Project Report: Implementing Lyrics Search using ElasticSearch

Rong Chang

1 Introduction

Fuzzy or approximate search has been widely adopted by search engines. It can tolerate typing/spelling errors and still provide relevant results even if there is no exact match. However, for some search applications, the user may not know that exact words/phrases they want to search, but only know the pronunciation. The current implementation of approximate search in mainstream search engines is not able to match the results that users need.

A typical example could be the search for the lyrics by how a phrase or a short sentence in the song sounds to us. In this project, I built a lyrics search engine using phonetic tokenization. In addition to using word-level tokenization, phonetic tokenization further convert the words into phonetic representation. Likewise, the query words are broken in to phonetic tokens before querying the index. Such tokenization allows us to search lyrics even if we don't know exactly the words, but know the sound of it (which happens often we hear a song in the radio or coffee shop).

2 Problem

I proposed an idea to allow people to search for items by phonetics. In this project, the implementation would focus on the lyrics search. Nowadays, we may hear songs while we are eating in the restaurant, shopping in the grocery store, or even ordering a beverage in the coffee shop. It is very common that we enjoy a song and would like to find out more details of this song. There are mobile apps that allow people to record audios of the songs or even hum to search (SoundHound). However, it is not always convenient for people to record the song anytime or remember how to hum the song accurate enough to be recognized. In comparison, with a phonetic search engine, the users may type a few words of phrases they believe they had heard (it doesn't need to be the correct words) and get the desired results.

There are a few websites providing search application for songs by lyrics, such as lyrics.com, findmusicbylyrics.com, and azlyrics.com. These lyrics search applications allow users to search for a song by words and phrase pieces they may remember from a song or any other information such as title, singer(s), and album. For example, if I heard a couple of words "slam the door" and typed to search, I could receive the search result with a song "Frozen" I wanted.

However, word or phrase search for lyrics is complicated. We need to build every word from how it sounds to us. Syntax or semantics cannot provide us the evidence in the lyrics. To make it musical, **a lot of single words in a song could be broken into pieces and may sound as a combination of two other words.** It is also possible that **the ending sounds can be covered up.**

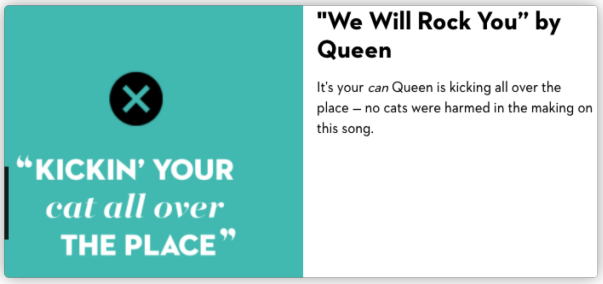


Figure 1: Example on how a word messed in the song of We Will Rock You

For example, there is a famous song *We Will Rock You*. Many people messing up the word "can" to "cat" in one sentence "Kicking your can all over the place", as shown in Figure 1 and listed in the web post. As seen in the search result on Lyrics.com with query as "kicking your cat all over the place" (Figure 2) with just one word "cat" messed, we could see the listed songs were even not close to the sentence either "kick" or "over the place".

