8th International Workshop on Semantic and Social Media Adaptation and Personalization

A Semantics-Driven Approach to Lyrics Segmentation

Adriano Baratè, Luca A. Ludovico, Enrica Santucci
Laboratorio di Informatica Musicale (LIM)
Università degli Studi di Milano
Via Comelico 39, 20135 Milano, Italy
{barate, ludovico, santucci}@di.unimi.it

***Abstract*—The purpose of this paper is describing a semantics-driven approach to the automatic segmentation of song lyrics. The proposed algorithm takes into account the basic formatting commonly in use for lyrics on CD booklets and specialized Web sites, in order to extract basic semantic information, such as the organization in lines and sections. Then the algorithm applies simple rules to reconstruct lyrics structure, supporting tolerance margins as regards possible errors and encoding variants. The output is a sequence of sections labelled according to the similarity of their contents. The resulting segmenter is publicly available as a set of methods exposed via a Web application programming interface (API).**

I. Introduction

In the digital era, huge and comprehensive collections of song lyrics are available to Web users. In some cases these contents come from authoritative sources such as music labels and music publishers; in other cases they are the result of collaborative efforts by users themselves.

The framework described in this paper addresses the problem of segmenting song lyrics. The primary goal is developing an algorithm that takes in input a set of text strings and - after some computation - outputs a set of symbols representing the recognized structure. The algorithm is a part of a more comprehensive framework that can be accessed from the Web via API[1] methods.

The idea is using the basic semantic information usually present in lyrics to drive the segmentation process. Hopefully, lyrics should be accurately transcribed and organized in lines and blocks through carriage returns and blank lines. Nevertheless the algorithm has to be in a certain measure fault-tolerant.

As discussed in more detail in Section III, in vocal music (above all in specific genres) the sections a piece is composed of can be clearly recognized and sometimes even detected *a priori*. For instance, in popular music a common form is *verse*, *chorus*, *verse*, *chorus*, *bridge*, *verse*, *chorus*.

The goal of the algorithm is not to label a set of lines in a given way (e.g. *verse*, *pre-chorus*, *chorus*, etc.), but rather identifying recurrent as well as non-recurrent groups of lines, no matter which structural meaning they have in the music piece. For example, from this perspective the structure cited before would become A-B-C-B-D-E-B, as the verse is typically a variable part whereas the text of the chorus is repeated.

This framework has heterogeneous theoretical and practical implications. For example, it can be employed for automatic large-scale investigation on song structures, as regards the verse-chorus layout as well as the internal composition of single sections.

Another application can be oriented to the segmentation of the corresponding audio track, supposing that fixed text parts are clearly recognizable in the audio signal too. For instance, this approach can be coupled with the one described in [1], where an input audio signal is segmented and then aligned with hand-labelled paragraphs in lyrics through dynamic programming algorithms to find the best alignment path. Other similar works will be cited in Section II.

Finally, it is worth citing a merely practical application for huge repositories of lyrics. One of the methods in our framework implements a string metric for measuring the difference between two character sequences. Even if the main purpose in this context should be finding string occurrences, repeated either literally or in varied form, this method could easily discover typographical errors (e.g. misspelled words) and unwanted variations (e.g. the adoption in similar lines of different apostrophes, such as ’ and ‘).

The paper is organised as follows:

- Section II presents a survey about other scientific works focusing this matter;
- Section III introduces the basic terminology and describes the main structures available for popular music;
- Section IV discusses which semantic aspects we can rely on when we parse lyrics archives;
- Section V explains the key design principles of the segmentation algorithm;
- Section VI declares the methods exposed via Web to analyse lyrics;
- Section VII finally discusses some clarifying examples.

II. Related Works

The problem of automatic lyrics segmentation has been already addressed in many works. In this section we review some of the most promising and relevant initiatives.

An approach can be found in [2]. In this case, standard natural language processing (NLP) tools are employed for the

[1] API is an acronym standing for Application Programming Interface. An API is a set of functions that accomplish a specific task or allow to interact with a specific software component.

