ROOTS: a toolkit for easy, fast and consistent processing of large sequential annotated data collections

Jonathan Chevelu, Gwénolé Lecorvé, Damien Lolive
IRISA / Université de Rennes 1
6, rue de Kerampont, Lannion, France
{jonathan.chevelu, gwenole.lecorve, damien.lolive}@irisa.fr

Abstract

The development of new methods for given speech and natural language processing tasks usually consists in annotating large corpora of data before applying machine learning techniques to train models or to extract information. Beyond scientific aspects, creating and managing such annotated data sets is a recurrent problem. While using human annotators is obviously expensive in time and money, relying on automatic annotation processes is not a simple solution neither. Typically, the high diversity of annotation tools and of data formats, as well as the lack of efficient middleware to interface them all together, make such processes very complex and painful to design. To circumvent this problem, this paper presents the toolkit ROOTS, a freshly released open source toolkit (`http://roots-toolkit.gforge.inria.fr`) for easy, fast and consistent management of heterogeneously annotated data. ROOTS is designed to efficiently handle massive complex sequential data and to allow quick and light prototyping, as this is often required for research purposes. To illustrate these properties, three sample applications are presented in the field of speech and language processing, though ROOTS can more generally be easily extended to other application domains.

Keywords: Toolkit, corpus management, speech and language processing, annotated data representation

1. Introduction

The development of new methods for given speech and natural language processing (NLP) tasks usually faces, beyond scientific aspects, various technical and practical data management problems. First, the sets of required annotated features and their desired distribution in the training data are rarely the same for two different tasks. For instance, in the field of NLP, developing a named entity extractor on texts mainly requires morphosyntactic and syntactic annotations, and relies on corpora with a high proportion of named entities. At the opposite, a word phonetizer is trained on alignments between graphemes and phonemes, eventually augmented with etymological information, whereas word and phoneme distributions are of lesser importance. Second, many dedicated systems or expert resources exist to provide annotations, e.g., part of speech (POS) taggers, named entity extractors, pronunciation dictionaries, list of proper names, etc. However, most of these tools use different file formats, time scales, or alphabets of tags. This assessment also stands for manual annotation tools since many of them coexist without sharing interoperable formats.

This heterogeneity frequently causes wastes of time when addressing elaborate tasks since a significant part of their resolution is spent in repeatedly converting data representations to others, and in aligning information coming from various tools. Possibly, some information is altered, simplified or even lost, while some other is redundantly stored over different files. Beyond the waste of time, this data management complexity can lead to errors in programs and in results, all the more frequently that addressed tasks are nowadays more and more elaborate.

The objective of this paper is to introduce a toolkit, called ROOTS, which homogenizes the management of such annotated data and thus simplifies application developments, especially for research purposes, in the frame of speech processing and NLP. ROOTS has been recently released on the Internet[1] under the terms of the GNU Lesser General Public Licence (LGPL) v3.0. This paper presents this first public version of the toolkit.

Precisely, after positioning ROOTS according to related tools in Section 2, the main features of the proposed toolkit are detailed in Section 3. Following sections then illustrate its performances on three sample tasks. An unsupervised process to create an annotated corpus of e-books is presented in Section 4, while Section 5 demonstrates how simply statistics can be extracted from this corpus using ROOTS, and how it can be interfaced with a more advanced application, namely corpus reduction over a set of constraints.

2. Positioning

In order to homogenize and to synchronize different annotation levels, (Barbot et al., 2011) have initially proposed ROOTS as a theoretical solution based on matrix algebraic relations. A Perl prototype of this solution has been validated in (Boëffard et al., 2012) by implementing an audiobook corpus construction process for text-to-speech (TTS). This process was taking advantage of ROOTS to consistently represent numerous annotation levels ranging from speech signal segments to syntactic information. Since, a new implementation in C++ has been developed, and ROOTS is now a real complete user-friendly toolkit able to efficiently process large data collections.

Other tools share this ability in the speech, NLP and multimedia communities. Among them, GATE (Cunningham et al., 2002) proposes a framework to develop NLP pipelines. However, GATE rather focuses on delivering tools for (manual or automatic) data labeling through an integrated development environment, but does not provide facilities to

[1]`http://roots-toolkit.gforge.inria.fr`.

