The Journal of Open Source Software

Graph Transliterator: A graph-based transliteration tool

A. Sean Pue[1]

1 Linguistics and Germanic, Slavic, Asian, and African Languages, Michigan State University

DOI: 10.21105/joss.01717

Software

- Review
- Repository
- Archive

Editor: George K. Thiruvathukal

Reviewers:

- @rlskoeser
- @vc1492a

Submitted: 28 July 2019
Published: 01 December 2019

Summary

Transliteration—the representation of one language or script in the characters or symbols of another—is a ubiquitous and important operation, used across the humanities, social sciences, and information sciences, as well as other fields. It enables text to be read by those who do not know the original alphabet, and also makes languages or related languages written in multiple scripts legible to readers conversant in only one script (Durrani, Sajjad, Fraser, & Schmid, 2010; Saini, Lehal, & Kalra, 2008). Transliteration enables the standardized organization and search of resources, as in library systems (Barry, 1997). It also permits the encoding of essential information often not found in the original script, such as morphological boundaries and unwritten elements, permitting disambiguation. In natural language processing tasks, transliteration has opened up new possibilities, especially in machine translation (Prabhakar & Pal, 2018) and named-entity recognition (Chen, Banchs, Zhang, Duan, & Li, 2018; Merhav & Ash, 2018).

`Graph Transliterator` is a Python package and command-line program that makes this process more accessible by using a standardized method for encoding rules for transliteration. It lets those rules be entered in an "easy reading" YAML format (Ben-Kiki, Evans, & döt Net, 2009) or directly, using standard Python data types. It also includes bundled transliterators that are rigorously tested and to which users can contribute. It differs from other rule-based software (Unicode Consortium, 2019) designed for handling transliteration in two primary ways. First, other software works directly on an input string, performing operations based on matches of particular characters. `Graph Transliterator` instead tokenizes the input into user-defined transliteration token types. Then it applies transliteration rules defined for those token types, rather than matching and manipulating the original characters of the input string. Second, other software requires a defined sequence of transliteration operations. `Graph Transliterator` instead automatically orders its transliteration rules so that the rule involving the largest number of tokens is applied first.

Each instance of `Graph Transliterator` is parameterized by the acceptable token types of the input string as well as by transliteration rules. The transliteration token types can be one or more letters in length, e.g. `a`, `b`, or `aa`. Each of the token types can be assigned to particular classes, e.g. `vowel` or `consonant`. The transliteration rules allow matching of a particular sequence of one or more tokens. They also allow lookahead and lookbehind matching for particular tokens or token classes.

`Graph Transliterator` includes features that accommodate common tasks involved in transliteration and associated forms of analysis. It includes customizable rules for defining and handling whitespace, which is often very important in transliteration, as many letters in non-Roman alphabets change their shape at the start of words. It accepts "on match" rules for the insertion of output based on which token classes are matched, e.g. the insertion of a character between two consonants. `Graph Transliterator` makes it possible for users to view the details about rule-matching, which may enable certain forms of analysis. It also allows the matching of all possible rules at a given index, which can be useful in particular analysis contexts. Each instance of a defined transliterator can be serialized and includes metadata

fields. Finally, the software includes full ambiguity checking and produces a warning if more than one rule could match the same input. As such, `Graph Transliterator` provides a rigorous and reproducible framework for transliteration.