978-0-7695-5132-6/13 $26.00 © 2013 IEEE
DOI 10.1109/SMAP.2013.15

analysis of music lyrics. Structure extraction occurs after a phase of language identification. The authors conducted experiments with lyrics from 5 different languages: Spanish, Italian, French, German and English. Unfortunately, this approach strongly relies on language detection and recognition tends to fail in cases where lyrics are written in more than one language (for example The Beatles' "Michelle") or with onomatopoeic or non sense words (for example Santana's "Jin-Go-Lo-Ba").

Another relevant approach is the one described in [3], which proposes a multimodal structure segmentation of music using both audio and textual information simultaneously. The semantic structure of songs is achieved by lyrics processing. The similarity between each pair of paragraphs is measured by the longest common subsequence (LCS), a sequence of matched words whose orderings is unaltered.

A powerful mix of bottom-up and top-down approaches, combining the complementary strength of low-level features and high-level music knowledge, is also presented in [4]. The evolution of this approach based on beat-space segmentation, chords, singing-voice boundaries, and melody- and content-based similarity regions can be found in [5].

As mentioned before, lyrics analysis can be profitably coupled with media segmentation, clustering, and similarity analysis. To this end, in the previous section we have already quoted [1]. It is worth citing also the early approach in [6] and more recent works such as [7], [8], and [9].

With respect to the mentioned works, this paper aims at improving the lyrics-related part by taking advantage of semantic information, as explained in Section V.

Finally, [10] proposes an original way to input both audio and lyrics information by using a robust Voice-to-MIDI system. The user can input MIDI sequence data by naturally singing melodies with lyrics, then the Voice-to-MIDI system translates singing voices into digital musical data, i.e., MIDI sequence data.

III. Song Structures in Popular Music

Music structures have been deeply investigated from different perspectives: Literature, Musicology, Music Composition, etc.

First, it is possible to recognize some recurring structures, dating back to ancient Greek poetry. For instance, *strophic form* (also called *verse-repeating* or *chorus form*) is the term applied to songs in which all verses or stanzas of the text are sung to the same music. A possible representation of its structure is A-A-A. Many folk and popular songs are strophic in form, including the twelve bar blues, ballads, hymns and chants. This simple structure was used also in classical art songs, such as 19th century German lieder. Relevant examples are some poems by Johann Wolfgang von Goethe set to music by Franz Schubert, e.g. "Heidenröslein" and "Der Fischer". Besides, several compositions in his song cycle *Die schöne Müllerin* use strophic form.

A song that uses the strophic form is "Bridge Over Troubled Water" by Simon and Garfunkel. Lyrics present three verses, each consisting of several lines. Each verse is lyrically different and there is melodic variation in the final verse. At the end of each verse the line "Like a bridge over troubled water" is repeated.

Another typical structure is *contrasting verse-chorus form*, a binary form that alternates between two sections of music (A-B-A-B). This kind of song form became predominant on the chart during the explosion of rock and roll in the 1950's and it is still in use. Examples are "Take It Easy" by The Eagles and "Un-break My Heart" by Toni Braxton.

An extension of the verse-chorus layout is the *verse-chorus-bridge structure*, which often uses this pattern: verse-chorus-verse-chorus-bridge-chorus. The first verse sets-up the theme of the song with the last line offering a natural progression to the chorus. The chorus contains the main message of the song that is worth repeating. Then another verse where new details are revealed followed by the chorus again. Then a bridge - often shorter than the verse - is added. The bridge must be different from the verse - lyrically and rhythmically - and offers a reason why the chorus needs to be repeated. Currently, most pop/rock songs adopt this structure, with a number of small variants.

