenation of units at the transition points between phones, which usually leads to audible artifacts at the joining points, the phonetic segmentation module splits the last phone of the lemma-like unit into two subphones and refers the second half of the last phone to the next unit. The same applies to the first phone, which is split into two sub-phones, where the first half is joined to the preceding unit. For example, the word sequence [xaraZati lfata:tu] (èAJ® Ë@Ik.Qk , meaning "the girl went out") will be segmented as [xaraZ] [Zati-lf] [fata:t] [tu].

The different steps described above are summarized in Figure 3.6.

3.2.4.2 *Features vectors extraction*

Once the utterance is segmented, the labeling module processes each unit and assigns a feature vector to it, which will be subsequently provided as input to the DSP block.

Different features are used in the literature to describe the target unit sequence. The extracted descriptors are usually related to the left and the right context, syllable stress, position in the phrase, POS, sentence modality, etc.

Nonetheless, other works consider a set of predicted acoustic and prosodic features such as pitch contour, phonemic duration, spectral envelope (e.g., MFCC), etc. These are generally predicted through trained statistical models (trees-based or DNN-based). For example, in (Alías, Formiga, and Llorà, 2011), the authors extracts the normalized pitch, the energy and duration. In the same vein, an earlier work conducted by Hifny and Rashwan, 2002 who presented a neural network build to predict the duration of Arabic phones based on the phonological representation of the target sequence.

Despite the popularity of this method, referred to as Hybrid synthesis, it still suffers from several limitations, and the prediction error is still high, thus leading to a degradation of the synthesis quality.

In our implementation, we enrich the linguistic and phonological descriptors with relevant and more comprehensive linguistic and prosodic features. For each phonetic unit constituting the target utterance, the labeling module builds a feature vector by answering the questions shown in Table 3.5.

Moreover, in this work, we do not predict any prosodic features. Instead, these are replaced with an implicit prediction based on the captured linguistic features. Besides, we assume that the continuity of *F*0 will be ensured by the join cost. Our challenge now is to investigate the linguistic features that capture the contextual information relevant to the prosody accurately.

Indeed, according to (Baloul and Baudry, 2003), there is a strong correlation between the sentence type (declarative, interrogative, and exclamative) and the intonation. At the spectral level, the intonation is determined by the *F*0 contour,

which varies with the sentence type.

Therefore, in this thesis, instead of investigating the prediction of the *F*0 contour, the prediction of the prosody is based on the sentence type, in addition to the position of the word in the phrase and the position of the unit in the word, which are found to correlate with the prosody of speech.

As for the lexical stress feature, it is commonly known that the stress depends mainly on the length of the syllable and its position within the word. As far as we are concerned, we hypothesize that predicting such information won’t have a considerable interest as our base units consist of large units beyond the syllable’s length. Rather, annotating the units with contextual information (left and right context) and the unit’s position would play a similar role to the stress features. This hypothesis remains, nevertheless, to be empirically endorsed in a future study.

A concrete example of a unit feature vector is illustrated in Figure 3.7.

3.3 digital signal processing (back-end)

In the previous section, we have presented the different components of our TTS front-end. This section will focus on the second bloc of the TTS pipeline: the TTS back-end.

As discussed earlier, in this thesis, we have decided to adopt the unit-selection synthesis approach to build our back-end bloc. The latter takes the sequence of feature vectors - will be referred hereafter as the target sequence, corresponding to the text to synthesize and inferred by the front-end bloc (NLP block). The aim is to find the best matching sequence of speech units - referred to as the **candidate sequence** - from a database of annotated speech.

In general, the unit-selection bloc accepts two inputs: the acoustic units of the speech corpus and the sequence of linguistic and prosodic features provided by the TTS front-end. Nevertheless, the corpus should be seen as a completely interchangeable brick. Therefore, a temporary test corpus was first used to conduct preliminary experimentation during the development of the TTS system.

This decision was motivated by the fact that the design of a high-quality speech corpus should be defined based on the requirements of the suggested unit-selection approach. Accordingly, it would be appropriate to use a test corpus as a first step to conduct function experiments of the developed system. Therefore, based on the preliminary tests conducted on the developed TTS system using the test corpus, we will define our synthesis approach requirements, upon which the final speech corpus will be built.