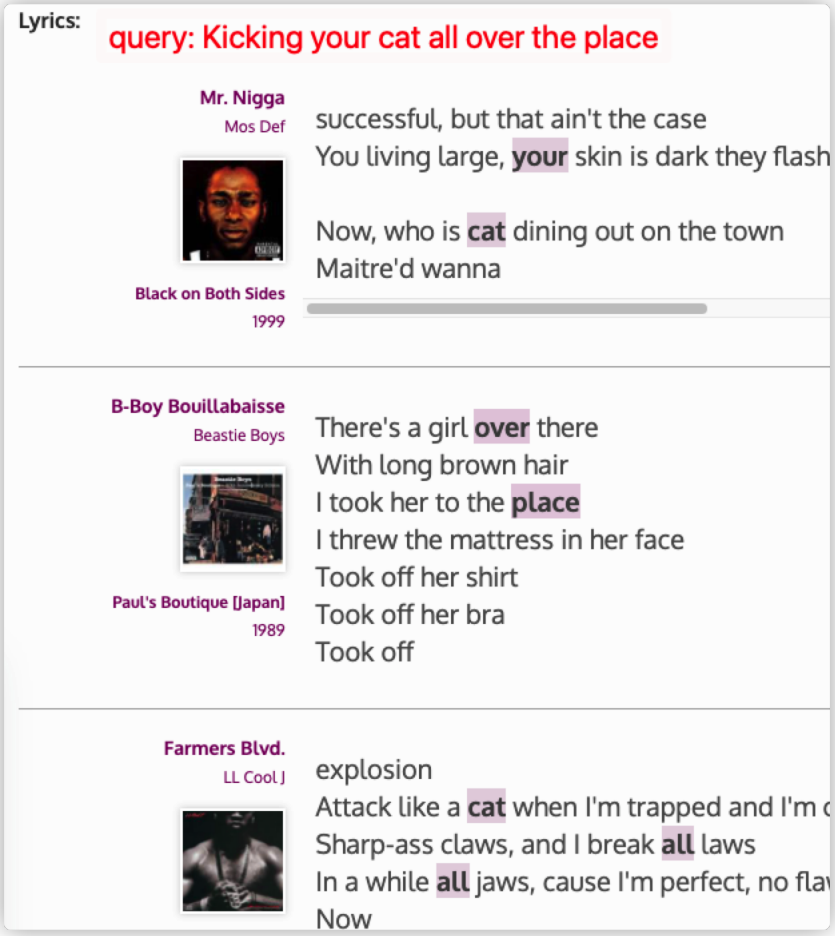
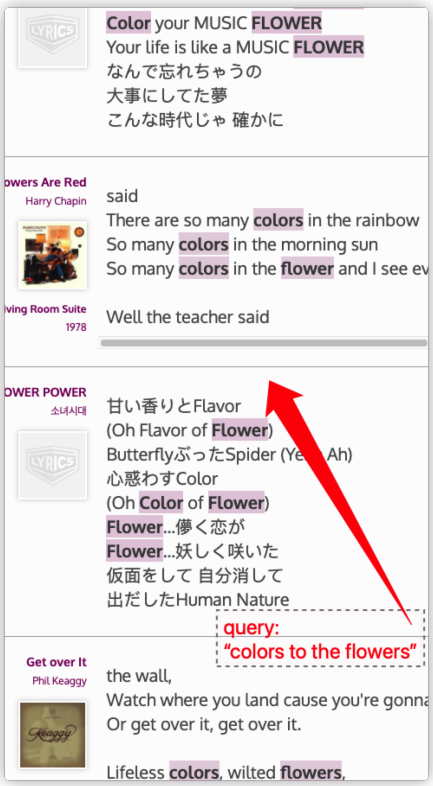
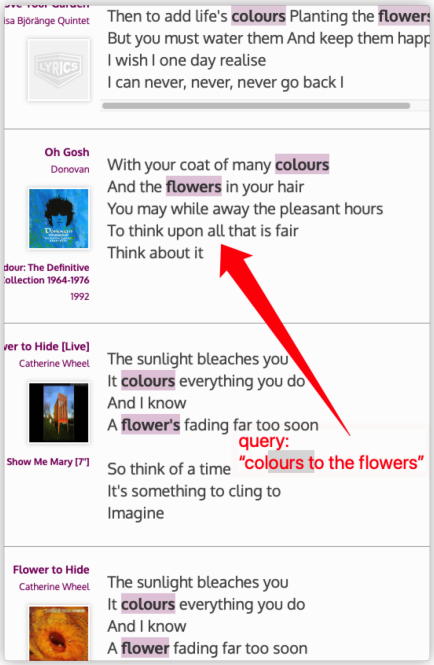


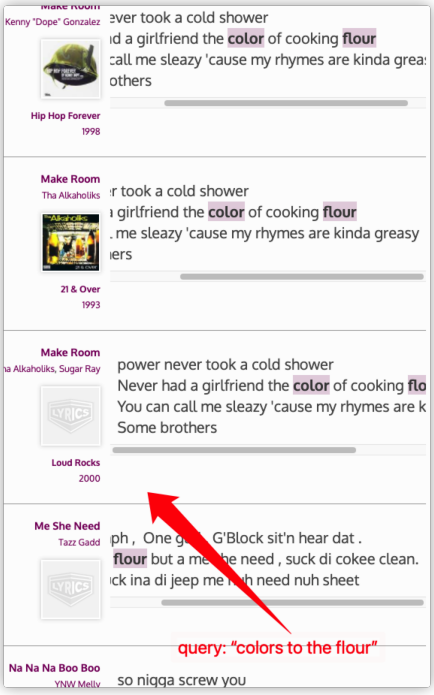
Figure 2: Example on how a word messed in the song of We Will Rock You



(a) Search "colors to the flower"



(b) Search "colours to the flower"



(c) Search "colors to the flour"

Figure 3: Search results for "Colours"

Once I heard a song *Colours* with some words like "color" and a phrase sound like "to the [flaw·ur]". The accurate sentence is "**...colours ... to the flour**". With the feeling of "**color**", I typed "to the **flower**" and never thought about "flour". I tried "**colors to the flowers**" yet could not locate any I want, as displayed in Figure 3. The rhyming words are not easy to identify in the lyrics search.

