International Transaction Journal of Engineering, Management, & Applied Sciences & Technologies

http://TuEngr.com

# A NOVEL AND EFFICIENT METHOD FOR ROMAN TO URDU TRANSLITERATION VIA HEURISTICS-BASED SEARCHING ON PARSE TREES

**Sayam Qazi [a], Humera Tariq [a*]**

[a] *Department of Computer Science, University of Karachi, Karachi, PAKISTAN*

ARTICLEINFO

*Article history:*
Received 21 January 2019
Received in revised form 29 March 2019
Accepted 01 April 2019
Available online
03 April 2019

*Keywords:*
Emission Frequencies; Terminal Frequencies; Transitional Frequencies; Zipf Law; Hash Map; Heat Map; Roman Urdu; Path Unfolding; Depth First Search.

ABSTRACT

Roman text still forms the significant part of Urdu Data on Internet and there exist ample room for improvement particularly in this domain of Natural Language Processing (NLP). Existing Systems for Roman to Urdu Transliteration possesses their own strengths but work still need to be done to improve their performance. The objective of this particular research is to build a reliable Roman to Urdu Transliteration Batch Processing System with least number of manual corrections required at the user end, thus enhancing the efficiency and reliability of existing and proposed transliteration systems. Parse Tree, Transliteration Tree and novel Heuristic function have been proposed by observing key characteristic of Roman Urdu language. The work has been concluded by giving a benchmark of the proposed solution in terms of computational complexity, performance, and accuracy. Correct transliteration with high score has been found up to 78%, with a low score they found to be 21% while the wrong transliteration would be only 0.53% for all tested word. Some limitations of the algorithms which are: (1) Sometimes it gets the translation correct but ranks are too low to be within the tolerance. This can be mitigated by using a better heuristic function. (2) Sometimes it generates too many correct translations which are in principle correct but invalid when considering the context.



## 1. INTRODUCTION

Users from different geographical background possess a strong desire to use their own native language instead of English for communication purpose in an email, chat and other existing text-based solutions. A quick and natural solution is Transliteration i.e. using combinations of English alphabets that made a similar sound to their native language's corresponding alphabet. This NLP mapping problem is well known to many computer scientists and researchers have attempted to address the problem with varying success rate. The most notable is by author (Tafseer, 2009). In his paper, he proposed a Rule-based System with default substitutions as a fallback strategy which works

*Corresponding author (H. Tariq). Tel: +92-21-3012387949. E-mail: humera@uok.edu.pk  Volume 10 No.3 ISSN2228-9860 eISSN1906-9642 http://TUENGR.COM/V10/567.pdf DOI: 10.14456/ITJEMAST.2019.53

quite well in some scenarios but fails on several edge cases that he discussed in the later section of the paper. Other significant work in this direction has been addressed as the Soundex algorithm and such other Phonetic Search Algorithm (Knuth, 1975). (Rusell, 1918) (Naseem, 2004) (Faruqi, et al, 2011). Neural Network based models are gaining popularity in every aspect of artificial intelligence and researchers have also attempted to orient themselves in this direction as demonstrated by encoder-decoder neural approach for Roman-Urdu to Urdu Transliteration. The model used by Google's GBoard still seems to a black box to many researchers (Alam and Hussain, 2011). Authors in (Hellsten et. al., 2017) explained that Google's GBoard model is working on the basis of finite-state transducers. Bilal et al. (2016) apply the various machine learning models for sentiment analysis of Roman Urdu blogs. The goal in this paper is to provide a solution in the form of a plug-in library which allows end users to convert a large amount of roman text into popular Urdu text with least number of manual error corrections. The solution thus provides convenience to developers to enable this functionality from the get-go in their application without worrying for the inner working details of the algorithm. We make some assumptions about the scenarios in which this system will be used. (1) The text to be transliterated belongs to the target language (Urdu) only. This means that words that are from language but spoken in Urdu anyway are not considered to be input to the algorithm. Any occasion of such inputs giving correct output should be considered a mere coincidence. (2) We also make some tradeoffs in speed and accuracy to favor the accuracy of the proposed algorithm while sacrificing some speed. Target speed is still kept reasonable for practical use cases. Rest of the paper is organized as follows. **Section 2** outlines the required data to perform this study and describes the techniques used to collect and clean the data and challenges that arose during the process. **Section 3** describes various linguistic analytical techniques to spot meaningful patterns in the Roman and Urdu language that could be leveraged in the design of the algorithm. **Section 4** illustrates the proposed algorithm that was developed with the help of information collected in Section 2 and also provide some examples to further clarify the workings of its internals. Finally, **Section 5** contains the results and findings from running the algorithm on the Roman data. The paper concludes with a thorough benchmark of the performance, accuracy, and limitations of the algorithm. Moreover, it presents some paths to be taken to further improve the algorithm.