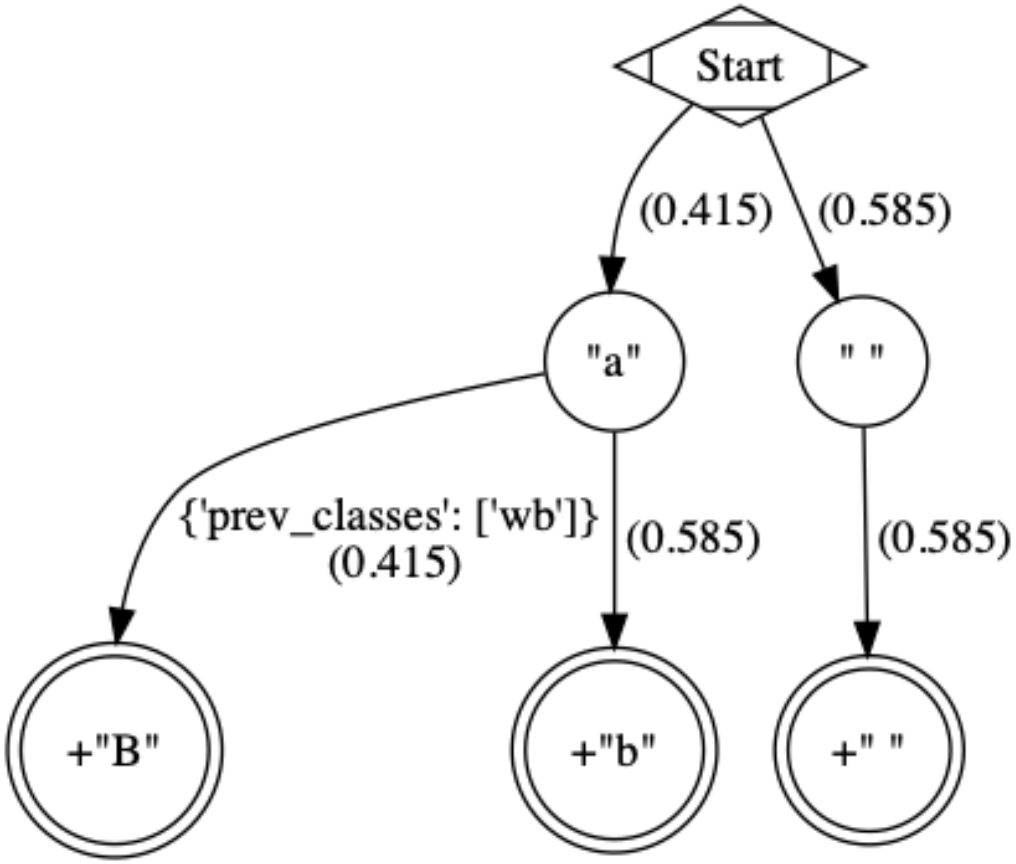

Figure 1: An example graph created for the simple case of a `Graph Transliterator` that takes as input two token types, `a` and `" "` (space), and renders `" "` as `" "`, and `a` as `b` unless it follows a token of class `wb` (for wordbreak), in which case it renders `a` as `B`. The `rule` nodes are in double circles, and `token` nodes are single circles. The numbers are the cost of the particular edge, and less costly edges are searched first. Previous token class (`prev_classes`) constraints are found on the edge incident to the leaf rule node.

How It Works

During initialization, a graph is created that is searched to find the best transliteration match at a particular index in the tokens of an input string. The graph, a directed tree, has nodes of three types: `Start`, `token`, and `rule`. The `Start` node is the root. A `token` node corresponds to a token being matched. The `rule` nodes are leaves representing transliteration rules. Each transliteration rule is assigned a particular cost between one and zero that lessens with more tokens, using the following cost function:

$$\text{cost}(rule) = \log_2\left(1 + \frac{1}{1 + \text{count\_of\_tokens\_in}(rule)}\right)$$

Each edge is assigned a cost corresponding to the least costly transliteration rule leaf node that can be reached from it. Edges contain constraints that must be met before a node can

be visited. Before token nodes, these constraints include a token in the input to be matched. Other constraints include previous and/or following tokens or token classes. To optimize the search, during initialization an `ordered_children` dictionary is added to each non-leaf node. Its values are a list of node indexes sorted by cost and keyed by the token that follows. Any rule immediately following a node is added to `ordered_children` as well as to each individual entry in it. Because of this preprocessing, `Graph Transliterator` does not need to iterate through all of the outgoing edges of a node to find the next node to search. Instead, it uses a best-first search implemented using a stack, and will backtrack if necessary to find the best match.

Graph Transliterator is available at https://github.com/seanpue/graphtransliterator under the MIT License. Detailed installation and usage instructions are available at https://graphtransliterator.readthedocs.io.

Acknowledgements

Software development was supported by an Andrew W. Mellon Foundation New Directions Fellowship (Grant Number 11600613) and by matching funds provided by the College of Arts and Letters, Michigan State University.

References

Barry, R. K. (1997). *ALA-LC romanization tables: Transliteration schemes for non-roman scripts*. Library of Congress & American Library Association. Retrieved from https://www.loc.gov/catdir/cpso/roman.html

Ben-Kiki, O., Evans, C., & döt Net, I. (2009). *YAML Ain't Markup Language (YAML) Version 1.2*. Retrieved from https://yaml.org/spec/1.2/spec.html

Chen, N., Banchs, R. E., Zhang, M., Duan, X., & Li, H. (2018). Report of news 2018 named entity transliteration shared task. In *Proceedings of the seventh named entities workshop* (pp. 55–73). doi:10.18653/v1/w18-2409

Durrani, N., Sajjad, H., Fraser, A., & Schmid, H. (2010). Hindi-to-urdu machine translation through transliteration. In *Proceedings of the 48th annual meeting of the association for computational linguistics* (pp. 465–474). Association for Computational Linguistics.

Merhav, Y., & Ash, S. (2018). Design challenges in named entity transliteration. In *Proceedings of the 27th international conference on computational linguistics* (pp. 630–640). Santa Fe, New Mexico, USA: Association for Computational Linguistics. Retrieved from https://www.aclweb.org/anthology/C18-1053

Prabhakar, D. K., & Pal, S. (2018). Machine transliteration and transliterated text retrieval: A survey. *Sādhanā*, *43*(6), 93. doi:10.1007/s12046-018-0828-8

Saini, T. S., Lehal, G. S., & Kalra, V. S. (2008). Shahmukhi to gurmukhi transliteration system. In *22nd international conference on on computational linguistics: Demonstration papers* (pp. 177–180). Association for Computational Linguistics.

Unicode Consortium. (2019). *General Transforms - International Components for Unicode User Guide*. Unicode, Inc. Retrieved from http://userguide.icu-project.org/transforms/general