For the test purpose, I tried with the accurate spelling of "colours", such that a sentence was typed as "**colours to the flower**", I still could not get to the song (Figure 3). In addition, the word "colours" could also be typed in "colour", "color", or "colors". It is very normal that the rhythm may result in the hidden of "s". As shown in Figure 3, even with the query "**colors to the flour**", "colors", as the variation of "colours" was not able to be identified. Thus, other than the phonetic issue, the features provided by current existing lyrics search is very limited.

In summary, the issues existing in current lyrics search are:

1. not able to detect rhyming words, such as "flower" and "flour";
2. not able to detect a word that sounds like a combination of a couple of other words;
3. zero tolerance to words with similar pronunciation, such as "call back" and "come back", "walk" and "work", the ones are not easily identified in a song;
4. not able to stay with the word order typed in the query;
5. not able to find lyrics if there are a couple of missing word(s);
6. no allowance on different word forms (stemming).

3 Design and Implementation

The backend of lyrics search engine was implemented through ElasticSearch (www.elastic.co). The application was built using Flask with html and css support for basic user interface.

3.1 Dataset

With the hope to test if this current lyrics search engine is more tolerant to the phonetic messy words in the search practice, I borrowed a lyrics dataset on Kaggle.

This music lyrics dataset is called "150K Lyrics Labeled with Spotify Valence" that accessible through the link at https://www.kaggle.com/edenbd/150k-lyrics-labeled-with-spotify-valence. It originally contains 158,352 song lyrics as well as the basic information about song title and artist(s). For the evaluation purpose, I manually added more, which result in a total of 158,354 songs.

3.2 Phonetic tokens

The majority of this project was the implementation of phonetic search with the hope to solve the first three issues. ElasticSearch does not provide phonetic search out of the box. But its **phonetic analysis plugin** [1] provides the phonetic analysis that could support the phonetic token filter process.

The phonetic analysis plugin provides token filters which convert tokens to their phonetic representation using Soundex, Metaphone, and a variety of other phonetic algorithms. I decided to choose the most commonly used Metaphone algorithm [2].

The Metaphone algorithm is proposed by Lawrence Philips in 1990 to index English words by their pronunciation. The algorithm encodes English words into some letters (and number 0) to represent the sounds. Similar sounded words should share the same encoding. For example, the word "feature" and "future" both have the same encoding of "FTR".

There are also some limitation with the phonetic algorithms. Some rhyming words are not able to detect while some words are mixed up due to the rhyming vowels. For examples, the phrases of "come back" and "call back" are easily messed because in the music, the ending sounds might be covered up. However, since words "come" and "call" are not rhyming on its majority, these two words do not have the same encoding. In the phonetic algorithms, "come" is coded as "KM" and "call" is coded as "KL". This limitation is because these algorithms **convert each token by how it spells not by how it sounds to us**.

Thus, in addition just using the phonetic tokens, standard word tokens are also used (see the code "replace": False) so that word matching can still be used. The analysis settings for the indexing is listed below.

```
phonetic_setting = {
    "settings": {
        "analysis": {
            "analyzer": {
                "default": {"tokenizer": "standard",
                            "filter": ["lowercase", "my_metaphone"]}},
            "filter": {
                "my_metaphone": {"type": "phonetic",
                                 "encoder": "metaphone",
                                 "replace": False} }
        }
    },
    "mappings": {"properties": {"lyrics": {"type": "text",
                                           "index_prefixes": { }}}
    }
}
```

3.3 Phrase and fuzzy search

In order to detect rhyming words, phonetic analyzer can sometimes make mistakes. For example, though we pronounce "flower" and "flour" in exactly same way, but their vowel combinations are different. The word "flower" contains two vowels "ow" and "er", yet "flour" only contains one vowel "our". Thus, in the phonetic algorithm, these two words are not converted into same phonetic representation. The word "flower" is converted as ['FLWR'] whereas "flour" is converted as ['FLR']. Thus, some tolerance to the fuzzy query was needed. Too much fuzziness would result in too many noises for the accurate result matched.