## 2. DATA AND PREPROCESSING REQUIREMENTS

Data is one of the most crucial requirement while designing a reliable algorithm. In our case, the available datasets are either very small or they are incomplete. The incomplete and small nature of data led us to formulate a strategy for acquiring Roman and Urdu data. Since the web has a fairly large number of websites that use the Roman script as their primary medium of written information. We decided to write a scraper that can scrape roman text from the web. Related code snippet can be found in the Appendix in the end. A large amount of Urdu data was also needed to train and validate the model. Se we collected that data from well-known sources: https://www.wikipedia.ur, https://www.hamariweb.pk, https://www.urdupoint.pk, and https://www.cle.org. Fortunately, not much effort was required to clean up the data from these sources. In fact, the last source had the cleanest data of them all ready to be used for training and testing. The next step was to remove the noise from acquired data i.e. remove non-roman data that made it through into the compiled corpus.

## 3. STATISTICAL ANALYSIS OF ROMAN AND URDU DATA

The main statistical properties of linguistic data are Emission, Terminal, and Transitional Frequencies. Other important properties are Transitional and Word Length Distribution.

### 3.1 ANALYSIS OF ROMAN DATA

#### 3.1.1 EMISSION FREQUENCIES

Knowledge of the probabilities, of which letters occurring at the start of a word is important and is referred to as emission frequency. Emission frequencies of roman have been shown in Figure 1which shows that there are several letters that are highly likely to occur at the start of the word along with some letters that rarely ever occur at the start of the word.

#### 3.1.2 TERMINAL FREQUENCIES

The frequency of a letter occurring at the end of a word in a language is said to be terminal frequency. This characteristic helps a prediction algorithm to forecast how likely it is that the word may have ended on a particular alphabet. The graph in Figure 2. shows the terminal frequencies of the Roman Urdu corpus. The Graph clearly shows that there are several letters that are highly likely to occur at the end of the word along with some letters that rarely ever occur at the end of the word for e.g. likely occurring set includes: { a, e, i, o, y } this makes sense because many verbs in Urdu end with vowel like “utho”, “khao” and “khelo” when used in sentences. There seem some exceptions which will occur rarely for e.g. { c, f, g, p, q }.

#### 3.1.3 TRANSITIONAL FREQUENCIES

Every language has the property that some letters are more likely to occur after some letters while less likely to occur after other letters. In order to spot this structure in the Roman language, we ran our algorithm through a python script and acquired a 26 x 26 transition matrix. This matrix is converted into a heatmap image shown in Figure 3 for better readability. In heat map image yellow color means the most frequent, green means moderately frequent and purple means less frequent occurrence of the transition. It is quite clear from the picture that the most dominant bigrams are ha, ar, ia, ka and ay.

#### 3.1.4 TRANSITIONAL DISTRIBUTION

Now we know that some bigrams are more likely and some are don’t but we still want to know how to balance this division i.e. the likelihood of occurrence is distributed linearly or not. So we sorted the frequencies as shown in Figure 4. It can be observed from the curve that transition frequencies show an exponential distribution and the more likely combinations dominate heavily on the less likely combinations. This is, in fact, a known phenomenon dubbed as **Zipf law**. This property of a language is of significant importance because this bias affects the occurrence of bigrams in our proposed solution and search space will dramatically decrease.

#### 3.1.5 WORD LENGTH DISTRIBUTION

Designing a Transliteration Algorithm also looks into the possibility that: how long will be the

*Corresponding author (H. Tariq). Tel: +92-21-3012387949. E-mail: humera@uok.edu.pk  Volume 10 No.3 ISSN2228-9860 eISSN1906-9642 http://TUENGR.COM/V10/567.pdf DOI: 10.14456/ITJEMAST.2019.53

encountered word. In order to find that, a script ran through the corpus and counted the words with lengths from 1 to 16 and plotted the frequencies as shown in Figure 5. One can observe that this is a left-skewed Gaussian distribution (Moisl, 2009). It means that there are rarely ever used words which are more than 12 characters long. This will be an important aspect to consider when developing the actual algorithm.

