QuranTree.jl: A Julia package for Quranic Arabic Corpus

Al-Ahmadgaid B. Asaad
UnionBank of the Philippines, Inc.
alahmadgaid@gmail.com

Abstract

QuranTree.jl is an open-source package for working with the Quranic Arabic Corpus (Dukes and Habash, 2010). It aims to provide Julia APIs as an alternative to the Java APIs of JQuranTree. QuranTree.jl currently offers functionalities for intuitive indexing of chapters, verses, words and parts of words of the Qur'an; for creating custom transliteration; for character dediacritization and normalization; and, for handling the Morphological Features. Lastly, it can work well with Julia's TextAnalysis.jl[1] and Python's CAMeL Tools (Obeid et al., 2020).

1 Introduction

The Quranic Arabic Corpus (Dukes and Habash, 2010) provides a complete annotation of the Morphological Features of the Qur'an. These features include part-of-speech tags, morphological segments, semantic ontology, and syntactic analysis using dependency graph. The corpus can be accessed through either the Java package, JQuranTree, or by downloading it as a Text (.txt) file from:

https://corpus.quran.com/

The corpus was made to produce a resource that enables further analysis of the Qur'an, the 1,400 year old central religious text of Islam. The 77,430 words of the Qur'an form a distinct genre difficult to compare to other texts of Arabic; and processing Quranic Arabic is a unique challenge from a computational point of view, since it differs significantly from Modern Standard Arabic (MSA) (Dukes and Habash, 2010).

In an effort to make the corpus accessible computationally, Dukes and Habash (2010) developed the Java package, JQuranTree. It includes APIs for accessing and analyzing the said corpus in its Arabic form, through the Uthmani distribution of the Tanzil[2] project. This distribution is encoded as a Orthographic Object Model, that is *immutable* and follows the following hierarchy: Document → Chapter → Verse → Token → Character → Diacritic. The package uses extended[3] Buckwalter (Buckwalter, 2002) encoding as transliteration for the Qur'an's Arabic texts, whilst also offering Simple Encoding[4] as another type of Arabic to Roman mapping. Lastly, JQuranTree supports APIs for Search and Analysis table for analyzing the corpus.

While JQuranTree provides all the fundamental functionalities for working with the corpus, it is, however, not easily accessible to researchers with little to no experience in programming. This is mainly due to the programming language used for the APIs, which is Java. Indeed, it is a powerful language but is only taught to *engineering* and *computer science* students, and rarely used in other non-engineering or non-IT courses. As such, it is therefore safe to assume, that most non-engineering or non-IT researchers who started coding for their research papers, are likely to be users of a high-level programming languages (relative to Java), like R, Python or Julia.

High-level languages are, by design, easier to learn compared to low-level languages. This is because high-level languages are often, if not always, aimed at abstracting some complexities of a low-level languages. On top of that, algorithms and models for scientific computing are often implemented in high-level programming languages. This is the motivation of QuranTree.jl, by providing APIs on a high-level language as an alternative module to JQuranTree's Java APIs, without com-

[1] https://github.com/JuliaText/TextAnalysis.jl

[2] http://tanzil.net/docs/tanzil_project

[3] https://corpus.quran.com/java/buckwalter.jsp

[4] https://corpus.quran.com/java/simpleencoding.jsp

Proceedings of the Sixth Arabic Natural Language Processing Workshop, pages 208–212
Kyiv, Ukraine (Virtual), April 19, 2021.

promising on the speed and robustness of the latter library.

While the availability of the Text file for the Quranic Arabic Corpus meant that anyone can load it into their favorite programming languages for analysis, it requires, however, heavy text processing on the Morphological Features, an advanced task for non-engineering researchers. It is, therefore, part of this project to abstract these complexities by providing intuitive high-level APIs.

Lastly, the fact that there are other libraries (*see* discussions in Section 2) dedicated for handling the Quranic datasets, simply suggests that there are growing interest for such tool, especially with the proliferation of computational approaches to *learning from the data*.

2 Related Work

There have been efforts on bringing the Qur'an into packages of a high-level languages for accessible analyses. For R language, for example, Heiss (2019) came up with the *quRan* library. The said library provides four types of datasets encoded as R data frames for the Qur'an: two in Arabic texts (with and without diacritics), and two as English translations (Saheeh International and Yusuf Ali). Each data frame contains columns describing the *descriptive* feature of the Surah or Chapter; for example, features like Surah ID, Ayah ID, Surah Title, etc.; and other indicator features like if a Sajdah is obligatory for the verse, or if it is recommended. As such, *quRan* lacks columns for Morphological Features of the Qur'an, which is useful for further analysis of the linguistic style of the said scripture.