After introducing fuzzy search, it would be very difficult to retrieve a desired document by using word-level queries. For example, a query of "colours to the flower" would be matched to any lyrics with the word "flower", "flour", "floor", "color", "cooler", and etc., and it would be impossible to get the correct lyric back ("colours to the flour").

However, if we implement phrase matching, meaning that the lyrics should contain the exact sequence "colours to the flower", in the word form or phonetic form, we could easily retrieve the desired document in the top result list. The default phrase search in ElasticSearch does not support fuzzy search, but fortunately we can implement phrase search with the span-near query from ElasticSearch and use fuzzy search at the same time.

Span-near query in ElasticSearch matches to a list of tokens that located close to each other. It has options to configure how "close" those tokens should be and whether those tokens should be in the exact same order as listed in the token list. When configured correctly, the span-near query can act like a phrase query but allow more customization for the query of each term (adding fuzziness or other logics). The option to allow some other words inserted between search terms is a perfect fit for lyrics search. For example, "He's gonna walk like you" may easily thought to be "his work like you" due to the covered up of "gonna" and similar pronunciation of "walk" and "work". This option (called "slop") can be used so that "his work like you" can match with "He's gonna walk like you".

With both the phrase search and fuzzy search, we are able to have some nice tolerance of both phonetic and spelling incorrectness in the search queries. With phonetic analyzer, "He's gonna walk like you" is represented as ['HS', 'KN', 'WLK', 'LK', 'Y'], while "his work like you" is converted to ['HS', 'WRK', 'LK', 'Y']. The strict phrase search would avoid all the other noises and focus on the original order. However, with the order ['HS', 'LK', 'Y'] is not able to detect this song. As such, fuzziness provides some tolerance between 'WLK' and 'WRK'. In addition, phrase search with one word span between every two ordered phonetic tokens would make search possible.

Below shows how the phonetic search is implemented and combined with fuzzy phrase queries. I first manually tokenize the query text using standard and phonetic analyzer, then saved them into the ordered list.

```
analyze_default = {
    "tokenizer": "standard", "filter": [ "lowercase"], "text": sentence }
```

```
words = es.indices.analyze(index=indexname, body=analyze_default)
words = [i["token"] for i in words["tokens"]]

analyze_phonetics = {
    "tokenizer": "standard",
    "filter": [ "lowercase", {"type": "phonetic",
                              "encoder": "metaphone",
                              "replace": True} ],
    "text": sentence }
phonetic_words = es.indices.analyze(index=indexname, body=analyze_phonetics)
phonetic_words = [i["token"] for i in phonetic_words["tokens"]]
```

Then, I used span-near to achieve phrase search. The in-order options specify that the order of the words need to the same as the order in the list of clauses. The slop options specify that one word distance may exist between every two ordered words or phonetic words in the query.

```
"query":{ "bool":{ "must":[ { "span_near":{ "clauses": spans,
                                            "slop": 1,
                                            "in_order":True}  }] } }
```

For each word in the list of both standard tokens and phonetic tokens, I created a span-or query to be nested under the span-near query. The span-or query would allow matching either the standard token or the phonetic token of the same query word to a lyric document. And for matching both the standard tokens of phonetic tokens, I used span-multi to add fuzziness. This gives the flexibility of each token to be matched to an exact word or a phonetic token, and both with fuzziness. For example, "colour to the flower" could be matched to "colour to the flour" because the words "colour", "to", "the" are matched with their standard tokens and "flower" is matched to "flour" with their phonetic tokens ("FLWR" and "FLR" has 1 edit distance).

```
spans = []
for i in range(len(words)):
span =  {"span_or": {
            "clauses": [
                {"span_multi": {"match": {"fuzzy": {"lyrics": {"value": words[i],
    "fuzziness":"AUTO"}}}}},
                {"span_multi":{"match":{"fuzzy":{"lyrics":{"value": phonetic_words
    [i],"fuzziness":"AUTO"}}}}}
                ]
            }
        }
spans.append(span)
```

3.4 Other ES API implementation

In Elasticsearch, stemming is handled by stemmer token filters. This token filter was not included in this project because different forms of the words may be covered by either phonetic analyzer or fuzzy query.