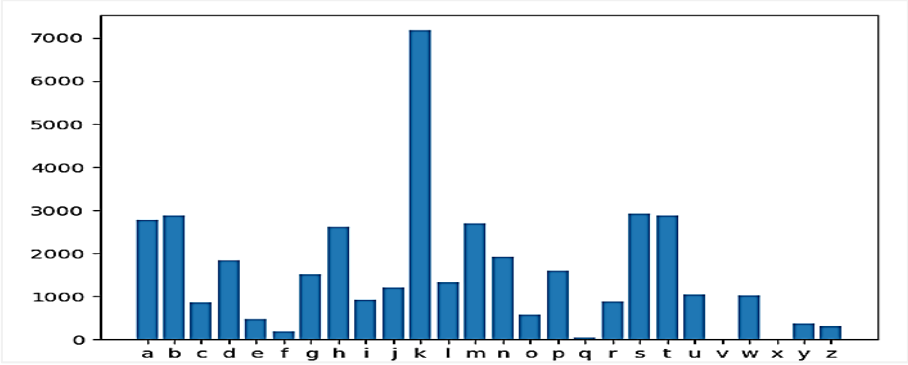


**Figure 1**: Emission Frequencies of Words in Corpus

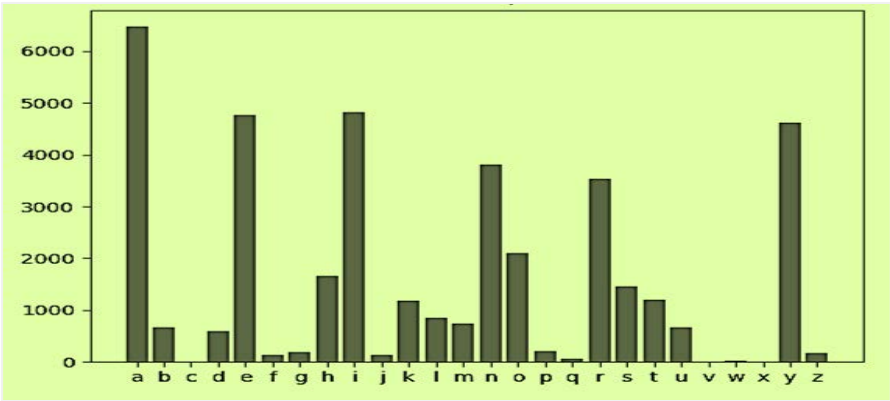


**Figure 2**: Terminal Frequencies of words in Corpus.

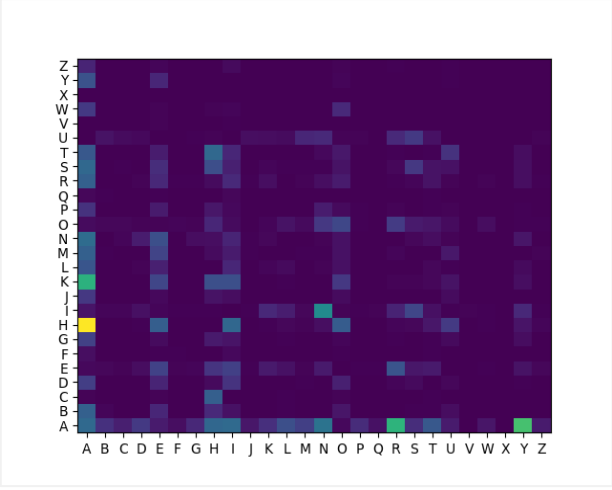


**Figure 3**: Transitional Frequencies of bigrams

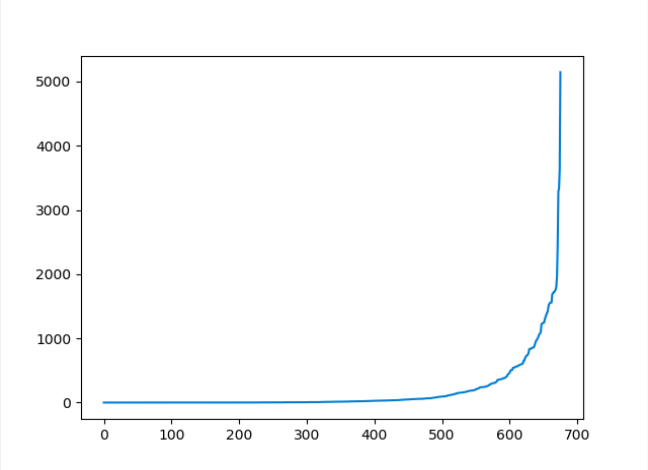


**Figure 4**: Exponential Transitional Distribution

## 3.2 ANALYSIS OF URDU DATA

The same procedure has been opted for analysis of Urdu Data and corresponding graphs for Emission Frequencies, Terminal Frequencies, Transition Frequencies, Transitional Distribution, and Word length distribution has been plotted and presented in Figure 6 till Figure 10. The numbers are

used for plotting convenience. The order in which the numbers are assigned to the alphabets is outlined in the Appendix in the end. Figure 6 and Figure 7 shows that most Urdu Words has started from starting alphabets in Urdu and the same is true for terminal frequencies in Urdu. This is not the case with English alphabets and they seem to scattered and appeared at regular intervals in search space. Beside frequency analysis, it has been observed from Urdu Data Heat map that there are many dark purple areas. What it means is, that there are some transitions that would never occur in Urdu language. This key observation and knowledge can be used to cut of search space branches to make any transliteration algorithm faster.