For Python, on the other hand, Yousef et al. (2018) came up with *PyQuran*, which unlike *quRan*, this package provides extra utilities for analyses. For example, APIs for character dediacritization; Buckwalter transliteration; other functionalities like grouping letters based on its diacritics; and, search functionalities. However, like the *quRan*, *PyQuran* is not using the Quranic Arabic Corpus (Dukes and Habash, 2010), and thus lacking Morphological Features as well, which again are useful for the analysis of the linguistic style of the Qur'an, a step further than just the descriptive part.

3 Design and Implementation

QuranTree.jl is an open-souce library consisting of a set of Julia APIs for working with the Quranic Arabic Corpus (Dukes and Habash, 2010), and in this section, we are going to discuss the design and implementation of the package.

3.1 Design Philosophy

The main philosophy on the design of the QuranTree.jl, is to make it *Julia with little-else*. That is, as much as possible the package should only provide the nuts and bolts, so users won't have difficulty in learning the package and give them the opportunity to create their own wrappers for specific functionalities. Further, this philosophy should also revolve around the three main design features: *speed*, *robustness* and *modularity*.

3.2 Implementation

QuranTree.jl was implemented on top of Julia, a new general-purpose programming language with heavy emphasis on numerical and high-performance computing (Bezanson et al., 2017). There are several reasons as to why Julia was a good choice for the project, all of which are geared towards the three design features mentioned. In its core, Julia aims to solve the *two-language problem*, which describes the process where high-level languages like R and Python are often used for prototyping, and then are rewritten to low-level languages like Java and C at the operationalization stage, for optimal performance. In short, Julia aims to be as fast as C but with the friendly syntax like that of Python (or other high-level programming languages). The choice of Julia therefore addresses the first design principle, which is *speed*. Moving on, the expressive type trait of Julia makes it a good choice for creating a *robust* and *scalable* package. Note that, the type trait of Julia is optional, so Python and R users can easily skip it. Another trait of Julia is *Multiple-Dispatch*, a feature that enables users to create interface to other Julia libraries with minimal effort. This built-in feature plus the extensive Julia's interoperability packages like PyCall.jl and RCall.jl, enables QuranTree.jl to connect to other languages as well, making the package *modular*.

4 Installation

QuranTree.jl is a registered Julia package with source code available on Github[5], and can be installed in the Julia REPL as follows:

[5] https://github.com/alstat/QuranTree.jl

```
using Pkg
Pkg.add("QuranTree")
```

Or by pressing `]` in the said REPL and then enter `add QuranTree` to run the installation.

5 Package Features

At the time of writing, QuranTree.jl has the following features:

1. Indexing - Intuitive indexing for Chapters, Verses, Words and Parts of Words.
2. Transliteration - Buckwalter (Buckwalter, 2002) as default; has function for creating custom transliterator; and, can update transliteration in 1 line of code.
3. Morphological features - Complete type for all Morphological Features and Parts of Speech.
4. Seamless transition between Arabic and Buckwalter (or custom transliteration).
5. Simple Encoding[6]
6. Character Normalization - For both Arabic and Buckwalter (or custom transliteration).
7. Character Dediacritization - For both Arabic and Buckwalter (or custom transliteration).
8. Utilities - Function for displaying detailed description of the Morphological Features.
9. Modularity and Type-Safe - Can easily interoperate with Python (using PyCall.jl) and R (using RCall.jl) for packages that are not yet in Julia; Type-safe like JQuranTree.
10. Others - Read-only array for raw datasets (Quranic Arabic Corpus and Tanzil[7] Data).

In the following subsections, we will discuss how the design principles (*robustness*, *modularity*, and *speed*) are implemented in the package.

[6] https://corpus.quran.com/java/simpleencoding.jsp

[7] http://tanzil.net/docs/tanzil_project

5.1 *Robustness*: Morphological Features

Among the main efforts in the Quranic Arabic Corpus project, is the morphological annotation of the Qur'an down to the level of every part of every word. The definition for the tags of the Morphological Features are detailed in Dukes and Habash (2010), including the tags for Parts of Speech. In QuranTree.jl, all tags from the said article are defined as Julia's immutable `struct`s. For example, the type for Indicative and Subjunctive Verb features are express as part of a hierarchy of types shown in Figure 1. This hierarchy of types allows users to create methods for particular type or group of types. For example, methods for the parent type `AbstractVerb` are shared across all Verb types under it.