There exists a correlation between music and lyric structure. As regards the latter, the set of structure types usually considered in popular music are:

- Introduction (intro): usually one verse composed by three or four phrases used to expose the main theme or to give a context to the listener;
- Verse: a part that roughly corresponds with a poetic stanza. Lyrics in verses tend to repeat less than they do in choruses;
- Pre-chorus: the optional song section that occurs before the chorus. Its main function is to act as a transition between the verse and chorus, to allow those two sections to connect together a little better;
- Chorus: the refrain of a song. When two or more sections of a lyric have almost identical text, these sections probably are instances of the chorus. This part repeats at least twice with none or little differences between repetitions, thus becoming the most repetitive part of a lyric. It is also where the main theme is more explicit. Along with the corresponding music, it is intended to be the part which listeners tend to remember;
- Bridge: it is an interlude that connects two parts of the song. As verses repeat at least twice, the bridge may then replace the 3rd verse or follow it thus delaying the chorus. In both cases it leads into the chorus;
- Outro: not always present, it is the counterpart of the intro, located at the end of lyrics. It is meant to be a conclusion about the main theme.

For further detail about music structures, please refer to [11] and [12].

IV. Common Conventions for Lyrics Encoding

We are experiencing a tremendous increase in the amount of music-related information made available in digital form. As regards lyrics in particular, with the creation of large collections, we need to infer semantic information from heterogeneously encoded repositories. The overall quality of a repository, as regards both the integrity and the reliability of lyrics, depends on a number of factors. Solving such a problem goes beyond the goals of our initiative, nevertheless it has to be considered. A possible solution could be extracting from the Web multiple versions of the same lyrics, as described in [13].

In any case, we cannot assume that lyrics are encoded following fixed rules. A typical example is the repetition of lines, above all in the chorus: in some cases all the lines are written out, in other cases they are abbreviated through indications such as *(x4)* or *(4 times)*. Another typical example is the recurrence of the chorus, often replaced by indications such as *[Chorus]*, *[Hook]*, or the first line followed by ellipsis.

Fortunately, there are also commonly accepted conventions that can help the segmentation process. Usually, lyrics lines correspond to different text lines and they are separated by control characters such as carriage return and line feed (with slight variations among Windows, Mac and Unix systems). Besides, lyrics are often pre-segmented by leaving blank lines between sections. In this way, verses and refrains are visually recognizable by a human reader and automatically distinguishable by a computer system.

Finally, the use of parentheses inside lyrics is a typical way to specify additional information. Their adoption implies a great number of different meanings:

- vocal sound effects, e.g. *(yeah-yeah-yeah)*;
- background lyrics, e.g. *(anyway the wind blows)*;
- alternate lyrics, e.g. *Karma police, arrest this man (Karma police, arrest this girl)*;[2]
- section identification, e.g. *[Chorus]*;
- singer identification, e.g. *[Women Singing]*;
- performance indications, e.g. *[fade out]* or *[x4]*;
- additional metadata, e.g. the title of the song.

Nevertheless, our approach aims at taking advantage from semantic information originally encoded in lyrics. The indications implying repetitions can be managed through *ad hoc* text expansions, carried out on lines or whole verses. A semantic parsing is required to determine the number of repetitions to perform. The problem of parenthesized contents can be easily solved by ignoring their contents. When parentheses occur within a line, their content is ignored in similarity measures; when they constitute a whole line, the line is deleted from lyrics.

As a final remark, since our algorithm takes formatted text in input, we could impose a set of typographical and semantic conventions to be strictly observed. However, this would imply loss in generality: for instance, most Web archives do not adhere to them.

V. The Algorithm

The segmentation algorithm works in two steps. First, user-defined lyrics are parsed taking into account the original semantic information. The result of the first phase is to label each line through an identifier. Completely identical lines are given the same identifier, ignoring their case, whereas progressive symbols from an ordered sequence (in our examples alphabet letters) are assigned to different lines. However, slight alterations in lines - due to intended variations or typos - should be recognized and managed.

Consequently, a strategic decision concerns the metric which evaluates similarities in text strings. This problem is well known in Information Technology and has been treated in a number os scientific works. For example, an exhaustive discussion of algorithms to compare text strings is provided in [14]. Other approaches have been listed in Section II. Our implementation employs a normalized Levenshtein distance, as described in [15]. In short, the Levenshtein distance between two character sequences is the minimum number of single-character edits (insertion, deletion, substitution) required to change one sequence into the other. The concept of distance is needed to establish if two lines are equal, similar, or completely different. In our implementation the user can set the threshold value as a percentage of tolerated edit distances, since line length can vary significantly.