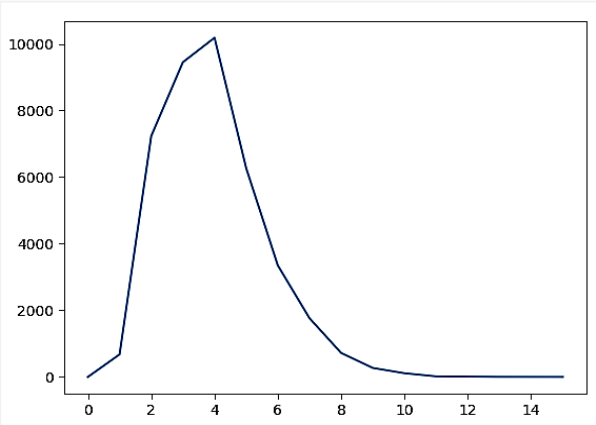


**Figure 5**: Gaussian distribution.

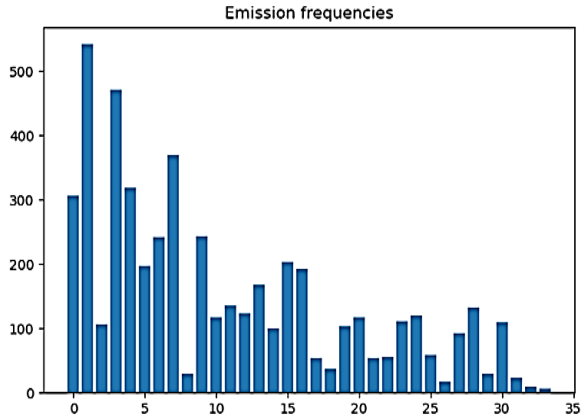


**Figure 6**: Urdu emission frequencies

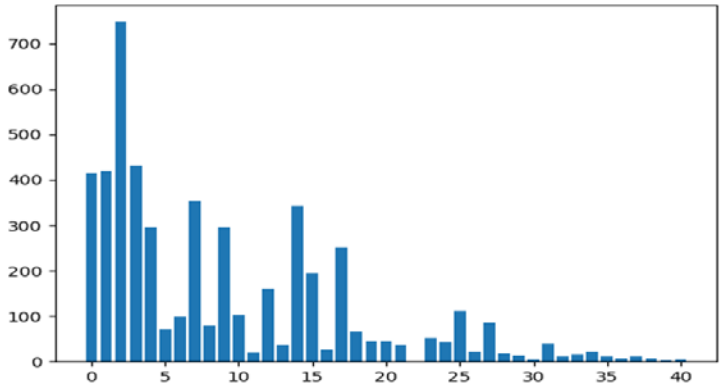


**Figure 7**: Terminal frequencies in Urdu data

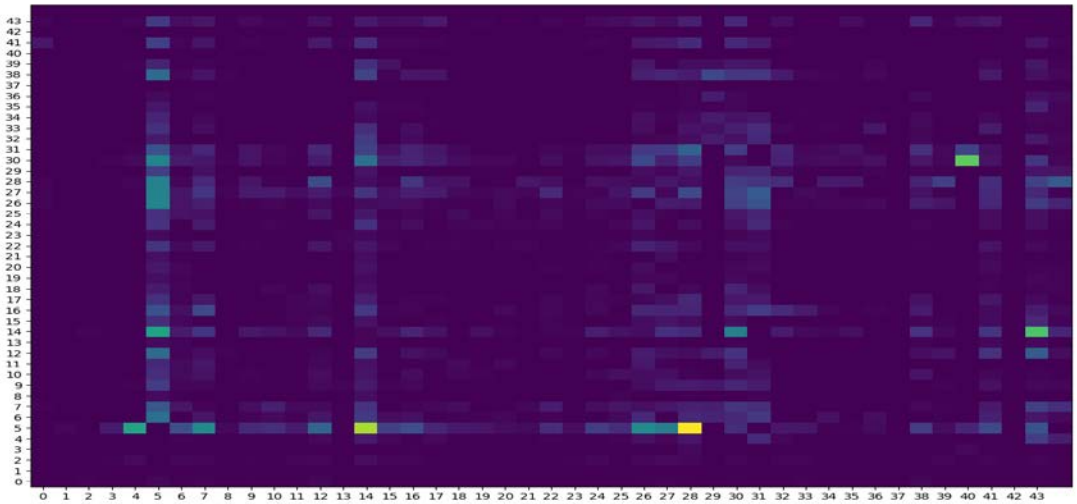

**Figure 8**: Heat Map for transitional frequencies of Urdu Data.

*Corresponding author (H. Tariq). Tel: +92-21-3012387949. E-mail: humera@uok.edu.pk  Transaction Journal of Engineering, Management, & Applied Sciences & Technologies. Volume 10 No.3 ISSN2228-9860 eISSN1906-9642 http://TUENGR.COM/V10/567.pdf DOI: 10.14456/ITJEMAST.2019.53

## 4. PROPOSED ALGORITHM DESCRIPTION

The Proposed algorithm consists of two main concepts which are not very evident from the algorithm directly. These are: (1) Parse Tree Generation (2) Transliteration Space tree (3) Heuristic-Based Searching. The very first thing that algorithm is the Generation of a parse tree from a Roman word. It simply slides a window on the roman string with a width value of 1 and 2 and records the branching as shown in Figure 9 for the word "ghar". We intentionally chose this ambiguous word to demonstrate the ability of the algorithm to get all right combinations. i.e. "g-h-a-r" , "gh-a-r". Note the use of dollar sign which reflects the start of an input string.

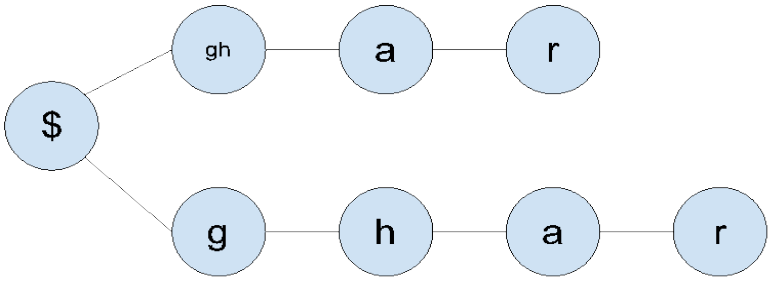


**Figure 9**: Parse tree generation.

**Table 1**: Transliteration space tree sample.

| Alphabet | Translation |
|---|---|
| g | گ غ ج |
| h | ه ه ح |
| a | آ ء ا ے ع $ |
| r | ر |
| gh | غ |