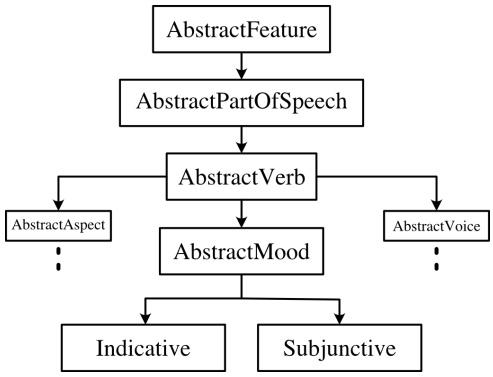

Figure 1: QuranTree.jl's Verb Mood Type Hierarchy.

These type hierarchy are useful for searching or filtering the corpus based on the Morphological Features as will be shown later. The Morphological Features in its raw form is a `String` that looks like this:

```
"STEM|POS:N|LEM:{som|ROOT:smw|M|GEN"
```

QuranTree.jl's `parse(Features, x)` parses String `x` into a Morphological Features object with `Features` as its type. This parsed input will have methods for processing the morphological annotations. For example, to check whether the parsed features is a particular Part of Speech, is done through the method `isfeature(parsed_x, Masculine)`, which checks whether the parsed features contains Masculine as part of the description of the token.

The expressive type system for the Morphological Features of the QuranTree.jl, allows users to create *robust* wrappers based on these types – something that is lacking in other high-level languages like R and Python, where both have the risks of

accepting any input to any defined wrappers or functions due to lack of type capabilities, out-of-the-box.

5.2 *Modularity*: Working with NLP packages

In studying the linguistic style of the Qur'an, computationally, researchers are expected to use some Natural Language Processing algorithms. QuranTree.jl was designed to be as modular as possible to existing NLP toolkits both in Julia and other programming languages. Succeeding subsections will illustrate how to use QuranTree.jl with Julia's leading NLP toolkit (TextAnalysis.jl[8]), and how it interoperate with recently released Python's leading (in terms of features) Universal Arabic NLP toolkit, the CAMeL Tools (Obeid et al., 2020).

5.2.1 TextAnalysis.jl

TextAnalysis.jl is the leading (in terms of features) library for analyzing texts in Julia. It offers utilities for basic NLP modeling but with close integration to Flux.jl (Innes, 2018) — Julia's leading (in terms of popularity) Deep Learning library. TextAnalysis.jl is specifically useful for creating feature matrix that can later be used as input to any Machine Learning models. An example of this, is on applying TextRank on summarizing Qur'an's Chapter 18 (The Cave) detailed in QuranTree.jl's documentation[9]. TextAnalysis.jl, however, was built for general text analyses, and working with specific feature of a language is obviously a limitation. For example in Arabic, disambiguation of *all-consonants* token is a problem in itself, which does not occur in English (default language assumed in TextAnalysis.jl) or other non-Semitic languages.

5.2.2 CAMeL Tools

At the time of writing, by far, the first package to offer toolkit for Universal Arabic NLP in terms of features, is the project from New York University Abu Dhabi, known as the CAMeL Tools (Obeid et al., 2020). The said library is written in Python, and offers a suite of tools for handling and modeling Arabic texts. For example, it has methods for analyzing, generating and reinflecting a given Morphological Features of a token. The said package has functions for the following: Disambiguator, Tagger, Tokenizers, Dialectic Identification, Sentiment Analysis and Named Entity Recognization. All tailored specifically for Arabic language.

QuranTree.jl can take advantage of the Python APIs available in CAMeL Tools. This is possible with the extensive list of Julia's interoperation libraries to other languages, including Python via the package PyCall.jl. As an example, Figure 2 illustrates the disambiguation of a dediacritized *basmala*. In this figure, CAMeL APIs work seamlessly in Julia, specifically with QuranTree.jl's APIs like: `dediac`, `encode` and `normalize`.

```
[42]: using PyCall
      @pyimport camel_tools.disambig.mle as camel

[43]: mled = camel.MLEDisambiguator.pretrained()

[43]: PyObject <camel_tools.disambig.mle.MLEDisamb
      iguator object at 0x7fbc48912820>

[44]: dvrs = dediac(vrs)

[44]: "بسم ٱلله ٱلرحمٰن ٱلرحيم"

[45]: dis = mled.disambiguate(split(dvrs));
      out = [d[2][1][2]["diac"] for d in dis]
      out = join(out, " ")

[45]: "بِسَمِ ٱللهِ ٱلرَحۡمَٰنِ ٱلرَحِيمِ"

[46]: encode(out)

[46]: "bisam~ All~`h AlraHom`n AlraHiym"

[47]: encode(vrs)

[47]: "bisomi {ll~ahi {lr~aHoma`ni {lr~aHiymi"

[48]: normalize(encode(vrs))

[48]: "bisomi All~ahi Alr~aHomaAni Alr~aHiymi"
```

Figure 2: CAMeL Tools' disambiguation on dediacritized *basmala*.