As a result of the first step, the algorithm produces an array of identifiers - one per each lyrics line - that put in relationship the lines. Such a list has been conceived to keep trace of detected variants. In the examples below the star symbol * is employed, whereas the computer implementation adopts different letter case: capital letters for new lines and lower case for varied occurrences. In this way, comparison between strings - either ignoring or honouring slight differences - can be easily achieved in both cases.

The second step works with lyrics divided into paragraphs. The strategy to recognize block structures is based on weighting a number of descriptors, in order to tag sections with a label. The list of descriptors include:

- The labelling of lines resulting from the previous step;
- Paragraphs in which lyrics are explicitly divided (namely song sections);
- Absolute and relative position of each section in lyrics;
- Number of lines or verses of each section;
- Section similarity.

Another version of the algorithm has been implemented to ignore the original semantic information submitted in input (e.g. line blocks), since in certain cases this can give better results. A trivial answer to this apparent paradox is that the user can input wrong structural information; if the algorithm considers such information reliable it infers wrong results. But a subtler case can happen. As Section III outlined, in a song not all variable sections are verses and all fixed ones are choruses. For instance, also a pre-chorus - if present - is repetitive, as well as the outro can recall the chorus. The version of the algorithm where blank lines are ignored tries to detect sections as long as possible. Lines are evaluated as regards their similarity and then grouped into either variable or repetitive blocks.

Depending on the purpose of the analysis, this behaviour can represent an advantage or a disadvantage. For instance, when the algorithm is used to get the finest classification of lyrics sections, maintaining the original structural information can be relevant; on the contrary, when the goal is retrieving

[2] In an early version of Radiohead's "Karma Police", the first verse was completely different.

```
Twinkle twinkle little star,
How I wonder what you are.

Up above the world so high,
Like a diamond in the sky.

Twinkle twinkle little starsss
HOW I WONDER WHAT YOU       ARE.
```

Fig. 1. Misspelled lyrics of *Twinkle Twinkle Little Star*

correspondences in the audio signal, a simple distinction between repetitive and non-repetitive sections can be sufficient.

VI. Web-based API Methods

The algorithm described in Section V has been implemented as an API, which includes a number of methods publicly available via Web. In this section only the main methods will be declared and commented. The following subsections will define the API request and response in all the available output formats. Responses are returned in one of the following formats:

1) HTML (MIME[3] Content-Type: text/html);
2) JSON[4] (MIME Content-Type: text/plain);
3) plain text (MIME Content-Type: text/plain);
4) XML (MIME Content-Type: text/XML).

Plain text is the simplest way to return results. In case of multiple outputs, comma-separated values notation is adopted. HTML has been included mainly for Web applications. Finally, JSON and XML allow to represent structured information and they are particularly fit for those methods that return multiple values.

The root URL to access API methods is located at `http://www.lim.di.unimi.it/segmenter/`

For ease of use, if nothing more is specified, this address redirects to a Web page where a graphical interface allows the user to input lyrics and launch API methods.

In order to show how API methods work, the lyrics of a classic children's song will be used (see Figure 1). For demonstration purposes, such a text intentionally contains a limited amount of misspellings and typographical errors (e.g. capitalized letters, multiple blank spaces, etc.).

Please note that UTF-8 is the format to encode arguments when calling API methods.

A. *word.count and other counter methods*

This method returns the number of words in the input text, considering white spaces as word delimiters.

Parameters:

- `lyrics_body`: the lyrics to analyse
- `response_type`: the output format. Supported values: `html`, `json`, `text` and `xml`

Launched on the lyrics in Figure 1, this method will return `32` in HTML and text format, `{"word.count":32}` in JSON format, and `<word_count>32</word_count>` in XML format.

Besides *word.count*, other methods to evaluate quantitative characteristics of lyrics are *word.count.per.line*, *line.count*, and *line.count.per.section*.