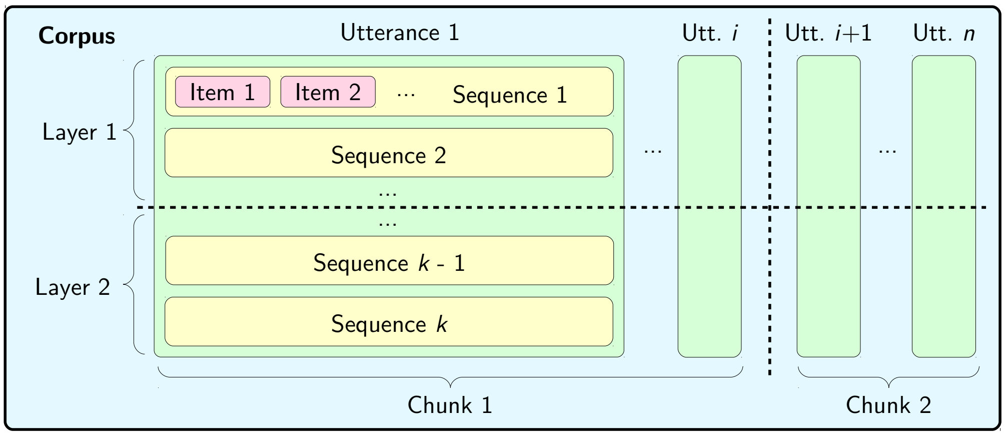

Figure 1: Hierarchical organization of data in ROOTS.

switch between bundled processing components and external tools.

More recently, the NITE XML Toolkit, or NXT, has been proposed to manage multimodal corpora (Carletta et al., 2005; Calhoun et al., 2010). NXT proposes a generic data organization model able to represent large corpora with a wide range of annotation types. Its philosophy is to consider corpora as databases from which data is accessed through a query language. On the contrary, in ROOTS, the user browses data as he sees fit. Furthermore, mechanisms to link heterogeneous annotations are not optimized in NXT whereas this is central in ROOTS.

In a more general approach, UIMA (Ferrucci and Lally, 2004; Ferrucci et al., 2006) proposes software engineering standards for unstructured data management, including annotation and processing. While this project is supported by the Apache Software Fundation, UIMA is technically too advanced for fast and light prototyping. It is rather devoted to industrial developments. Moreover, UIMA's commitment stands in imposing a common format to data processing tools whereas ROOTS' philosophy rather aims at bridging *ad hoc* formats to easily pipeline data processing tools.

In this spirit, ROOTS is close to work done within the TTS system Festival (Black et al., 2002). This system relies on a formalism called HRG, standing for Heterogenous Relation Graphs, which offers a unique representation of different information levels involved in the TTS system (Taylor et al., 2001). HRG proposes a C++ library with classes for many linguistic, phonological, acoustic, etc., structures and the relations between them. Our tool is different from HRG in the sense that the latter is part of the TTS system Festival whereas ROOTS is completely autonomous. Moreover, ROOTS comes along with a true application programming interface (API), in C++ and Perl for the moment.

Finally, handling and analyzing massive collections of annotated data—which is nowadays often referred to as the "big data" problem—also implies problems to store very large amounts, e.g., petabytes, of information (Jacobs, 2009). Solutions exist to deal with such storage issues, e.g., the system Hadoop (Shvachko et al., 2010). However, this aspect is voluntarily left aside in this paper, as well as in ROOTS in general terms.

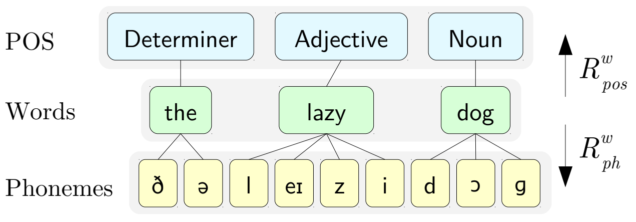

Figure 2: Example of three sequences linked by two relations.

The next section draws an overview of the toolkit ROOTS.

3. Presentation of the toolkit Roots

ROOTS stems from Rich Object Oriented Transcription System. Basically, it provides many mechanisms enabling annotated sequential data generation, management and processing. This section gives a broad description of ROOTS by presenting the proposed architecture of data, the storage capabilities, and facilities made available to users through the API and wrappers. Finally, a minimal example of code is presented.

3.1. Logical architecture

As summarized in Figure 1, data are organized hierarchically in ROOTS, starting from fine grain information in items and moving to macroscopic representations as corpora.