3.3.1 *The test corpus*

Building a large inventory of speech units is a laborious task that requires a strict recording procedure and long hours of recording sessions. Therefore, as long as the TTS system is not built yet, a definitive corpus should not be investigated. Consequently, we have decided to use a pre-recorded *Audiobook*.

Masmoo3[[6]](#footnote-6) is a collection of expressive audiobooks recorded in natural prosody by a professional male speaker. The selected *Audiobook* has a mean F0 of nearly 88 Hz on voiced segments. The corpus built from the *Audiobook* contains 1,148 sentences, mostly statements, and few interrogative sentences. The total recording length is 4 hours and 20 minutes of clean speech sampled at 44kHz, in mono (1 channel). The speech files were segmented into 98 files with an average file length of 2.5 minutes. The orthographic transcription of each text file was diacritized and checked with respect to the speaker’s pronunciation. Afterward, we performed manual text normalization to the entire transcription file by expanding numbers, signs, and abbreviations in the textual form. Furthermore, as we have spotted some spelling mistakes, manual spell checking was carried out to ensure the correct matching between the recordings and text spellings.

In accordance with our unit-selection approach, the speech files were split and annotated at lemma-like and diphone levels. The segmentation and annotation were both performed manually in Praat software (Boersma and Weenink, 2000); lemma-like unit boundaries were placed at the center of the first phone of the lemma to the center of the last phone of the same lemma. Table 3.6 presents the main statistics of the used *Audiobook* corpus.

3.3.2 *The Base Unit*

One of the key challenges in concatenative synthesis is defining the base unit. The nature and the size of this latter have a decisive effect on the naturalness of the synthesized speech.

In chapter 2, we have discussed this point, showing that the units used in the literature vary from small units (half-phones, phones, and diphones) to larger ones such as triphones, half-syllables, syllables, words, and sometimes entire sentences for domain-specific applications.

In general, two conflicting facts guide the choice of the acoustic unit: A limited number of units is preferred to reduce the search space during the unit-selection process; and, on the other hand, the units have to be large enough to model the co-articulation dynamics (Peterson, Wang, and Sivertsen, 1958).

This compromise is commonly addressed with diphones that allow for coverage of co-articulation effects with a limited inventory size (less than two hours); thus, they are the most affordable for low-resource languages. Nevertheless, despite the advantage of requiring small footprints, the diphone can still inherit the drawbacks of short units, which only allow for coverage of local co-articulation variations. Besides, using diphones as a base unit may result in many concatenation points, thus causing audible artifacts.

It seems, therefore, evident that we cannot cope with the problem of spectral discontinuity and concatenation artifacts unless larger units are adopted. However, a significant number of units are needed to achieve the required coverage,

which would require tremendous effort and resources to build large speech corpora (up to tens of hours), especially for non-domain-specific TTS. Therefore, the choice of the base unit can be considered as a footprint/performance trade-off.

In this thesis, we have chosen to consider this trade-off by using larger units and dealing with the coverage problem. For this, we have investigated thoroughly the choice of a large unit that inherits the advantages of words as speech units - which are known to yield high naturalness of speech - without requiring an excessive amount of data. Consequently, the **lemma-like** speech unit was found to meet this goal. Moreover, units extending from a lemma-like to the following lemma-like unit, unless a pause is encountered, are also considered as favored units as they cover the transitions between adjacent words. This unit is referred to as the **transition segment**.

3.3.2.1 *Addressing the performance*

In this work, the lemma-like unit refers to the Arabic lemma stripped of its last short/long vowel or last consonant (for feminine words ending with TaeMarbuta). In the Arabic language, the lemma refers to the reduced canonical representation of a word before undergoing any inflection. The lemma for verbs is the masculine singular form without clitics. For nouns and adjectives, it corresponds to the nominative singular masculine (or feminine if the word does not accept the masculine) form without clitics. In the case of particles, it is the particle stripped off its clitics.