In accordance with their names, these methods return the total amount of words per line, the number of lines in the input text, and the amount of lines section by section respectively. Carriage return / line feed characters are line delimiters and blank lines are section delimiters. In some cases an additional parameter can be specified, i.e. `blank_lines`, a boolean value to count or ignore blank lines.

For example, the method *line.count.per.section* launched on the lyrics in Figure 1 will return `2,2,2` both in HTML and in text format,

```
[{"section":{"number":1,"line.count":2}},
 {"section":{"number":2,"line.count":2}},
 {"section":{"number":3,"line.count":2}}]]
```

in JSON format, and finally

```
<line_count_per_section>
   <section>
      <number>1</number>
      <line_count>2</line_count>
   </section>
   <section>
      <number>2</number>
      <line_count>2</line_count>
   </section>
   <section>
      <number>3</number>
      <line_count>2</line_count>
   </section>
</line_count_per_section>
```

in XML format.

B. *line.label*

This method returns a sequence of identifiers which represent the identity, similarity or inequality among lines.

Parameters:

- `lyrics_body`: the lyrics to analyse
- `threshold`: a value to set the maximum percentage of differences tolerated in the normalized measure of distance between lines. The percentage always refers to the longest string. Default value: `0.2`, corresponding to 2 edit differences each 10 characters.
- `mark_differences`: a boolean value to highlight slightly different lines or not. Supported values: `true` (default) or `false`
- `response_type`: the output format. Supported values: `html`, `json`, `text` and `xml`

Launched on the lyrics in Figure 1 with the default value for `threshold`, in text format this method will return either `AB CD AB` or `AB CD ab` depending on the settings of

[3] Multipurpose Internet Mail Extensions (MIME) is an Internet standard originally conceived to extends the format of email to support text and header information in character sets other than ASCII, non-text attachments, multi-part messages etc. Now MIME is used to describe content type in general, including for the web.

[4] JavaScript Object Notation (JSON) is a lightweight data-interchange format, easy for humans to read and write as well as for machines to parse and generate. JSON is based on a subset of the JavaScript Programming Language.

mark_differences. The JSON and XML result provides additional information on the edit distance (-1 stands for not available), both in absolute terms and as a percentage referred to the length of the longest string:

```
<line_label>
   <line>
      <text>Twinkle twinkle little star,</text>
      <label>A</label>
      <distance>-1</distance>
      <distance_perc>-1</distance_perc>
   </line>
   <line>
      <text>How I wonder what you are.</text>
      <label>B</label>
      <distance>-1</distance>
      <distance_perc>-1</distance_perc>
   </line>
   ...
   <line>
      <text>Twinkle twinkle little starsss</text>
      <label>a</label>
      <distance>3</distance>
      <distance_perc>0.1</distance_perc>
   </line>
   <line>
      <text>HOW I WONDER WHAT YOU      ARE.</text>
      <label>b</label>
      <distance>5</distance>
      <distance_perc>0.1613</distance_perc>
   </line>
</line_label>
```

in XML format.

Please note that for the last line the Levenshtein distance is 5, since casing is not considered: in other case, it would be $31 - 2 - 5 - 1 = 23$, namely the full length of the second string minus capitalized characters, blank spaces and the final period. Both the last lines are recognized as varied occurrences of the first two, since the threshold is set to 0.2; e.g. with threshold=0.15 the last one would be marked through a new identifier.

C. *song.segment*

This method is probably the most important one, since it returns a sequence of identifiers representing the identity, similarity or inequality among sections. Please note that this method relies on *line.label* in order to mark lines.

Parameters:

- lyrics_body: the lyrics to analyse
- threshold: a value to set the maximum percentage of differences tolerated in the normalized measure of distance between sections. Default value: 0.2.
- mark_differences: a boolean value to highlight slightly different sections or not. Supported values: true (default) or false
- response_type: the output format. Supported values: html, json, text and xml

Some results will be presented and discussed in the next section.

VII. Relevant Examples and Experimental Results

Now we present some clarifying examples, showing how the algorithm deals with the increasing complexity of the structures to decode.