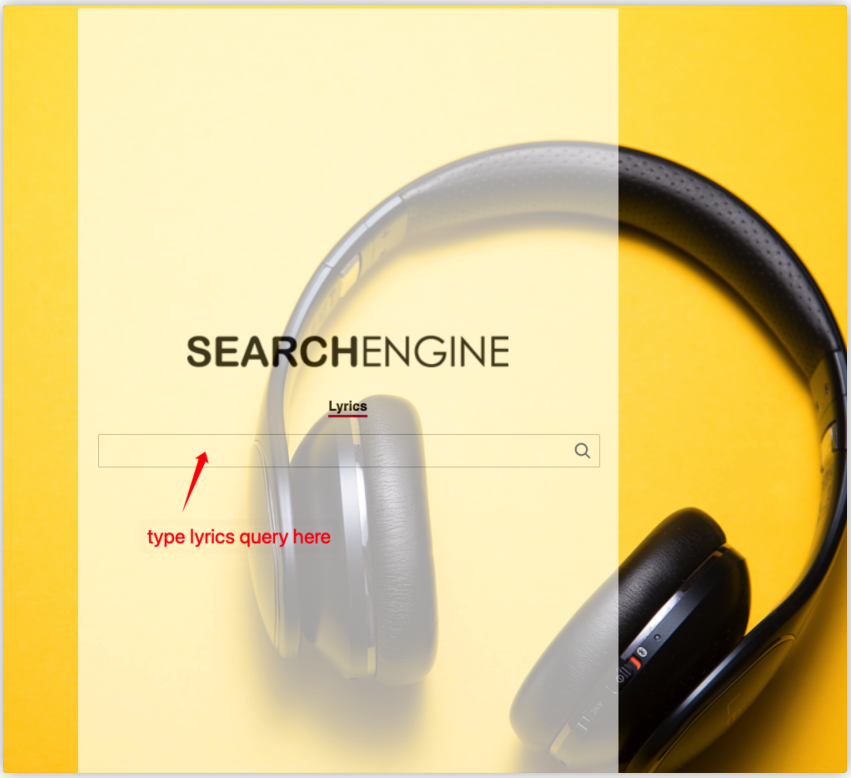


Figure 4: Example on how a word messed in the song of We Will Rock You

As the lyrics collection contains over 150K songs, the indexing process originally took over 7 hours on my first trial. I managed to increase the indexing speed by using BULK API and use mutli-threading and it reduces the total indexing time to within 30 seconds on my laptop.

```
deque( parallel_bulk( client=es,
                      actions=gendata(),
                      thread_count=16),
       maxlen=0 )
```

I also added highlight feature and the matched texts are highlighted in the text. For highlighting, the following field is added to the query payload in ElasticSearch. The returned query result would contain a highlight field that would embed $< em >$ tag to the matched text in the lyrics. Then we changed the style for the $< em >$ tag in the HTML template so that the highlighted lyrics can be displayed with good visuals.

```
"highlight": {"fields": {"lyrics": {}}}
```

3.5 User interface

The application is built using Flask. The interface is built from the template created by vanGato (accessible through link: https://github.com/phpSoftware/search-engine-template). The interface

in this project is simple which contains only two pages - an index page for query entry (Figure 4) and a display page for results (refer to figures in test cases). The display page list the top 50 matched results with the highlight of the matched words.

4 Evaluation and Conclusion

How people perceive the words while they first hear a song is complicated. Due to the inaccessible experiment data, the efficiency of current project is not able to evaluated statistically. Thus, the evaluation is limited on the test cases from my personal experience and knowledge background. The test cases were created focus on the evaluation of the feasibility and accomplishment achieved by this phonetic search engine.

- Test case 1: *We Will Rock You* with query "Kicking your cat all over the place", compares with Figure 1.

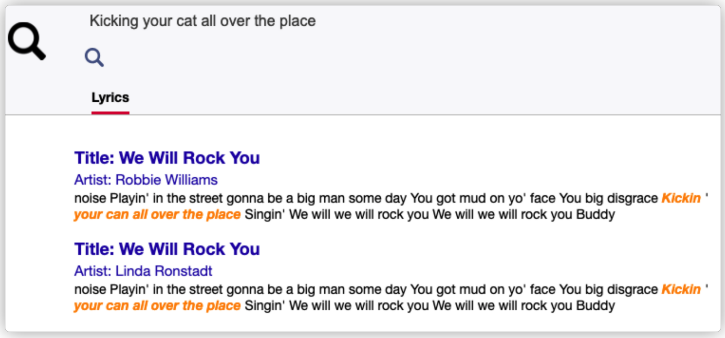


Figure 5: Search result for test case 1 on this app

- Test case 2: *Colours* with the query "colors to the flower", compares with Figure 3.