Our primary motivation for using this unit is based on the assumption that it represents the largest segment shared by a broad range of Arabic words. Hence, we hypothesize that adopting the lemma-like as a base unit would considerably reduce the number of concatenation points required to generate a given word sequence.

To further investigate this hypothesis, we have conducted a statistical experiment aiming to get the percentage of words that the top frequent 10,000 lemmas might cover in a random Arabic text. In this study, all the words made from a lemma-like plus clitics are counted as instances of their lemma.

Consequently, a list of lemma frequency bands and their coverage in the reference test corpora (≈ 4 million words) was compiled. The process of generating the lemma frequency dictionary will be described in detail in Chapter 5. As shown in Table 3.7, the study has revealed a high coverage ratio with a relatively small amount of lemmas; 95.54% Arabic word coverage can be achieved with the 10,000 most frequent lemmas, and 78.86 % with only 1,000 lemmas. In other words, a speech database of only 1,000 lemmas can be used for synthesizing more than 78% of the Arabic words. We are, however, aware that this number should be multiplied by at least ten if we foresee a multi-representation of the acoustic units in various contexts.

This conclusion drawn from this study was endorsed in another work (Mohamed Attia Mohamed Elaraby, 2000), where it was shown that fewer words yield greater coverage when counted as lemmas. This suggests that the Arabic language is highly inflected and derivative, and rules for generating new words from base words( i.e., lemmas) are applied extensively.

As for the acoustic representation of the base units, the different units of the acoustic database are split from the center of their first phone to the center of their last one. Accordingly, the lemma-like unit and the transition segment are extended from the second half of the first phone to the first half of the last phone of the same unit. This is to allow for a smooth concatenation at the steady state parts of the units’ boundaries.

Figure 3.8 shows an example of a sequence of lemma-like units **(2)** and transition segments **(1)** used to synthesize the sentence presented in **(3)**.

3.3.2.2 *Addressing the footprints*

Ensuring the full coverage of lemmas is not a trivial task, if not impossible, given their extremely large number in the Arabic language. According to (Namly et al., 2020), the number of lemmas in Arabic is over 164,272, to which we must add the different variants of each lemma that differ in terms of phonetic and prosodic features, such as the position in the sentence and the acoustic and phonetic parameters, among others. Therefore, we have decided to target only the top frequent Arabic lemmas. This decision systematically implies setting a frequency threshold (i.e., number of frequent lemmas to cover) to consider for coverage. However, this task will be deferred to the corpus design phase(5).

Apart from this, as there is no guarantee that a given lemma from the target sequence will be available in the speech database, we have implemented a backoff strategy to compensate for such missing units by using shorter units. This might also concern the non-lemma-sized units if the algorithm fails to find the whole phone sequence.

The back-off will be referred in the following section to as a the filter relaxation.

This process is described in detail in Section [[7]](#footnote-7).3.3.

that match the phonetic unit and are represented in the speech database with more than one instance (A Alabbad, 2019).

Moreover, most existing implementations that use pre-selection filters do not use any target cost, which is generally set to 0. This decision is mainly based on the fact that the linguistic and phonetic filters can efficiently replace the target cost. Nonetheless, the selection of the best path of candidate units will be based only on the join cost, which will likely cause a unit shortage for the selection algorithm, making the cost function mostly ineffective. For example, (Guennec, 2016) applies a set of pre-selection filters that include predicted features such as the *F*0 contour, in addition to other linguistic features (position in syllable, position in the phrase, etc.). However, the accuracy of the predicted features (namely F0 contour) is not guaranteed, resulting in a rejection of units that might have had better matching of *F*0 than those selected by the filter. Such cases, when they occur, might lead to drastic degradation of the synthesized speech.

Therefore, we hypothesize that the choice of units based on such parameters should be entrusted to the cost function, which will penalize the units based on the concatenation and target costs.

In our system, we have implemented a pre-selection module that applies filters to accept or reject a candidate unit by comparing its features to the target unit feature before adding its corresponding node to the search graph. In fact, we consider that there is no need to add the nodes failing to have a minimal fitting

with the target sequence in the graph.