After extraction of possible phoneme combinations for transliteration, we can move on to build the transliteration tree. For this purpose, we have a handcrafted transliteration map object. A partial object sufficient for the demonstration of an example given in Table 1. The complete object description can be found in the Appendix in the end. We create a tree with an empty root node and then take phonemes one by one and put their translation from the map onto the tree by increasing depth level each time. While doing so we keep updating the remaining word i.e. the suffix. A fully constructed tree for the word "ghar" is shown in Figure 10 (a) and Figure 10(b). As you can see in Figure 10(a), that proposed algorithm will find all possible translations. The key strength of the algorithm is that it will "NEVER" miss a correct transliteration but will contain a lot of wrong translations. Let's calculate how much combinations it has to go through. Let *s* be the string to be transliterated. Let *n* be the length of the string *s*. Let the $i^{th}$ be chthe aracter in the string. Let $t_i$ be the substitution array for the $i^{th}$ character in the string. Let $|t_i|$ be the length of $t_i$. If the total possible translations is denoted by $k$ then its computation can be expressed as following product formula. For example, in our case, the possible paths are calculated as $2 \times 3 \times 5 \times 2 = 60$. As we can see this number can go up to thousands for large strings for which we introduce the greedy heuristic search in the next section for efficient path unfolding.

$$k = \prod_{i=0}^{n} |t_i| \quad (1)$$

Now we have established the crude form of algorithm to improve the performance of the function.

The key idea can be expressed in these words: *"Keep track of the growth of the branches via a scoring function that scores in proportion to the likelihood of the branch transition occurring in a real word which was learned from the collected Urdu data."*

