DAFSA: a Python library for Deterministic Acyclic Finite State Automata

Tiago Tresoldi[1]

1 Department of Linguistic and Cultural Evolution, Max Planck Institute for the Science of Human History

DOI: 10.21105/joss.01986

Software

- Review
- Repository
- Archive

Editor: George K. Thiruvathukal

Reviewers:

- @geoffbacon
- @desilinguist

Submitted: 12 December 2019
Published: 19 February 2020

Summary

This work describes dafsa, a Python library for computing graphs from lists of strings for identifying, visualizing, and inspecting patterns of substrings. The library is designed for usage by linguists in studies on morphology and formal grammars, and is intended for faster, easier, and simpler generation of visualizations. It collects frequency weights by default, it can condense structures, and it provides several export options. Figure 1 depicts a basic DAFSA, based upon five English words and generated with default settings.

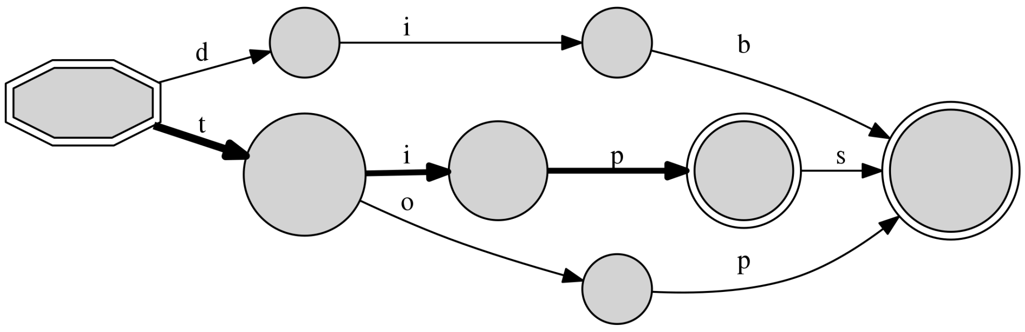

Figure 1: Visual representation of a DAFSA for the list of strings `"dib"`, `"tip"`, `"tips"`, and `"top"`.

Background

Deterministic Acyclic Finite State Automata (DAFSA, also known as "Directed Acyclic Word Graphs", or DAWG) are data structures extended from tries and used to describe collections of strings through directed acyclic graphs with a sole source vertex (the `start` of all sequences), at least one sink node (each pointed to by one or more edges), and edge labels carrying information on the sequence of characters that form the strings (Black & Pieterse, 1998; Blumer et al., 1985; Lucchesi & Kowaltowski, 1993). A compact variant (Crochemore & Vérin, 1997) condenses the structure by merging every node which is an only child with its parent, concatenating their labels. The resulting graph is a particular finite state recognizer, accepting all and only the strings from the original list.

DAFSAs are mostly used for the memory-efficient storage of sets of strings, such as in spelling correction and in non-probabilistic set membership check (Blumer et al., 1985; Ciura & Deorowicz, 2001; Havon, 2011; Lucchesi & Kowaltowski, 1993). While there have been proposals for applying them to the treatment and analysis of pattern repetitions, especially in genomics

(Crochemore & Vérin, 1997), no general-purpose library designed for such exploration and visualization is available. In specific, as a consequence of most implementations being designed for an efficient set membership testing, no available library builds DAFSAs that collect node and edge frequency.

Installation, Usage, & Examples

The library can be installed with the standard `pip` tool for package management:

```
$ pip install dafsa
```

The documentation offers detailed instructions on how to use the library. For most purposes it is sufficient to create a new `DAFSA` object and initialize it with the list of strings, as in the generation of the graph for Figure 1:

```
>>> from dafsa import DAFSA
>>> d = DAFSA(["dib", "tap", "top", "taps", "tops"])
```

The library will by default collect frequency weights for each edge and node. We can export the resulting structures in either a custom textual format (using the standard `repr()` command) or in GML format (using the `.write_gml()` method), or convert them to equivalent graphs in the `networkx` library (using the `.to_graph()` method). Visualizations can be generated through DOT source code (using the `.to_dot()` method), and manipulated according to the users' preferences and needs. An auxiliary `.write_figure()` method allows to generate figures in PNG, SVG, or PDF format if `graphviz` is available, as in Figure 2.

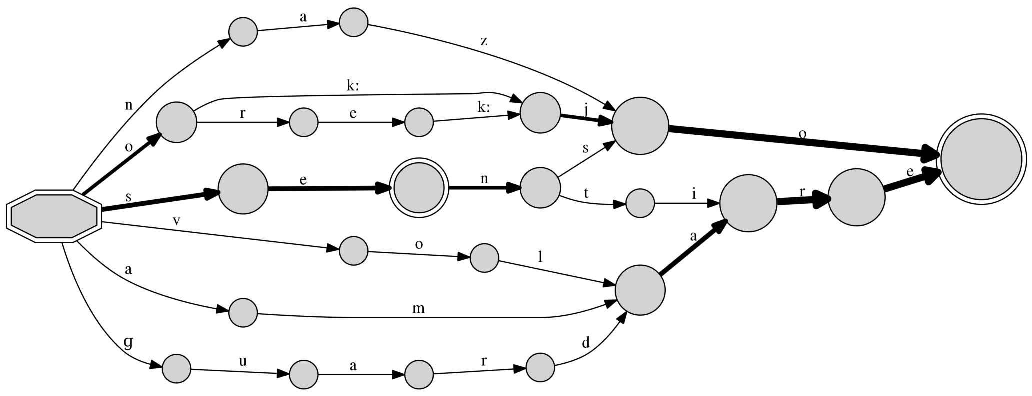

Figure 2: A non-condensed DAFSA for a list of 9 Italian words phonetically transcribed.