As a fundamental concept, data in ROOTS is modeled as sequences of items. These items can be of many types, e.g., words, graphemes, named entity classes, signal segments, etc., and can thus represent various annotation levels of the same data. Correspondences between items from different sequences are then defined as algebraic relations, leading to a graph where nodes are items and edges are derived from relations. An example of a sequence of words along with corresponding POS and phonemes is given in Figure 2. Relations R^w_{pos} and R^w_{ph}, respectively between words and POS, and between words and phonemes, are implemented as (sparse) matrices. For instance, R^w_{ph} can be

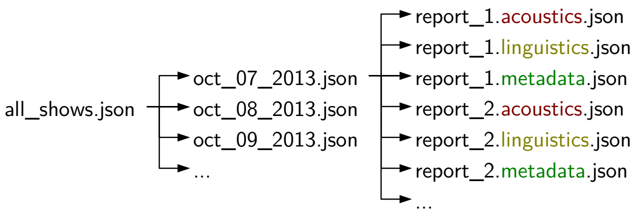

Figure 3: Example of file structure for a corpus of broadcast news shows.

written as:

$$R_{ph}^{w} = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix}, \quad (1)$$

where columns denotes phonemes and rows denotes words. Then, requesting a given target annotation for a source item simply consists in finding all paths from this item to all possible items in the target sequence. In practice, this is done by composing relations by means of matrix multiplications and transpositions. For instance, the relation R_{pos}^{ph} between phonemes and POS can be computed as $R_{pos}^{ph} = {R_{ph}^{w}}^{\intercal} \cdot R_{pos}^{w}$. All theoretical details are exposed in (Barbot et al., 2011).

Then, interrelated sequences are gathered into utterances. According to the application domain, utterances can refer to sentences, breath groups, or any relevant unit. Finally, a list of utterances forms a corpus. Hence, many structures can be considered as a corpus: paragraphs, book chapters, news reports, broadcast news shows, etc. To hierarchize structures, a corpus can be chunked into subcorpora, for instance to represent a chapter as a list of paragraphs. Regarding usage, chunks are considered as real autonomous corpora, which makes it easy to test programs on small chunks first before applying it on their full embedding corpus.

Finally, besides this "vertical" slicing, corpora can also be partitioned "horizontally" into layers. These layers are meant to gather annotations from a same field. For instance, if dealing with broadcast news shows, 3 layers of interest may be imagined to separate metadata, linguistic and acoustic information. The choice of this segmentation scheme is up to the corpus designer and should match the needs of future users.

3.2. Data storage

In ROOTS, information is stored in files, as opposed to databases, because files are easy to move, save, copy through operations that most of users know, whereas database management systems usually require more advanced skills. Furthermore, this is the most common usage in the speech and NLP research communities.

In this frame, ROOTS enables storing data in various ways. Basically, a corpus is stored as one file containing all utterances, sequences, items and relations. However, a corpus can be segmented into several files by independently storing every chunk and every layer, and thus making read/write operations lighter. An example of a segmented corpus is drawn in Figure 3. The top-level file `all_shows.json` represents a corpus of daily broadcast news shows, and enumerates the files containing the show of each day. Then, every show file points to alls layers of each news report.

On a user point of view, this file tree is hidden, and loading and saving operations are transparent, though controllable if needed. Users simply request access to utterances and sequences *via* identifiers (indices or names). Merges and splittings of utterances when loading and saving corpora are handled automatically by ROOTS. Moreover, memory is efficiently managed by postponing file loadings, and thus memory allocations, until it is really needed, and thanks to a cache over chunks. This cache prevents from reloading data from files too frequently. It also limits the maximum number of allocated utterances at runtime.

Two file formats are currently available: XML and JSON. Those formats have been chosen because they are human readable, and because they are particularly suited for the development of Web services. Other formats will be added in future versions, especially a binary format to fasten read/write operations in time-critical applications, or formats of other popular data management toolkits like those presented in Section 2, e.g., UIMA or NXT. Of course, it is still possible for users to develop wrappers to load from or dump to other formats. For instance, CSV-like formats are very easy to parse with ROOTS' API.

3.3. API and tools

Since the prototype presented in (Barbot et al., 2011) and (Boëffard et al., 2012), a new implementation of the toolkit has been written with a core library in C++ and a collection of utility scripts. The library represents about 33K lines of code and is widely documented, providing a rich API. The main useful operations accessible through this API are listed below for each type of element in ROOTS.

- **Item:** get and set the content or characteristics, e.g., gender, number for POS ; get items which are in relation with another item ; dump[2].
- **Sequence:** add, remove, get, and update one or several items ; dump.
- **Relation:** get items related to another ; link or unlink items ; dump.
- **Utterance:** add, remove, get, and update sequences ; add, remove, and update relations ; get direct or composed relations ; dump.
- **Corpus:** add, remove, get, and update an utterance ; add and remove a chunk or a layer ; dump.