With this in mind, we have defined four pre-selection filters assumed to be mandatory in every candidate unit; otherwise, it will not be integrated into the graph. On the other hand, the features deemed to be more flexible are left for the target cost calculation.

The pre-selection filters are defined as follows :

1. Is the segment an NSU?
2. What is the phonetic identity of the unit?
3. What is the following diphone?
4. What is the preceding diphone?

The filters (1), (3), and (4) are the most selective — that is, if a candidate unit from the database has different values for these features compared to the target unit features, the candidate unit is rejected. The filters (3) and (4), being the most important in this work, will be referred hereafter to as *context-dependent* (CD) filters.

For each unit from the target sequence, the pre-selection function creates a list of filters’ values represented by strings and binary values. Afterward, the algorithm looks up its corresponding filters’ values in a hash table containing the corpus units.

Suppose no item is returned, which means that none of the available segments corresponds to the unit’s filters. In that case, the phonetic identity filter (2) is relaxed by deleting the last diphone and attaching it to the next unit. If the diphone to be deleted contains a word transition marked with "-", the algorithm removes the whole segment starting from the phone before the transition to the end of the unit.

For example, for the target unit sequence:

if we assume that the second phonetic unit is not available in the speech corpus, thus, will require a back-off to shorter segments, the relaxation of the second filter (i.e., phonetic identity) would result in the following new unit sequence:

[?*iStar*][**rat**][**t-D**][*Dahab*][*ban*]]

This decision was based on the fact that a word transition represented with "-" is considered an atomic segment; therefore, it should not be segmented.

The above example shows that the relaxation process triggers an update of the target sequence with the new unit sequence and the corresponding labels (features concerned). The update affects mainly the current unit subjected to the relaxation and the following two units. All the labels attached to each unit are updated except the phrase level features. The algorithm resets the units’ preselection filters based on the updated labels before restarting the lookup of the new phonetic unit given its filters.

This process of relaxation followed by the lookup is repeated until the number of the retrieved candidates for each target unit is higher than 15, or if the length of the current phonetic unit subjected to the relaxation is equal to 2 (diphone).

The pre-selection algorithm inserts the matching candidate units in the search graph, which the selection algorithm will browse to find the best path based on the cost functions.

This mechanism ensures finding a path in all cases with a reduced search space and a reduced selection time.

3.[[8]](#footnote-8).4 *Selection Algorithm*

After the pre-selection function has performed an initial selection of candidate units based on their phonetic identity and their context, resulting in several candidate units for each target unit, the next task is to find the cheapest path representing the best matching unit sequence among the initially selected candidates.

As reported in the state-of-art (Chapter 2), Computing the best sequence is classified as a path-finding problem in a directed weighted graph (Guennec and Lolive, 2014). Therefore, the Viterbi algorithm (Viterbi, 1967) (and its derivatives), Bellman-Ford, Dijkstra and *A?* (Hart, Nilsson, and Raphael, 1968) are generally fitted for the task.

The former is the most widely used algorithm. However, as long as we include a pre-selection phase leading to reduced search space, the choice of the algorithm wouldn’t have a considerable effect on the system’s performance. For this purpose, the Viterbi algorithm was implemented.

The candidate units are organized in a graph *G* = (*V*; *A*; *C*) directed and ordered with the same order of the target phonetic sequence. V represents the nodes corresponding to the set of candidate units linked with a set of arcs A. Each arc represents a possible concatenation between two units. The mismatch between every pair of units is quantified with the cost *C* linked to each arc, referred in this work to as the join cost. Higher join costs can be seen as a higher risk of creating audible artifacts if the pair of units are concatenated.

Each node from the graph is associated with a feature vector corresponding to a candidate unit. All the feature vectors are pre-computed during the corpus construction and stored in a text file. Two classes of features are defined: acoustic and linguistic. The latter is used to calculate the target cost of each candidate unit with respect to the target unit. Whereas the former is used to calculate the join costs. It is to note that the linguistic feature vector associated with each candidate unit should have the same format as the feature vector assigned to the target units.Therefore, the linguistic features described in 3.2.4.2 are the same used for annotating the corpus speech units. Details about the corpus unit features will be described in Chapter 5.