Graphs are not condensed by default, as in Figure 2, but condensation can be performed by setting the `condense` flag when initializing the object, as done in the following code snippet and illustrated in Figure 3:

```
>>> words = [
    "o k: j o", "o r e k: j o", "n a z o",
    "s e", "s e n t i r e", "s e n s o",
    "g u a r d a r e", "a m a r e", "v o l a r e"]
>>> words = [word.split() for word in words]
>>> d = DAFSA(words, condense=True)
```

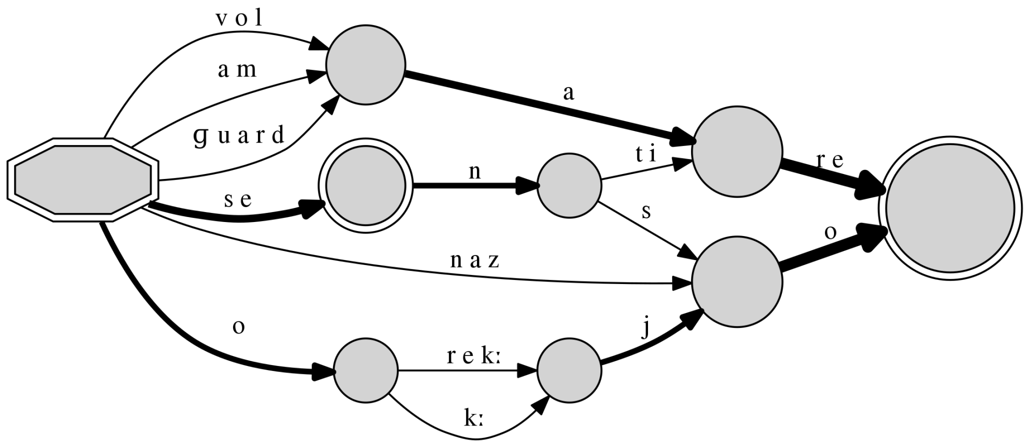

Figure 3: A condensed DAFSA for the same list of 9 Italian words phonetically transcribed used in Figure 2.

A command-line `dafsa` tool is provided along with the library. Assuming the data is found in a `phonemes.txt` file, with one sequence per line, a PDF version of Figure 3 can be generated with the following call, where `-c` instructs to condense the graph, `-t` specifies the output format, and `-o` the output file:

```
$ dafsa -c -t pdf -o phonemes.pdf phonemes.txt
```

Alternatives

The main alternatives to this library, such as the Python `DAWG` package, are based on `dwagdic` C++ library, designed for production usage of memory- and speed-efficient data structures. The unsupported `adfa` and `minim` packages by Daciuk, Mihov, Watson, & Watson (2000) are closer in intention, as well as the Python prototype by Havon (2011). Similar functionalities are offered by several tools for analysis of genetic data, usually as an extension of sequence alignments, but none as an autonomous tool that can be employed with generic lists of strings.

Code and Documentation Availability

The `dafsa` source code is available on GitHub at https://github.com/tresoldi/dafsa.

The documentation is available at https://dafsa.readthedocs.io/.

Acknowledgements

The author has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. ERC Grant #715618, Computer-Assisted Language Comparison).

References

Black, P. E., & Pieterse, V. (Eds.). (1998). Directed acyclic word graph. In *Dictionary of algorithms and data structures*. Gaithersburg: National Institute of Standards and Technology.

Blumer, A. C., Blumer, J. A., Haussler, D., Ehrenfeucht, A., Chen, M.-T., & Seiferas, J. I. (1985). The smallest automation recognizing the subwords of a text. *Theoretical computer science*, *40*, 31–55. doi:10.1016/0304-3975(85)90157-4

Ciura, M. G., & Deorowicz, S. (2001). How to squeeze a lexicon. *Software: Practice and Experience*, *31*(11), 1077–1090. doi:10.1002/spe.402

Crochemore, M., & Vérin, R. (1997). On compact directed acyclic word graphs. In *Structures in logic and computer science* (pp. 192–211). Springer.

Daciuk, J., Mihov, S., Watson, B. W., & Watson, R. E. (2000). Incremental construction of minimal acyclic finite-state automata. *Computational linguistics*, *26*(1), 3–16.

Havon, S. (2011). Compressing dictionaries with a dawg. Retrieved from http://stevehanov.ca/blog/?id=115

Lucchesi, C. L., & Kowaltowski, T. (1993). Applications of finite automata representing large vocabularies. *Software: Practice and Experience*, *23*(1), 15–30. doi:10.1002/spe.4380230103