As C++ is an object-oriented language, the API can be very easily extended, for instance to integrate new types of items and new file formats. For simple and fast prototyping, a Perl binding of this API is also available. This binding has been generated by the open source automatic interface generator SWIG[3]. As SWIG is a rich tool, ROOTS' C++ API could be ported to many other programming languages. For

[2]Dump refers to input/output operations in raw text, XML and JSON formats.

[3]`http://www.swig.org`.

```perl
 1  use roots;
 2
 3  my $corpus = new roots::Corpus();
 4  my $utterance = new roots::Utterance();
 5  my $word = new roots::linguistic_Word();
 6  my $lemma = new roots::linguistic_Word();
 7  my $word_sequence = new roots::WordSequence(”Word”);
 8  my $lemma_sequence = new roots::WordSequence(”Lemma”);
 9  my $word2lemma = new roots::Relation($word_sequence, $lemma_sequence);
10
11  $utterance -> add_sequence($word_sequence);
12  $utterance -> add_sequence($lemma_sequence);
13  $utterance -> add_relation($word2lemma);
14
15  while (<>) {
16    chomp;
17    if ($_ ne ””) {
18      my ($word_str, $lemma_str) = split(/\t/, $_);
19      $word -> set_label($word_str);
20      $lemma -> set_label($lemma_str);
21      my $m = $word_sequence -> add($word);
22      my $n = $lemma_sequence -> add($lemma);
23      $word2lemma -> link($m, $n);
24    }
25    elsif ($word_sequence -> count() > 0) {
26      $corpus -> add_utterance($utterance);
27      $word_sequence -> clear();
28      $lemma_sequence -> clear();
29    }
30  }
31  if ($word_sequence -> count() > 0) { $corpus -> add_utterance($utterance); }
32  $corpus -> save(”MyCorpus.json”);
33
34  for (my $i = 0; $i < $corpus -> count_utterances(); $i += 1){
35      $utterance = $corpus -> get_utterance($i);
36      print $utterance -> get_sequence(”Word”) -> to_string().”\n”;
37      print $utterance -> get_sequence(”Lemma”) -> to_string().”\n”;
38      print ”−−\n”;
39      $utterance -> destroy();
40  }
```

Algorithm 1: Sample code using ROOTS’ Perl API.

instance, a binding for Python, which is very popular in the scientific community, is scheduled.

Furthermore, the toolkit comes with various wrapping scripts developed using the Perl API. Those scripts provide facilities for simple operations, e.g., merges of corpora and layers, statistics extraction, search, textual and graphical visualization, etc. Other scripts enable command-line imports and exports with the standard speech and language processing tools HTK (Young et al., 2009), Praat (Boersma, 2002), Transcriber (Barras et al., 2001), and WaveSurfer (Sjölander and Beskow, 2000).

All sources, scripts and documentations, as well as tutorials, are freely available on `http://roots-toolkit.gforge.inria.fr`.

The API is illustrated on a first minimal example of code in the next section.

3.4. Sample code

Let us assume a text file containing sentences where words are given with their lemma. The considered input format is one word per line, words and lemmas are separated by a tabulation, and the end of a sentence is marked by an empty line or the end of file. One wants to write a program which converts the text into a ROOTS corpus and prints the content of this corpus sentence by sentence. An example of input and the corresponding expected output are shown in Table 1.

Algorithm 1 presents one possible way to write this program using ROOTS’ API. This example, and following ones in the paper, is written in Perl as it is more compact than C++. In practice, the same code could be written in C++ with only minor changes. First, the package for ROOTS is imported (line 1), and empty objects are created (l. 3-9): a corpus, an utterance, two temporary items

Input			Output
`Clouds`	`cloud`		`Clouds depart`
`depart`	`depart`		`cloud depart`
		→	`--`
`The`	`the`		`The sun shines`
`sun`	`sun`		`the sun shine`
`shines`	`shine`		`--`

Table 1: Example of input and output for Algorithm 1.