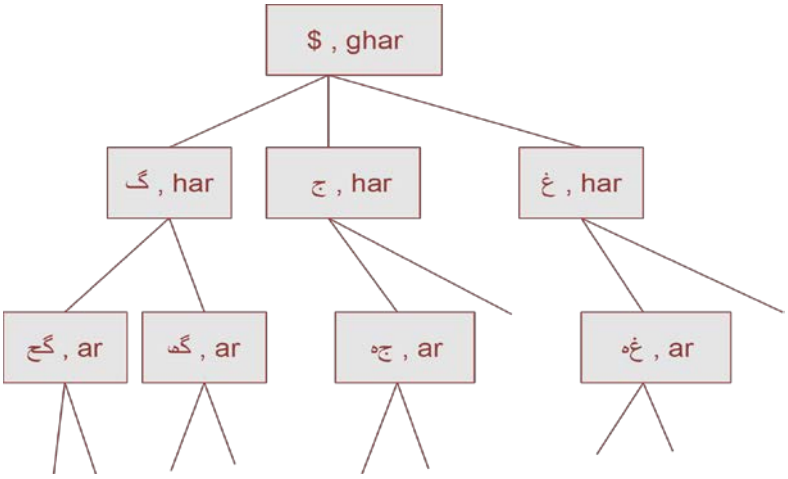


**Figure 10** (a): Transliteration Tree for Roman Word Ghar

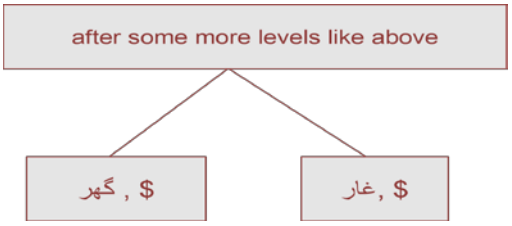


**Figure 10** (b): Leaf Nodes of Transliteration Tree for Roman Word Ghar

This is, in fact, a well-known technique in the field of Artificial Intelligence. Such scoring function is called a Heuristic function. In order to design a good scoring function, we must know the factors that contribute to the good solution being found. For this purpose, we can observe many important characteristics in the Roman language. Some of which are: (1) Even when one English phoneme translates to Urdu phoneme there is a significant bias towards a particular choice. For example, **g** can be used for both گ and ج but in practice, there is a significant bias towards using it only for the prior one. (2) When a biography phoneme occurs it is less likely to be translated as two separate uni-graphic phonemes like ch, gh, kh. This means when a bi-graphic phoneme gh occurs in a word it is highly likely that it will correspond to the Urdu letter. Keeping in mind above points and already computed transition matrix we built a heuristic function like below.

$$h_{i \to n} = A_0 + Z_n \sum_{i=1}^{n} trans(u_i, u_{i-1}) + bias(r_i, u_i) \quad (2),$$

where $A_i$ is emission frequency of the $i$-th character, $Z_i$ is terminal frequency of the $i$-th character $U_i$ is the $i$-th urdu character, $R_i$ is the $i$-th roman character, trans($a$, $b$) is the transition frequency from character $a$ to character $b$, bias($j$, $k$) is the bias of substitution map from roman character to urdu character. Putting in all together, the algorithm pseudocode has been provided in Table 3. Might be readers feel and notice some of the steps were omitted. That is due to the redundancies in the process which were identified and removed to improve the performance of the algorithm. Some key changes for optimality are: (1) We merged the process of parsing and translation into one by keeping track of

*Corresponding author (H. Tariq). Tel: +92-21-3012387949. E-mail: humera@uok.edu.pk 

the remaining suffix in the node of the translation tree. (2) We actually got rid of the tree data structure and replaced it with the DFS algorithm using a stack. (3) We omit the branches that are of the lowest score after every iteration. With the implementation of the above optimization techniques, we saw a significant improvement in the runtime complexity and memory consumption of the algorithm. After the implementation, we ran the algorithm on the data we collected in the earlier sections as well as some fabricated data and made observations of whether the algorithm performs as expected. A description of data and variables has been provided in Table 2 as they will be used to describe algorithm pseudocode.

**Table 2**: Variables and Data to be used in the Proposed Algorithm

| Terminology | Description | Terminology | Description |
|---|---|---|---|
| [] | 1D Array | T | Transition frequencies of Urdu |
| [] [] | 2D Array or Matrix | A | emission frequencies of Urdu |
| {} | hash map | Z | terminal frequencies of Urdu |
| R | results | H | heuristic function for decision of expansion of path in the tree |
| - | - | M | Map containing handcrafted substitution values for English phonemes to Urdu phonemes |

**Table 3**: The proposed heuristic-based algorithm for Roman to Urdu transliteration.

```
    DATA :
    R[] , T[][] , A[] , Z[] ,H() ,M{}
    Node : {
      suffix : roman string
      score : 0
 translation : empty string
    }
    STEPS :
1. initialize an empty stack with the above Node
2. while the stack is not empty
      a. pop node n from the stack
      b. if n.suffix has 0 length
           add to results
           continue
      c. let u <- n.suffix[0]
      d. let sub <- M[u]
              for each s in sub:
                 1. push new node stack with {
                      suffix : n.suffix[1:]
                      trans +: s
                      score +: Heuristic Function
                    }
      e. let b <- n.suffix[0:2]
      f. let sub <- M[u]
              for each s in sub:
                 1. push new node stack with {
                      suffix : n.suffix[2:]
                      trans +: s
                      score +: Heuristic Function
                    }
3. sort the results by score
4. for r in results :
    if r.score < threshold
      drop r
```