We must recall that the candidate units in the matter are the ones selected by the pre-selection function — that is, the search will only be performed on a graph of pre-selected units without having to retrieve other units from the database.

As described in Chapter 2, throughout the search, target sub-costs are assigned to each node. A sub-cost of *Zero* related to a given feature *j* means that the candidate and the target unit have exactly the same value for *j*.

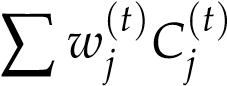
The linguistic and acoustic features extracted or predicted in the front-end are usually used to compute the target sub-costs (Section 3.2.4.2).

Nonetheless, the complete set of features is rarely provided in the literature, and the relevancy of these features for the target cost calculation is scarcely addressed, especially for Arabic, as we assume that these features are languagedependent. For example, (Black and Campbell, 1995) noted that the target cost is calculated using 30 sub-costs without providing any details.

In this work, the target sub-costs are calculated based on the features defined in 3.2.4.2 after excluding those used by the pre-selection function, namely the phonetic identity, the NSU, and the left and right diphones. The decision to use the defined set of features was based on a battery of tests. Indeed, to include only the most relevant descriptors that have a considerable contribution to the synthesized speech quality, we have started with a comprehensive set of features collected from the literature, and we gradually removed the features one by one from the target cost function. At each step, we assessed the resulting speech quality, whereby the decision on whether or not a given feature will be retained or rejected. The resulting set of features that we have deemed relevant for our system are the ones presented in Table 3.5.

For each node from the graph, distance functions *C*(*t*)*j*(*ti*, *ui*) are calculated between each feature *j* of a candidate unit *ui* and the target unit *ti*. The weighted sum of sub-costs results in the unit target cost *C*(*t*)(*ti*, *ui*):

*J*

*C*(*t*)(*ti*, *ui*) =(*ti*, *ui*) (3.1)

*j*=1

Another commonly addressed issue when defining target costs is the weighting of features. Indeed, some features have a stronger effect on the synthesis quality than others. Hence, a set of weights, either trained or hand-tuned using expert knowledge, is used to prioritize some features over others. These weights are mostly used with Independent Features Formulation (IFF) as defined in Festival Multisyn (Clark, Richmond, and King, 2007), which computes the target cost by summing binary sub-costs of a candidate unit depending on whether or not the candidate features match exactly the target unit features. Since this simplest formulation doesn’t include a "near match" concept, the need for weights that quantify the relevance of each feature arises.

Nevertheless, in our work, as the target cost is calculated based on numeric distances between features, all weights (*w*(*jt*) in equation 3.1) are set to 1. Additionally, we assume that all the features used for target cost calculation have the same relevance.

3.3.6 *Join cost*

The join cost aims at preventing the concatenation of units that might cause audible artifacts at concatenation points by assigning high costs to candidate units showing a significant mismatch with the target unit. This cost is added to the target cost upon which the choice of the least-cost unit sequence is made.

Many works have demonstrated that using merely join costs without a target cost calculation usually results in acceptable speech quality. In contrast, selection based on the target costs ignoring the join cost often leads to unintelligible speech.

Therefore, defining the relevant information for computing the join cost was given great attention in the literature. As stated in many works, the most important distance is *F*0 (Black and Campbell, 1995). Besides, the Euclidean distance of MFCCs around the joining point, the Cepstral distances, particularly the LPC (Linear Prediction Coefficient), are also investigated in many works (Alías, Formiga, and Llorà, 2011).

For Arabic synthesis, the *F*0, intensity, and MFCC are the most commonly investigated A Alabbad, 2019; Abdelmalek and Mnasri, 2016.

In our work, we have adopted four sub-costs:

* Mean F0,
* F0 at the unit boundaries,
* The amplitude,
* The duration of the half-phone at the unit boundaries.