of word type`$word` and `$lemma`[4], two sequences of word type items called “Word” and “Lemma” respectively, and a relation between these two sequences. The sequences and the relation are inserted into the utterance (l. 11-13). Then, each line of the input file is parsed (l. 15-30). If the line is not empty, the pair of strings for a word and its lemma are read (l. 18), the content of temporary items are set and items inserted into the sequences[5] (l. 19-23). Based on positions m and n in the sequences, a link between inserted items is set in the relation (l. 23). If the line is empty and the sequence of words is not empty (l. 25), the utterance is added to the corpus (l. 26) before clearing the sequences in anticipation to a new sentence (l. 27-28). This latter action implicitly removes all the links in the relation. Similarly, when the end of file is reached, the utterance is added to the corpus if relevant (l. 31), and the corpus is saved into a JSON file (l. 32). Finally, the corpus is browsed (l. 34-39). For each index i, a copy of the corresponding utterance is retrieved (l. 35), its word and lemma sequences are read and printed to the standard output (l. 36-38). At the end of each iteration, the utterance copy is destroyed in order to free memory (l. 39).
Following the same principles as those presented in this section, the two next sections present examples of more complex corpus creation and processing tasks using ROOTS.

4. Application to corpus generation

ROOTS has been developed to help users creating and processing corpora with a large variety of information coming from various dedicated data analysis tools. While Section 5 deals with corpus processing, this section first demonstrates how ROOTS is effective for corpus generation on a real-life example. Precisely, an automatic text annotation process for French is introduced before applying it to a collection of e-books and briefly analyzing the resulting corpus.

4.1. Automatic annotation process

Based on a raw text in French, we designed an automatic processing chain which enriches the text with the following labels: graphemes, POS, lemmas, named entities, syntax trees, phonemes, non speech sounds and syllables. As shown in Figure 4, this is achieved by augmenting a rudimentary textual layer containing the sole sequences of raw words and corresponding graphemes with a linguistic and a phonological layer. Information of those two layers are

[4]There is no structural difference between a lemma and a word since lemmas are real words.

[5]Inserted elements are *copies* of the original element.

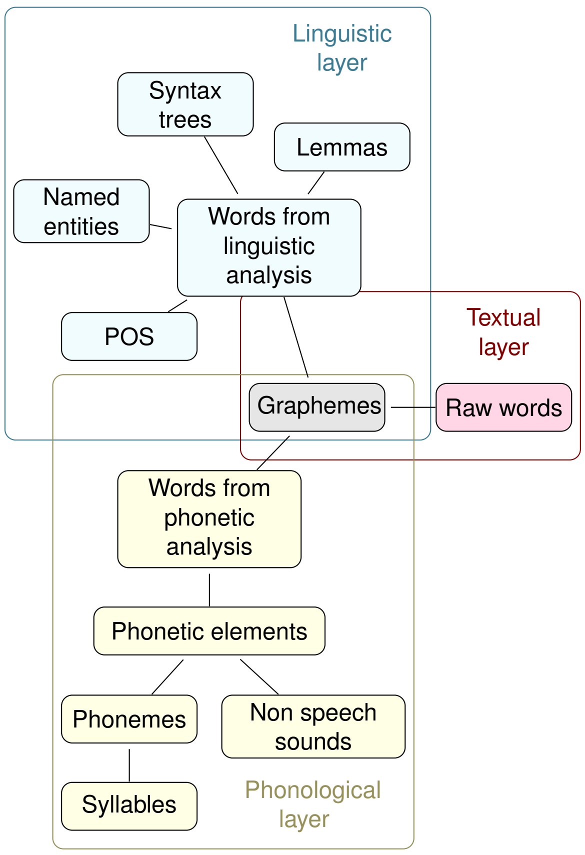

Figure 4: Schematic view of relations between sequences.

produced using two state-of-the-art dedicated tools: an NLP toolkit developed by Synapse Développement[6]; and a grapheme-to-phoneme conversion Web service from Voxygen[7]. As NLP tools usually rely on their own specific word tokenization[8], three different word sequences are considered: raw words, words from linguistic processing, and words as considered by the phonetizer. Graphemes are used as pivots to navigate between these sequences. Outputs of the annotation tools are parsed and transformed into ROOTS objects using glue scripts in Perl (about $2,000$ lines).

4.2. Creation of a corpus of e-books

The automatic annotation process described above has been applied to $1,300$ free e-books (classics) in French collected from `http://www.ebooksgratuits.com`. Precisely, after converting the e-books[9] and cleaning the resulting raw texts, a first ROOTS corpus made of the sole raw word and grapheme sequences has been created. Then, the annotation process has been executed in parallel on a computer powered by a 64 core 2 GHz Intel Xeon CPU and 120 GB RAM.
The annotation process resulted in a ROOTS corpus whose statistics about items are presented in Table 2. This corpus

[6]`http://www.synapse-developpement.fr`.