## 5. RESULTS AND DISCUSSION

We tested the algorithm on a variety of input lengths and plotted the runtime complexity of the algorithm. The x-axis represents the input length and Y-axis represents a number of a second required

to complete the translation for 100 words. Figure 11 shows that although the complexity is almost exponential even then the proposed algorithm is fast enough in practical scenarios. Table 4 contains some examples of the algorithms Accuracy. Notice that the algorithms translated all the words correctly but in the last two cases of *maloom* and *jadeed* it ranked the word a little lower than expected. This is a shortcoming of the heuristic function. Mitigations of which are suggested in the improvement section. We also tested that the proposed algorithm is decent enough for batch processing use with almost never making a mistake. Correct transliteration with high score has been found up to 78%, with a low score they found to be 21% while the wrong transliteration would be only 0.53% for all tested words.

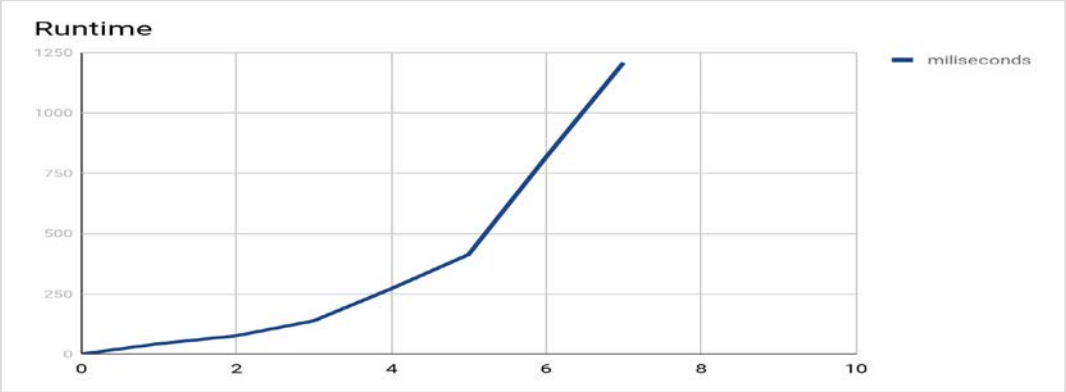


**Figure 11**: Performance is measured in terms of Processed Words per milliseconds.

**Table 4**: Performance demonstration table.

| | |
|---|---|
| tum | { توم : 334 } , { تم : 266 } , { ٹوم : 106 } , { طوم : 77 } , { ٹم : 61 } , { طم : 39 } |
| phool | { پہوول : 405 } , { پہول : 395 } , { پہول : 395 } , { پھوول : 383 } , { پھول : 373 } , { پہول : 373 } , { پہوُول : 352 } , { پحوول : 346 } , { پھوُول : 338 } , { پحول : 336 } , { پحول : 336 } , { پحوُول : 323 } , { پہوؤل : 318 } , { پھوؤل : 296 } , { پہل : 294 } , { پہوُل : 272 } , { پہوُل : 272 } , { پہوؤل : 272 } |
| maloom | { مالووم : 1002 } , { مالوم : 992 } , { مالوم : 992 } , { مالوؤم : 961 } , { مالؤوم : 920 } , { مالم : 915 } , { مالؤم : 886 } , { مالؤم : 886 } , { مالؤم : 886 } , { معلووم : 739 } , { معلوم : 729 } , { معلوم : 729 } , { ملووم : 722 } , { ملوم : 712 } , { ملوم : 712 } , { معلوؤم : 698 } , { ملوؤم : 681 } , { مآلووم : 664 } , { مےلووم : 663 } , { مءلووم : 658 } , { معلوؤم : 657 } , { مآلوم : 654 } , { مآلوم : 654 } , { مےلوم : 653 } , { مےلوم : 653 } , { معلم : 652 } |
| jadeed | { جادید : 482 } , { جادیڈ : 478 } , { جادعد : 388 } , { جادےد : 370 } , { جادےڈ : 370 } , { جادعڈ : 365 } , { جادد : 362 } , { جادڈ : 357 } , { جادءد : 357 } , { جادءڈ : 357 } , { جعدید : 318 } , { جعدیڈ : 314 } , { جدید : 302 } , { جےدید : 299 } |

## 6. CONCLUSION

An efficient algorithm for Roman to Urdu Transliteration has been proposed in the form of a library for developers to facilitate their users with transliteration as a feature. All the source code and the documentation can be found in the open source repository of the project at Qazi (2017a). All computed results can also be found in the CSV and JSON format at Qazi (2017b). Correct

*Corresponding author (H. Tariq). Tel: +92-21-3012387949. E-mail: humera@uok.edu.pk 

transliteration with high score has been found up to 78%, with a low score they found to be 21% while the wrong transliteration would be only 0.53% for all tested word. In future augmentation of the proposed algorithm with a recurrent neural network more specifically an LSTM to maintain a sense of context. Genetic algorithms can be used as a second option to evolve heuristic function with collected data instead of trying to handcraft it.