As shown in the figure, the simplicity of the `@pyimport` macro enables users to use Python APIs with little boilerplate. Further, this seamless interoperability allows QuranTree.jl to be modular since Julia APIs can now work well with Python APIs from the CAMeL Tools.

Needless to say, all other functionalities in CAMeL Tools not illustrated here are also accessible to QuranTree.jl.

5.3 *Speed*: Experiment

Finally, for the third design feature, we attempt to compare CAMeL Tools' APIs on transliteration and dediacritization against the QuranTree.jl's APIs. The task is to transliterate the Uthmani Corpus to Buckwalter, then dediacritized, then encode again to its Arabic form. The task was replicated

[8] https://github.com/JuliaText/TextAnalysis.jl

[9] https://alstat.github.io/QuranTree.jl/dev/man/nlp/text_summarization/

for 16 times and summarized into the following table:

Metric	CAMeL Tools	QuranTree.jl
Min	342.3ms	418.3ms
Mean	352.4ms	434.9ms
Median	350.6ms	436.2ms
Max	375.6ms	452.7ms

Table 1: Speed of CAMeL Tools vs QuranTree.jl in milliseconds (ms).

While QuranTree.jl is aimed at *speed*, at the time of writing, the transliteration and dediacritization APIs are not well optimized compared to that of CAMeL Tools, as evident in the table. Indeed, this optimization is still a work-in-progress.

6 Conclusion and Future Work

In conclusion, QuranTree.jl provides the nuts and bolts for doing Arabic text analysis for the Quranic Arabic Corpus in Julia, with the main advantage of being able to work seamlessly with existing Julia's general NLP toolkit, TextAnalysis.jl; and seamlessly interoperate with other packages outside Julia, like CAMeL Tools. This *modularity* has indeed been part of the main design principles of the package, along with the *robustness* provided by the expressive type system of Julia, and of course the inherited *speed* – still a work-in-progress for the package.

There are still room for improvements, however, given the fact that QuranTree.jl was just recently added to Julia's package registry, at the time of writing. Among the future works, the first one is the standardization of the mappings for dediacritization and normalization functionalities. Specifically, to align QuranTree.jl to JQuranTree and CAMeL Tools to avoid confusion and unnecessary adjustments; second, the binary encoding of the Text (.txt) files of the raw data in the backend; third, the development of a pure Julia library for Universal Arabic NLP, with close integration to QuranTree.jl. This is a huge effort but will indeed complete the Arabic NLP toolkit for Julia; and lastly, any suggestions to further improve the package are welcome, simply open an issue in the Github repository[10].

[10] https://github.com/alstat/QuranTree.jl

References

Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B Shah. 2017. Julia: A fresh approach to numerical computing. *SIAM review*, 59(1):65–98.

Tim Buckwalter. 2002. Buckwalter arabic morphological analyzer version 1.0. Linguistic Data Consortium, University of Pennsylvania.

Kais Dukes and Nizar Habash. 2010. Morphological annotation of quranic arabic. In *Proceedings of the International Conference on Language Resources and Evaluation, LREC 2010, 17-23 May 2010, Valletta, Malta*. European Language Resources Association.

Andrew Heiss. 2019. andrewheiss/quran: Initial cran release.

Mike Innes. 2018. Flux: Elegant machine learning with julia. *Journal of Open Source Software*, 3(25):602.

Ossama Obeid, Nasser Zalmout, Salam Khalifa, Dima Taji, Mai Oudah, Bashar Alhafni, Go Inoue, Fadhl Eryani, Alexander Erdmann, and Nizar Habash. 2020. CAMeL tools: An open source python toolkit for Arabic natural language processing. In *Proceedings of the 12th Language Resources and Evaluation Conference*, pages 7022–7032, Marseille, France. European Language Resources Association.

Waleed A. Yousef, Taha M. Madbouly, Omar M. Ibrahime, Ali H. El-Kassas, Ali O. Hassan, and Abdallah R. Albohy. 2018. Pyquran: The python package for quranic analysis. https://hci-lab.github.io/PyQuran-Private.