[7]`http://www.voxygen.fr`.

[8]For instance, a tool may consider acronyms as one word while another may explode them into sequences of capitals.

[9]Conversion has been achieved using Calibre by Kovid Goyal (`http://calibre-ebook.com`).

Type	Number (millions)
Words (raw)	94
Graphemes	534
Words (linguistic)	111
POS	111
Lemmas	111
Named Entities	4
Syntax trees	6
Words (phonological)	112
Syllables	134
Non speech sounds	21
Phonemes	309

Table 2: Numbers of automatically generated items in the corpus of e-books.

is spread over 35, 388 JSON files for a total disk usage of about 220 GB. Processing time was about 15 hours. Individually, this is about 27 files, 173 MB, and 45 min per e-book. Let us note that most of the execution time is due to calls to the external tools, especially the remote phonetizer. Furthermore, files could be highly compressed since JSON is a verbose format.
An example of a generated utterance is drawn in Figure 5. This visualization has been automatically computed by a powerful alignment algorithm developed for ROOTS. Drawn sequences have been manually selected to maintain readability. Especially, intermediate word sequences from linguistic and phonological tagging are not displayed. Two items are one above the other if and only if there exists a relation linking them.

5. Application to corpus processing

This section demonstrates how the toolkit is particularly convenient for annotated data processing through two sample tasks: statistics extraction, and corpus reduction.

5.1. Statistics extraction

A first considered data processing task is to compute frequencies over syllables in a textual corpus. Precisely, one wishes to highlight how frequently each syllable is found at the end of a word in a corpus. To make the objective a bit more complex, and to show a more advanced usage of ROOTS, it is further constrained to discard mono-syllabic words from frequency computation.
This target task is realized using the short script presented in Algorithm 2. This code is split into two parts: the function `count_last_syllables` (lines 3-18); and the "main" code (l. 20-31). The main code declares a map `%frequency` to store syllable counts (l. 20). Then, a ROOTS corpus is created and populated by loading a JSON file (l. 22). Similarly to Algorithm 1, the corpus is browsed (l. 23-27). Precisely, last syllables are count by iteratively updating the map `%frequency` for each utterance (l. 25). Finally, all encountered syllables are printed with their respective total frequency (l. 29-31). Regarding the function `count_last_syllables`, arguments, i.e., the reference to the map of counts and a reference to the