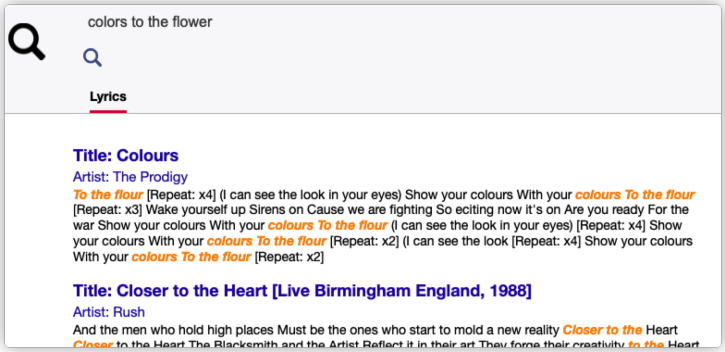
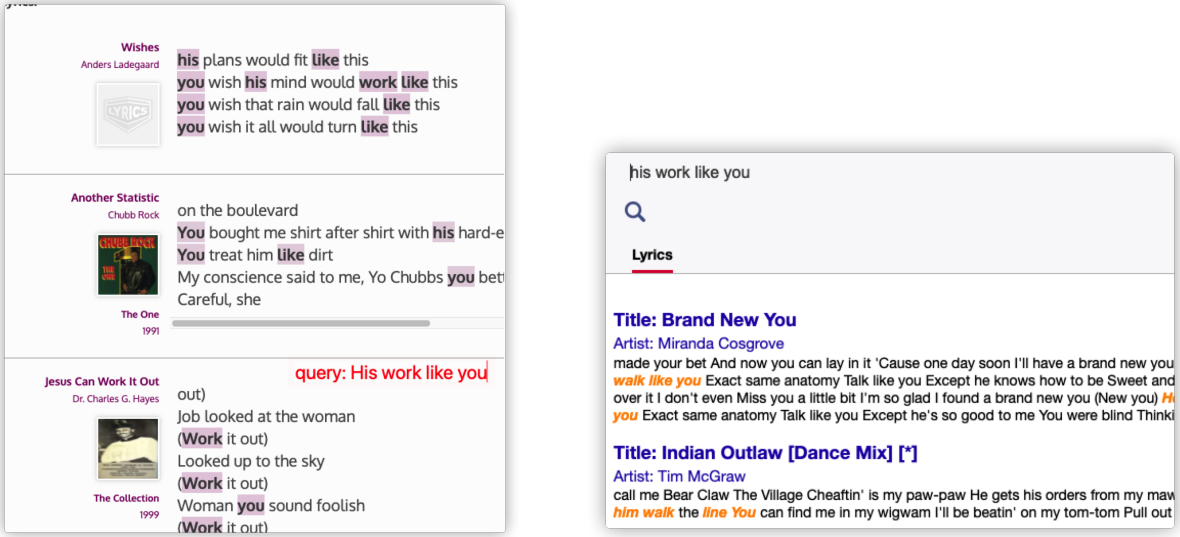


Figure 6: Search result for test case 2 on this app

- Test case 3: *Brand New You*. A sentence of "He's gonna walk like you" is messed with "His work like you".



(a) Search result on lyrics.com

(b) Search result on this app

Figure 7: Search result comparison for test case 3

As seen in the above test cases, this project has overcome the common problems in current lyrics search engines listed in section 2 and obtained the features of phonetic search. With additional features of fuzzy and phrase research, this lyrics search engine produced better query matching results. Such phonetic technique may also be implemented future to see if it works efficiently for other similar search fields which are not based on the grammar or context, such as movie search and kid phonetic vocabulary search.

References

[1] Phonetic analysis plugin: Elasticsearch plugins and integrations. Accessible at `https://www.elastic.co/guide/en/elasticsearch/plugins/current/analysis-phonetic.html`.

[2] Lawrence Philips. Hanging on the metaphone. *Computer Language Magazine*, 7(12):39–44, December 1990. Accessible at `http://www.cuj.com/documents/s=8038/cuj0006philips/`.