A first example is *California Girls* by The Beach Boys. The structure of this song is a contrasting verse-chorus form that can be represented through the sequence A-A-B-A-A-B-B, where A represents the variable part (namely the verse) and B the repetitive one (namely the chorus). The last column of Table I shows the segmentation of the lyrics reported in the first column. The first step of the algorithm simply recognizes new occurrences of lines already present in the data structure. In the verses all lines differ, whereas the chorus is made by almost identical lines. The fact that the third line in the chorus is recognized as a new line or a repetition of the previous ones clearly depends on the tolerance set for the Levenshtein distance. After the first step, the algorithm parses and compares user-defined blocks of lines. Repetitions or similar blocks are identified by the same letter, whereas variable blocks are marked through different letters.

TABLE I. Lyrics of *California Girls* by The Beach Boys

Well East coast girls are hip	A	$\mathcal{A}$
I really dig those styles they wear	B	
And the Southern girls with the way they talk	C	
They knock me out when I'm down there	D	
The Midwest farmer's daughters	E	$\mathcal{B}$
Really make you feel alright	F	
And the Northern girls with the way they kiss	G	
They keep their boyfriends warm at night	H	
I wish they all could be California	I	$\mathcal{C}$
I wish they all could be California	I	
I wish they all could be California girls	I	
The West coast has the sunshine	J	$\mathcal{D}$
And the girls all get so tanned	K	
I dig a French bikini on Hawaii island dolls	L	
By a palm tree in the sand	M	
I been all around this great big world	N	$\mathcal{E}$
And I seen all kinds of girls	O	
Yeah, but I couldn't wait to get back in the states	P	
Back to the cutest girls in the world	Q	
I wish they all could be California	I	$\mathcal{C}$
I wish they all could be California	I	
I wish they all could be California girls	I	
I wish they all could be California	I	$\mathcal{C}$
I wish they all could be California	I	
I wish they all could be California	I	
I wish they all could be California	I	

A more challenging example is *Alejandro* by Lady Gaga. The structure of this pop song is more complex. Lyrics have been reported in Table II, together with their segmentation. Once again, the columns represent respectively the original lyrics from MusixMatch[5] (including typos, in bold), labelled lines, and the corresponding segmentation. This time, we use also the variant of the algorithm with the removal of

[5] MusixMatch is one of the largest lyrics catalog, containing more than 7 million song lyrics in 32 languages, available through Web API calls.

parentheses and other special characters. The new results are reported in the additional columns.

The first step of the algorithm aims at discovering similarities in lines, also supporting slight variations. In the table, the latter case is graphically rendered through the star (*) symbol. Here the Levenshtein distance for measuring the difference between two character sequences was set to 20%. Consequently, lines containing typos are recognized as occurrences of the same text (see the bold line in Table II), as well as lines that share most characters (see K and K*).

The second step organizes line occurrences in blocks by using the original semantic information about block structures and analysing the composition of each block. The final result is shown in the rightmost column of Table II. In order to evaluate the performances of the algorithm, we provide a short analysis of the song, based on its audio. The first lines (identified by $\mathcal{A}$) represent a sort of intro and in the audio track they are spoken. After the first section, marked as $\mathcal{B}$, there is a pre-chorus $\mathcal{C}$, followed by the chorus $\mathcal{D}$ and the post-chorus $\mathcal{E}$. In accordance with many statistical analyses on pop/rock repertoire, such a layout is literally repeated. Then, the bridge occurs and the chorus is exposed again.

It is worth noting that the last section is better analysed by the algorithm when the contents of parentheses are ignored. In fact, in this case the last block is automatically recognized as a new occurrence of the chorus plus a final line. As mentioned in Section IV, often parentheses are used to indicate non-relevant parts of the text (repetitions, performance indications, etc.) and ignoring them can improve the quality of the segmentation.

A further step, based on the analysis of the number of lines constituting each section, could provide information on meta-sections: $2+4$ (verse, variable), $4+5+4$ (chorus, repeated), $2+4$ (verse, variable), $4+5+4$ (chorus, repeated), $4+5+5$ (bridge/chorus, partially repeated), n (outro, repeated).