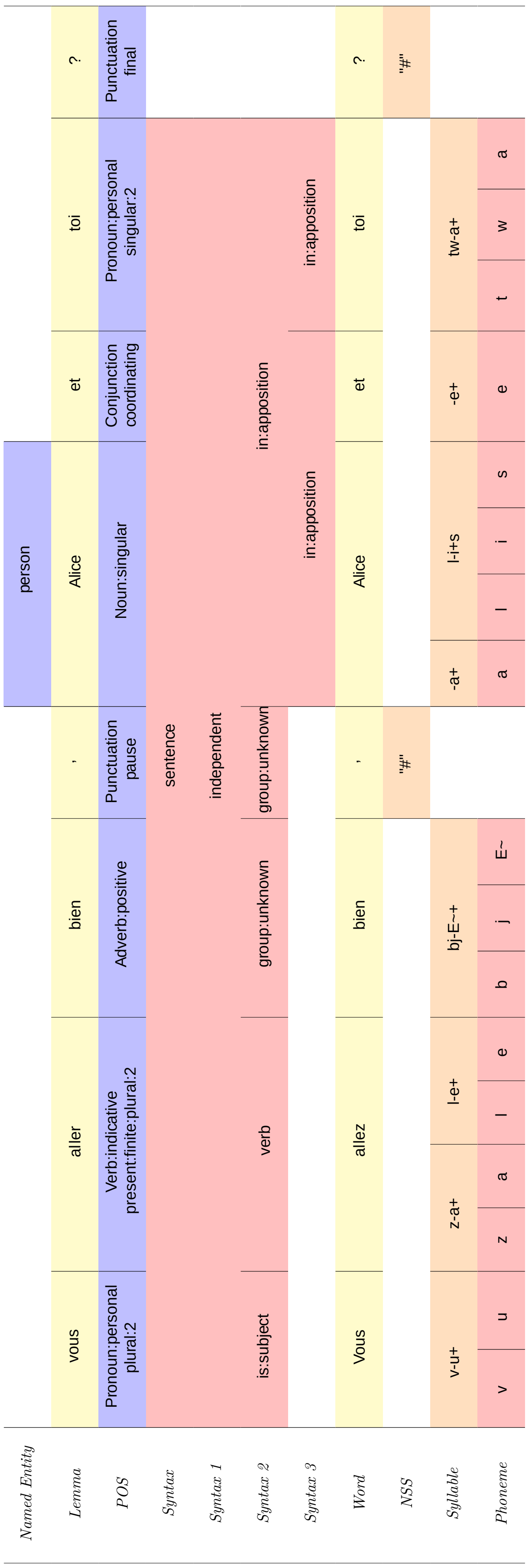

Figure 5: Automatically generated display of an utterance with ROOTS.

```
1  use roots;
2
3  sub count_last_syllables {
4      my $p_frequency = shift;
5      my $utterance = shift;
6
7      my $word_seq = $utterance -> get_sequence(”Word”);
8
9      foreach my $word (@{$word_seq -> get_all_items()}) {
10         my @syllables = @{$word -> get_related_items(”Syllable”)};
11         if($#syllables >  0) {
12             my $last_syllable = pop(@syllables) -> to_string();
13             if(!exists($$p_frequency{$last_syllable}))
14                    { $$p_frequency{$last_syllable} = 1; }
15             else   { $$p_frequency{$last_syllable} += 1; }
16         }
17     }
18 }
19
20 my %frequency;
21
22 my $corpus = new roots::Corpus(”MyRootsFile.json”);
23 for (my $i = 0; $i < $corpus -> count_utterances(); $i++) {
24     my $utterance = $corpus -> get_utterance($i);
25     count_last_syllables(\%frequency, $utterance);
26     $utterance -> destroy();
27 }
28
29 while (my ($syl,$freq) = each(%frequency)) {
30     print ”$syl\t$freq\n”;
31 }
```

Algorithm 2: Perl script for syllable frequency computation based on ROOTS.

Syllable	Frequency
/mɑ̃/	1,098,968
/te/	1,041,858
/tɛ/	885,349
/vɛ/	822,038
/sjɔ̃/	649,986
/se/	495,601
/ne/	430,312
/si/	418,720
/ʁɛ/	417,144
/ve/	338,582

Table 3: List of the 10 syllables which most frequently end words in the corpus of French e-books.

current utterance, are read (l. 4-5). Then, the word sequence of this utterance is retrieved (line 7) and all the words are browsed (l. 9). For each word, the list of related syllables is computed (l. 10). After checking if the word is not monosyllabic (l. 11), the last syllable is accessed and its string value is got (l. 12). Finally, the frequency of the current syllable is either initialized or updated (l. 13-15).

This script has been applied on the ROOTS corpus of e-books from Section 4 using the same parallelized computational setup as previously. The overall execution has lasted about 20 min, with a duration of 1 min and a 300 MB memory usage on average per e-book. The resulting top 10 most frequent syllables is given in Table 3. A quick analysis on these results show that most words end with /mɑ̃/, which is a typical ending for adverbs in French, or with /e/ or /ɛ/, which are verb inflection marks in French for past participles and the imperfect tense, respectively. Hence, the results are consistent with the narrative style of the corpus.

5.2. Corpus reduction

Besides simplicity, delivering efficient processing skills is a key aspect. This section now shows that ROOTS is perfectly suited for such a demand by addressing a corpus reduction problem as a second sample task. Corpus reduction consists in extracting from a huge corpus a minimal sized subset which satisfies a set of constraints over features of the corpus. Corpus reduction is a prerequisite for some tasks where the quality of results mainly depends on the diversity of the reference corpus content but where considering a very large corpus is prohibitive.

(Chevelu et al., 2008) proposed an efficient greedy algorithm to reduce corpora. Since ROOTS structures heterogeneous annotations, a simple and practical usage of this corpus reduction method consists in feeding the algorithm with information derived from a ROOTS corpus. In practice, this has been done in our experiments by generating a sparse matrix of features using ROOTS' C++ API, before processing this matrix with the corpus reduction algorithm.

To illustrate the work flow, corpus reduction has been applied to the annotated corpus of e-books of Section 4. Two constraints for the target reduced corpus were considered:

- every phoneme paired with its corresponding POS should appear at least twice;
- and every trigram of phonemes, independently of their related POS, should appear at least once.

The corpus covers $10,246$ unique phoneme-POS pairs and $31,612$ unique phoneme trigrams. Using the same experimental setup as previously, the process compiled information from e-books during about 3 h 30 min (ROOTS) before computing a solution in only 2 min 40 s (specialized algorithm). As a result, while the full corpus is made of 6.2 million sentences and sizes 309 million phoneme instances, the returned solution contains only $17,846$ sentences ($0.3\,\%$ of the full corpus) and $547,111$ phoneme instances ($0.2\,\%$ of the full corpus). A major interest of ROOTS is that this work flow can be applied to any corpus without changing anything except the names of sequences under study and requested minimal frequencies of their items.

