ROOTS : un outil pour manipuler facilement, efficacement et avec cohérence des corpus annotés de séquences

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

RÉSUMÉ

De nombreux outils de traitement automatique de la parole et du langage naturel permettent aujourd'hui d'annoter des documents écrits et oraux. Cependant, cette richesse logicielle conduit à une grande diversité de formats de fichiers et de types d'information. Du fait de cette hétérogénéité, le développement de processus de traitements complexes nécessite souvent de convertir et d'aligner, parfois de manière répétée, de nombreuses informations. Pour pallier ce problème, cet article présente ROOTS, un outil libre dédié à la manipulation homogène de données séquentielles annotées. Écrit en C++ et disposant d'une interface de programmation (API) dans plusieurs langages, ROOTS est rapide et facile à prendre en main. Ces propriétés sont étayées par deux exemples applicatifs.

ABSTRACT

Roots: a tool for easy, efficient and consistent manipulation of annotated sequence corpora

Numbers of speech and natural language processing tools are nowadays available to annotate spoken and textual documents. However, this diversity of software leads to numerous different file formats and information types which are inherently not interoperable. Hence, developing complex data processing processes often implies to convert and align, sometimes repeatedly, these pieces of information. To overcome this problem, this paper presents ROOTS, a tool for homogeneous management and processing of annotated sequential data. As ROOTS is written in C++ and provides an efficient application programming interface (API) in several programming languages, it is fast and user-friendly. These porperties are demonstrated on two sample tasks.

MOTS-CLÉS : Outil logiciel, gestion de corpus, représentation de données annotées.

KEYWORDS: Software tool, corpus management, annotated data representation.

1 Introduction

Mettre au point de nouvelles méthodes de traitement automatique de la parole ou du langage naturel passe fréquemment, au-delà de l'aspect scientifique, par la résolution de divers problèmes techniques liés à la gestion des données. Tout d'abord, parce que le type des données, leur distribution et la nature des descripteurs nécessaires à deux tâches différentes sont rarement les mêmes. Par exemple, dans le cadre du traitement automatique des langues (TAL), le développement d'un extracteur d'entités nommées requiert généralement un corpus de textes annotés en informations morphosyntaxiques et syntaxiques et dont la proportion d'entités nommées est élevée. Alors que l'apprentissage d'un phonétiseur requiert avant tout un corpus aligné de graphèmes et de phonèmes, éventuellement agrémentés d'informations étymologiques sur les mots représentés,

les distributions *a priori* de ces différents éléments restant, elles, plus secondaires.

Ensuite, il existe beaucoup de systèmes spécialisés ou de ressources expertes capables de fournir des annotations (des étiqueteurs en partie du discours (POS), des dictionnaires de prononciation, des listes de noms propres...) mais la plupart de ces outils utilisent des formats spécifiques, travaillent sur des échelles temporelles différentes ou s'appuient sur des jeux d'étiquettes qui leur sont propres. Ce constat est également valable pour les outils d'annotation manuelle puisque de multiples logiciels co-existent sans véritablement partager de format en commun.

Cette hétérogénéité induit souvent un travail long et rébarbatif de conversion et d'alignement des informations issues de différents outils. Éventuellement, certaines informations sont simplifiées ou perdues durant ces étapes, d'autres sont redondantes car stockées de différentes manières dans divers fichiers. Au delà de la perte de temps, cette complexité peut conduire à des erreurs dans les programmes et dans les résultats, ceci d'autant plus fréquemment que les tâches à résoudre sont aujourd'hui de plus en plus élaborées.

L'objectif de cet article est de présenter un outil, appelé ROOTS, qui homogénéise la gestion de ces données annotées et permet de simplifier le développement d'applications dans le cadre du traitement de la parole et du langage naturel, notamment à des fins de recherche. ROOTS est le fruit de plusieurs années de développement en interne et, comme nous pensons qu'il devrait profiter à la communauté, il est depuis peu librement téléchargeable sur Internet (`http://roots-toolkit.gforge.inria.fr`). Cet article présente cette première version publique de l'outil.

Précisément, la section 2 détaille tout d'abord l'architecture de ROOTS. Notre outil est ensuite comparé à d'autres outils existants à la section 3, puis son fonctionnement pratique et ses performances sont illustrés à la section 4 à travers deux tâches : l'annotation non supervisée d'un large corpus de livres numériques et l'extraction de statistiques à partir d'annotations.

2 Présentation de l'outil Roots

Afin d'homogénéiser et de synchroniser les différents niveaux d'annotation d'un corpus, (Barbot *et al.*, 2011) ont initialement proposé ROOTS (pour *Rich Object Oriented Transcription System*) comme une solution fondée sur des relations algébriques matricielles. Un prototype en Perl implémentant cette solution a été validé dans (Boëffard *et al.*, 2012) par la mise au point d'un processus de construction automatique d'un corpus de livres audios à des fins de synthèse vocale. Ce processus tirait profit de ROOTS pour représenter de manière homogène de nombreux niveaux d'annotations.