## 7. APPENDIX

1. Data extraction Code Snippets
    a. Generate Transition Frequencies
    https://github.com/devqazi/roman-urdu/blob/master/scripts/build_frequency_map.py
    b. Collect Emission Frequencies
    https://github.com/devqazi/roman-urdu/blob/master/scripts/emission_frequencies.py
    c. Collect Terminal Frequencies
    https://github.com/devqazi/roman-urdu/blob/master/scripts/terminal_frequencies.py
2. Data Visualization Code Snippets
    a. Visualize Transition Frequencies
    https://github.com/devqazi/roman-urdu/blob/master/scripts/visualize_freq.py
    b. Visualize Emission Frequencies
    https://github.com/devqazi/roman-urdu/blob/master/scripts/visualize_emissions.py
    c. Visualize Terminal Frequencies
    https://github.com/devqazi/roman-urdu/blob/master/scripts/visualize_terminals.py
    d. Visualize Empty Bigrams
    https://github.com/devqazi/roman-urdu/blob/master/scripts/visualize_empty_bigrams.py
    e. Visualize Frequency Distribution
    https://github.com/devqazi/roman-urdu/blob/master/scripts/visualize_freq_dist.py

## 8. REFERENCES


Ahmed, T., (2009). Roman to Urdu transliteration using Wordlist. In Proceedings of the Conference of Language and Technology (CLT'09).

Alam, M., Hussain S. (2017). Sequence to Sequence Networks for Roman-Urdu to Urdu Transliteration. 20th International Multioptic Conference (INMIC' 17).

Bilal M. et al. (2016). Sentiment classification of Roman-Urdu opinions using Naïve Bayesian, Decision Tree and KNN classification techniques. *Journal of King Saud University - Computer and Information Sciences.* 28(3), 330-344.

Faruqi M., Majumdar P., Pado, B. (2011). Soundex-based Translation Correction in Urdu–English Cross-Language Information Retrieval. *Proceedings of the 5th International Joint Conference on Natural Language Processing*, p25–29, Chiang Mai, Thailand, November 8-12.

Hellsten, L. et. al. (2017). Transliterated Mobile Keyboard Input via Weighted Finite-State Transducers. *Proceedings of the 13th International Conference on Finite State Methods and Natural Language Processing*, Umeå, Sweden, 4–6 September 2017. Association for Computational Linguistics, 10–19.

Knuth, D. E. (1975). Fundamental Algorithms, volume III of The Art of Computer Programming. Addison-Wesley, Reading, MA.

Moisl, H. (2009) Sura Length and Lexical Probability Estimation in Cluster Analysis of the Qur'an. *ACM Transactions on Asian Language Information Processing (TALIP).*8 (4), 1-19.

Naseem, T., (2004). A Hybrid Approach for Urdu Spell Checking. *Nust University of Computer and Emerging Sciences (NUCES).* MS Thesis, Lahore, Pakistan.

Russel, R. C. (1918). U.S. patent number no. 1261167.

Qazi, S. (2017a) https://www.github.com/devqazi/roman-urdu

Qazi, S. (2017b) https://github.com/devqazi/roman-urdu/tree/master/res

**Sayam Qazi** is a computer science graduate from Department of Computer Science, University of Karachi, where he received his BSCS. He is a professional developer and served in corporate sector while he was studying computer science. Mr. Sayam current interests involve applications of emerging technologies to computer science.

**Dr. Humera Tariq** is an Assistant Professor at Department of Computer Science, University of Karachi, Pakistan. She received her B.Eng.(Electrical Engineering) from NED University of Engineering and Technology and Ph.D. (Computer Science) from University of Karachi. Her focuses of research include computer graphics, image processing, neural networks and simulation.

*Corresponding author (H. Tariq). Tel: +92-21-3012387949. E-mail: humera@uok.edu.pk  Volume 10 No.3 ISSN2228-9860 eISSN1906-9642 http://TUENGR.COM/V10/567.pdf DOI: 10.14456/ITJEMAST.2019.53