6. Conclusion and further developments

In this paper, we have presented a powerful toolkit called ROOTS which enables easy, fast and consistent generation, management and processing of large annotated sequential data collections. After presenting the toolkit itself, these abilities have been demonstrated through three application examples. This toolkit is efficient, fully operational, open source, widely documented, and easily extensible.

Further developments are already planed for future releases. First, a graphical user interface to allow manual editing and annotation correction will be added. New file formats should be integrated to optimize read/write performances for time-critical applications, or to become compliant with standards from various communities. Likewise, more wrappers for existing automatic taggers and data processing tools will be integrated. More fundamentally, sequences should be extended to lattices in order to enable information uncertainty modeling when needed. Finally, corpora generated using ROOTS on free data sets should be made available. Especially, the corpus of French e-books should be published in the near future.

7. Acknowledgements

This work has been partially funded by the *Agence Nationale de la Recherche* (France) as part of the project ANR-CONTINT 2011 Phorevox.

8. References

Barbot, N., Barreaud, V., Boëffard, O., Charonnat, L., Delhay, A., Le Maguer, S., and Lolive, D. (2011). Towards a versatile multi-layered description of speech corpora using algebraic relations. In *Proceedings of Interspeech*, pages 1501–1504.

Barras, C., Geoffrois, E., Wu, Z., and Liberman, M. (2001). Transcriber: development and use of a tool for assisting speech corpora production. *Speech Communication*, 33(1-2):5–22.

Black, A. W., Taylor, P., Caley, R., and Clark, R. (2002). The Festival speech synthesis system. Technical report, University of Edinburgh.

Boersma, P. (2002). Praat, a system for doing phonetics by computer. *Glot international*, 5(9/10):341–345.

Boëffard, O., Charonnat, L., Le Maguer, S., Lolive, D., and Vidal, G. (2012). Towards fully automatic annotation of audiobooks for TTS. In *Proceedings of LREC*, pages 975–980.

Calhoun, S., Carletta, J., Brenier, J. M., Mayo, N., Jurafsky, D., Steedman, M., and Beaver, D. (2010). The nxt-format switchboard corpus: a rich resource for investigating the syntax, semantics, pragmatics and prosody of dialogue. *Language Resources and Evaluation*, 44(4):387–419.

Carletta, J., Evert, S., Heid, U., and Kilgour, J. (2005). The NITE XML toolkit: Data model and query language. *Language Resources and Evaluation*, 39(4):313–334.

Chevelu, J., Barbot, N., Boëffard, O., and Delhay, A. (2008). Comparing set-covering strategies for optimal corpus design. In *Proceedings of LREC*, pages 2951–2956.

Cunningham, H., Maynard, D., Bontcheva, K., and Tablan, V. (2002). GATE: an architecture for development of robust HLT applications. Proceedings of the Annual Meeting of the ACL, pages 168–175.

Ferrucci, D. and Lally, A. (2004). UIMA: an architectural approach to unstructured information processing in the corporate research environment. *Natural Language Engineering*, 10(3-4):327–348.

Ferrucci, D., Lally, A., Gruhl, D., Epstein, E., Schor, M., Murdock, J. W., Frenkiel, A., Brown, E. W., Hampp, T., Doganata, Y., et al. (2006). Towards an interoperability standard for text and multi-modal analytics.

Jacobs, A. (2009). The pathologies of big data. *Communications of the ACM*, 52(8):36–44.

Shvachko, K., Kuang, H., Radia, S., and Chansler, R. (2010). The Hadoop distributed file system. In *Proceedings of the IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, pages 1–10.

Sjölander, K. and Beskow, J. (2000). Wavesurfer - an open source speech tool. In *Proceedings of Interspeech*, pages 464–467.

Taylor, P., Black, A. W., and Caley, R. (2001). Heterogeneous relation graphs as a formalism for representing linguistic information. *Speech Communication*, 33(1-2):153–174.

Young, S. J., Evermann, G., Gales, M. J. F., Hain, T., Kershaw, D., Moore, G., Odell, J., Ollason, D., Povey, D., Valtchev, V., and Woodland, P. C. (2009). *The HTK Book, version 3.4*. Cambridge University Engineering Department.