VIII. Conclusion and Future Works

The algorithm proposed here virtually addresses vocal music from any historical period, genre, geographical area, etc. Nevertheless, this early implementation mainly focuses pop/rock music and its application to other forms should be better investigated. For instance, some tests conducted through our framework have shown that rap songs have structures difficult to be recognized.

This paper represents only the first step towards an automatic semantics-based segmentation of lyrics. The described algorithm is very easy to implement and it has revealed to be efficient and effective on relatively short texts as song lyrics. Needless to say, hyphenation and phonemes - aspects clearly depending on the language - could add further analysis tools. Other improvements could be achieved by performing a pre-clustering and classification of lyrics, in order to use *ad hoc* analyses depending on text features.

As future work, the idea is investigating different algorithms to obtain lyrics segmentation and using supervised neural networks with two goals in mind: 1) setting the numeric parameters of the proposed algorithm to improve performances on a given repertoire, and 2) weighting the results of different

TABLE II. LYRICS OF *Alejandro* BY LADY GAGA

I know that we are young and I know you may love me	A	$\mathcal{A}$	A	$\mathcal{A}$
But I just can't be with you like this anymore, Alejandro	B		B	
She's got both hands in her pockets	C	$\mathcal{B}$	C	$\mathcal{B}$
And she won't look at you, won't look at you	D		D	
She hides true love, en su bolsillo	E		E	
She's got a halo around her finger around you	F		F	
You know that I love you, boy	G	$\mathcal{C}$	G	$\mathcal{C}$
Hot like Mexico, rejoice	H		H	
At this point I gotta choose	I		I	
Nothing to lose	J		J	
Don't call my name, don't call my name, Alejandro	K	$\mathcal{D}$	K	$\mathcal{D}$
I'm not your babe, I'm not your babe, Fernando	L		L	
Don't wanna kiss, don't wanna touch	M		M	
Just smoke one cigarette and hush	N		N	
Don't call my name, don't call my name, Roberto	K*		K*	
Alejandro, Alejandro	O	$\mathcal{E}$	O	$\mathcal{E}$
Ale-Alejandro, Ale-Alejandro	P		P	
Alejandro, Alejandro	O		O	
Ale-Alejandro, Ale-Alejandro	P		P	
Stop please, just let me go	Q	$\mathcal{F}$	Q	$\mathcal{F}$
Alejandro, just let me go	R		R	
She's not broken, she's just a baby	S	$\mathcal{G}$	S	$\mathcal{G}$
But her boyfriend's like a dad, just like a dad	T		T	
And all those flames that burned before him	U		U	
Now he's gonna firefight, got cool the bad	V		V	
You know that I love you, boy	G	$\mathcal{C}$	G	$\mathcal{C}$
Hot like Mexico, rejoice	H		H	
At this point I gotta choose	I		I	
Nothing to lose	J		J	
Don't call my name, don't call my name, Alejandro	K	$\mathcal{D}$	K	$\mathcal{D}$
I'm not your babe, I'm not your babe, Fernando	L		L	
Don't wanna kiss, don't wanna touch	M		M	
Just smoke one cigarette and hush	N		N	
Don't call my name, don't call my name, Roberto	K*		K*	
Alejandro, Alejandro	O	$\mathcal{E}$	O	$\mathcal{E}$
Ale-Alejandro, Ale-Alejandro	P		P	
Alejandro, Alejandro	O		O	
Ale-Alejandro, Ale-Alejandro	P		P	
Don't bother me, don't bother me, Alejandro	W	$\mathcal{H}$	W	$\mathcal{H}$
Don't call my name, don't call my name, bye Fernando	K*		K*	
I'm not your babe, I'm not your babe, Alejandro	L*		L*	
Don't wanna kiss, don't wanna touch, Fernando	X		X	
Don't call my name, don't call my name, Alejandro	K	$\mathcal{D}$	K	$\mathcal{D}$
I'm not your babe, I'm not your babe, Fernando	L		L	
Don't wanna kiss, don't wanna touch	M		M	
Just smoke one cigarette and hush	N		N	
Don't call my name, don't call my name, Roberto	K*		K*	
Alejandro, Alejandro	O	$\mathcal{E}$	O	$\mathcal{E}$
Ale-Alejandro, Ale-Alejandro	P		P	
Alejandro, Alejandro	O		O	
Ale-Alejandro, Ale-Alejandro	P		P	
Don't call my name, don't call my name, Alejandro	K	$\mathcal{I}$	K	$\mathcal{D}$
(Alejandro, Alejandro)	O*		-	
nI'm not your babe, I'm not your babe, Fernado	L*		L*	
(Ale-Alejandro, Ale-Alejandro)	P*		-	
Don't wanna kiss, don't wanna touch	M		M	
(Alejandro, Alejandro)	O*		-	
Just smoke one cigarette and hush	N		N	
Don't call my name, don't call my name, Roberto	K*		K*	
(Ale-Alejandro, Ale-Alejandro)	P*		-	
Alejandro	Y		Y	