These have been chosen as they are well-rated in the literature. We assume that these features are pre-computed for all the corpus units. Hence, we compute the distance between each unit and the following unit sub-costs. Using the formulation introduced in Chapter 2, the join cost assigned to each arc joining two subsequent candidate units is formulated as:

*C*(*j*)(*ui*−1, *ui*) = *Camp*(*ui*−1, *ui*)+*CMF*0(*ui*−1, *ui*)+*CBF*0(*ui*−1, *ui*)+*CPHdur*(*ui*−1, *ui*),

(3.2)

where *Camp*(*ui*−1, *ui*), *CMF*0(*ui*−1, *ui*), *CBF*0(*ui*−1, *ui*), and *CMPHdur*(*ui*−1, *ui*) cor-

responds to the amplitude, mean F0, F0 at the unit boundaries, and half-phone duration, respectively.

Nonetheless, these are still far from being representative of the perceptual speech parameters, and audible artifacts were still noticed after conducting preliminary tests with the defined costs. Therefore, in an attempt to locate and identify the source of artifacts, we have analyzed a number of sentences that included audible artifacts. The results have shown that most artifacts occur at the concatenation points between a pair of half-vowels, especially the short vowel /a/, which was highly influenced by the neighboring phones. These latter are mostly responsible for the modification of the vowel emphasis and duration.

Based on these facts, it seems that capturing the right and left phonetic context of the vowel are not enough. Many authors argued this issue concerning the spread of emphasis, leading to the conclusion that we cannot minimize the distortion between two concatenated half-vowels unless extracted from the same sequence (constituted with at least a word). This is, however, impractical if not impossible.

Another more optimal solution is to penalize units based on the nature of the last phone (Guennec, 2016). Consequently, we have empirically set a high penalty for the half-vowels /a/ and a less important penalty for the other halfvowels /u/ and /i/. As for the other phones, their penalties are all set to Zero. Accordingly, the new join cost function will be

*C*0(*j*)(*ui*−1, *ui*) = *C*(*j*)(*ui*−1, *ui*) + *P*(*ph*), ([[9]](#footnote-9).3)

where *C*(*j*)(*ui*−1, *ui*) is the join cost formulated with the equation 3.2 and P(ph)

is penalty of the last half-phone of the first unit *ui*−1, which is the same as the half-phone at the beginning of the unit *ui*.

The problem of finding the best sequence of units that minimizes the sum of candidate units’ total cost can be formulated as follows:

*n n*

*U* = arg min(∑ *C*(*t*)(*ti*, *ui*) + ∑ *C*0(*j*)(*ui*−1, *ui*)) (3.4)

*u*=*u*1,...,*un i*=1 *i*=2

It is to note that the calculation of the join cost *C*0(*j*) is calculated only for units from *u*2 to *un*.

An example of a generated graph for the target sequence *[?iStarat]* which is formed by the lemma-like *[?iStar]* and the transition segment *[rat]*, is shown in Figure 3.9. Shorter units are retrieved by the pre-selection module, which backs off to alternative segments if the threshold is not achieved. The "#" symbol denotes a silence tag. We define a start node "Init" and an end node "End" to avoid arbitrary choices of the first and last units. The start node is linked to all candidate units that match the beginning of the target sequence. The same applies to the end node, which is linked to candidate units that match the end of the target sequence.

3.3.8 *Units concatenation*

The output of the unit-selection function is the least-cost path corresponding to a sequence of candidate units that best matches a target sequence delimited by two pause boundaries <LP>.

3.4 conclusion

Provided that we consider the speech corpus as a whole speech band rather than a dictionary of pre-segmented speech units, the sequence of the selected units are only extracted at synthesis time.

Therefore, the last task before the signals concatenation is to retrieve each unit’s start and end times from the speech database based on its unique ID. The latter is indexed in an annotation file where all database units are listed along with their linguistic and acoustic information. The complete description of the corpus format will be presented in Chapter 5.

The concatenation function uses the start and end times to extract the unit from a long WAV file using the Praat toolkit. For consecutive units (having start and end times ), the concatenation function retrieves only the start time of the first unit and the end time of the last contiguous one, thus avoiding useless concatenations between adjacent units.