segmentation algorithms still not implemented in our framework.

References

[1] K. Lee and M. Cremer, "Segmentation-based lyrics-audio alignment using dynamic programming," in *Proceedings of the 9th International Conference on Music Information Retrieval (ISMIR)*, 2008, pp. 395–400.

[2] J. P. Mahedero, Á. Martínez, P. Cano, M. Koppenberger, and F. Gouyon, "Natural language processing of lyrics," in *Proceedings of the 13th Annual ACM International Conference on Multimedia*. ACM, 2005, pp. 475–478.

[3] H.-T. Cheng, Y.-H. Yang, Y.-C. Lin, and H. H. Chen, "Multimodal structure segmentation and analysis of music using audio and textual information," in *IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2009, pp. 1677–1680.

[4] N. C. Maddage, C. Xu, M. S. Kankanhalli, and X. Shao, "Content-based music structure analysis with applications to music semantics understanding," in *Proceedings of the 12th Annual ACM International Conference on Multimedia*. ACM, 2004, pp. 112–119.

[5] N. C. Maddage, "Automatic structure detection for popular music," *Multimedia*, vol. 13, no. 1, pp. 65–77, 2006.

[6] J. Foote, "Visualizing music and audio using self-similarity," in *Proceedings of the 7th ACM International Conference on Multimedia (Part 1)*. ACM, 1999, pp. 77–80.

[7] M. Cooper and J. Foote, "Summarizing popular music via structural similarity analysis," in *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*. IEEE, 2003, pp. 127–130.

[8] E. Peiszer, T. Lidy, and A. Rauber, "Automatic audio segmentation: Segment boundary and structure detection in popular music," *Proceedings of the 2nd International Workshop on Learning Semantics of Audio Signals (LSAS)*, pp. 45–59, 2008.

[9] F. Kaiser and T. Sikora, "Music structure discovery in popular music using non-negative matrix factorization," in *Proceedings of the 11th International Society for Music Information Retrieval Conference (ISMIR)*, 2010, pp. 429–434.

[10] N. Itou and K. Nishimoto, "A voice-to-midi system for singing melodies with lyrics," in *Proceedings of the International Conference on Advances in Computer Entertainment Technology*. ACM, 2007, pp. 183–189.

[11] P. Tagg, "Analysing popular music: Theory, method and practice," *Popular Music*, vol. 2, no. 1, pp. 37–67, 1982.

[12] D. Brackett, *Interpreting popular music*. University of California Press, 1995.

[13] G. Geleijnse and J. H. Korst, "Efficient lyrics extraction from the web," in *Proceedings of the 7th International Society for Music Information Retrieval Conference (ISMIR)*, 2006, pp. 371–372.

[14] M. Crochemore, C. Hancart, and T. Lecroq, *Algorithms on Strings*. Cambridge University Press, 2007.

[15] L. Yujian and L. Bo, "A normalized levenshtein distance metric," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 6, pp. 1091–1095, 2007.