Besides, some works include a post signal processing phase which performs a set of prosody modifications over the resulting chain (Eide et al., 2003) or smoothing the joins between concatenated units extracted from different contexts. It is, however, argued that this signal processing usually leads to significant degradation of the resulting speech, as stated in state of the art. Therefore, after conducting preliminary listening tests on the synthesized speech produced by our TTS system, we have decided to skip the signal preprocessing phase as the resulting speech without any signal modification is already deemed acceptable.

3.4 conclusion

In this chapter, we have presented the different components of our TTS system.

We have presented the NLP block, through which we have discussed the suitability of the most popular diacritization tools considering the Arabic TTS requirements. Furthermore, we have suggested an approach for predicting the breath groups’ locations based on a dependency parser.Besides, we have reported a comprehensive set of phonological rules which were thoroughly discussed based on their importance. The selected rules were used to implement a G2P converter. To prepare the target sequence for the DSP block, we have defined a set of relevant features used for labeling the target phonetic sequence with linguistic and implicit prosodic information.

In the DSP block, we have suggested a novel base unit for unit-selection, which we defined as the ’lemma-like’. This unit was adopted as it represents the ideal compromise between performance and footprints, as we only target the most frequent ones, which were found to cover a broad range of Arabic words. A backing-off mechanism was also described to deal with missing units in the database.

3.4 conclusion

Figure 3.10: Workflow view of the developed TTS system.

As searching for variable-sized units in a large corpus is computationally expensive, we defined relevant pre-selection filters that prune the candidate units based on their minimal but most essential match with the target sequence context (surrounding diphones) before adding them to the search graph. Moreover, target and join cost functions were described. A particular focus was put on penalizing the phonemes that are more likely to produce concatenation artifacts.

1. .2 natural language processing (front-end)

   As discussed in 2.3.1, the typical front-end bloc (NLP) involves four principal components: a normalizer, a phonetizer (G2P), and a feature extraction module for building the linguistic and prosodic feature vectors.

   For the Arabic language, the use of a diacritization module is mandatory (Section2.6.1). This concludes that a diacritization module should be implemented and incorporated in the TTS system’s front-end. Nevertheless, there is no need to re-invent the wheel, as there are many available diacritization tools for the Arabic language. Instead, we should investigate the choice of an accurate diacritizer with respect to some factors (e.g., accuracy, availability, etc.). Moreover, in the previous chapter, we raised an issue related to the inappropriate use of pauses in most of the available TTS systems, which seem to rely only on the [↑](#footnote-ref-1)
2. https://sourceforge.net/projects/arabicdiacritizer/ [↑](#footnote-ref-2)
3. http://www.ali-soft.com [↑](#footnote-ref-3)
4. https://github.com/qcri/ArabicRBGParser [↑](#footnote-ref-4)
5. https://farasa-api.qcri.org/dependency/ [↑](#footnote-ref-5)
6. http://www.masmoo3.com/ [↑](#footnote-ref-6)
7. .3.3 *Pre-selection*

   In a typical unit-selection implementation, at synthesis time, the selection algorithm retrieves all the available units from the speech database that match the phonetic identity of the target unit sequence. Hence, the cost function will be computed for all the units before selecting the best path. This process is computationally expensive. The search complexity increases exponentially with a speech database of variable-sized units, making the search space larger.

   Therefore, hash tables and pre-selection filters are commonly implemented to optimize the size of the selection graph and hence the selection time (Beutnagel, Mohri, and Riley, 1999). Another approach consists of pre-selecting the units [↑](#footnote-ref-7)
8. .3.5 *Target cost*

   While the pre-selection function selects the matching units from the database based on a minimal set of filters, the target cost has to use more refined features to score the nodes (candidate units) based on their level of similarity to the target unit’s features. [↑](#footnote-ref-8)
9. .3.7 *Total cost*

   As stated in 2, the total cost of each node is the sum of its join cost *C*0(*j*)(*ui*−1, *ui*) and the target cost *C*(*t*)(*ti*, *ui*). With *ti* being *ith* unit in the target sequence *T* = *t*1, .., *ti*. [↑](#footnote-ref-9)