Depuis, une nouvelle implémentation en C++ a été développée, celle-ci faisant de ROOTS un outil complet, facile à prendre en main et capable de traiter de larges collections de données séquentielles annotées, en particulier issues de documents oraux ou écrits. Cette section en donne une vue d'ensemble en présentant son architecture logique des données, les fonctionnalités mises à disposition des utilisateurs et ses aptitudes en terme de stockage.

2.1 Architecture logique des données

Comme le résume la figure 1, les données sont structurées hiérarchiquement dans ROOTS. Fondamentalement, les données sont modélisées comme des séquences d'items. Ces items sont

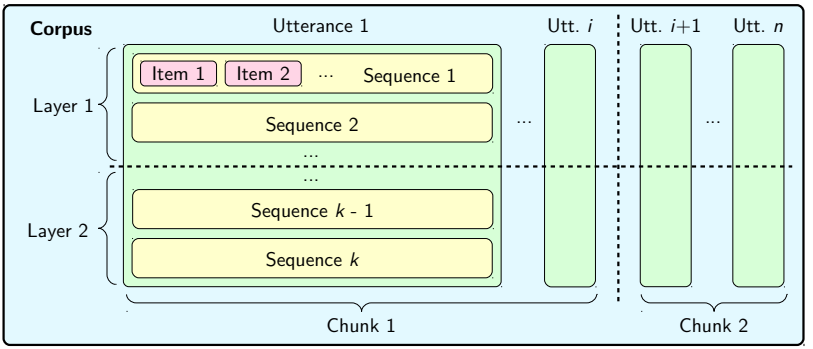

FIGURE 1 – Organisation logique des données dans ROOTS.

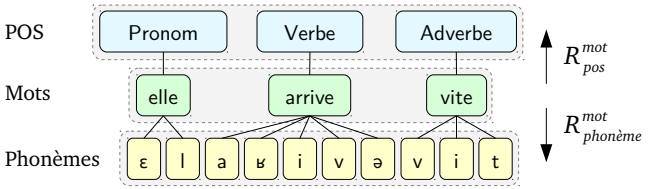

FIGURE 2 – Exemple de 3 séquences liées par 2 relations, R^{mot}_{pos} et $R^{mot}_{phonème}$.

typés, par exemple, en mots, en graphèmes, en classes d'entités nommées…, et peuvent donc représenter différents niveaux d'annotations des mêmes données comme autant de séquences de types différents. Les correspondances entre items des différentes séquences sont définies comme des relations algébriques représentées sous forme de matrices. L'ensemble de ces relations permet de produire un graphe non orienté dont les nœuds sont des items et les arêtes sont dérivées des relations. La composition des relations algébriques qui forment le plus court chemin entre deux séquences permet de trouver des correspondances entre items même en l'absence de relation directe. L'exemple d'une séquence de mots annotée en POS et en phonèmes est donnée en figure 2. Les lecteurs intéressés trouveront tous les détails théoriques dans (Barbot *et al.*, 2011).

Les séquences d'un même contenu sont regroupées en énoncés, ou *utterances* dans la terminologie de ROOTS. Ils peuvent correspondre à n'importe quelle unité pertinente pour un domaine donné (un mot isolé, une phrase, un groupe de souffle…). Rassemblés en une liste, les énoncés forment un corpus. Beaucoup de structures peuvent être modélisées ainsi : des paragraphes, des chapitres, des livres, des reportages télévisés, des brèves d'actualités… Pour hiérarchiser ces structures, un corpus peut être divisé en sous-corpus, par exemple pour représenter un chapitre comme une liste de paragraphes. Ces divisions « verticales » (*cf.* figure 1) d'un corpus sont appelées *chunks* dans ROOTS. Les corpus peuvent aussi être découpés « horizontalement » en couches d'informations, ou *layers*. Ces couches permettent, au besoin, d'isoler certaines annotations relevant d'un même niveau d'abstraction. Par exemple, dans le cadre d'un corpus de journaux télévisés, il peut être intéressant de découper le corpus en 3 couches : l'une pour les informations acoustiques et phonétiques (fréquences fondamentales, spectres, allophones…) ; une autre regroupant les informations d'ordre linguistique (transcription orthographique, étiquetages en POS, arbres syntaxiques…) ; une dernière pour des métadonnées (heure de diffusion, lieu d'un reportage…).

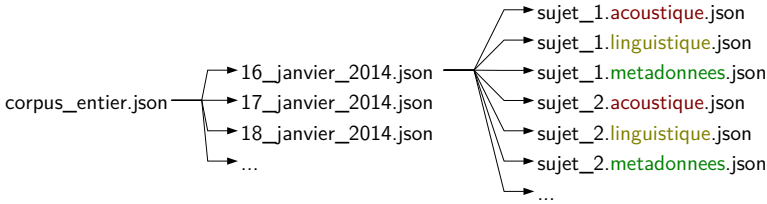

FIGURE 3 – Exemple de structuration des fichiers pour un corpus de journaux télévisés.

2.2 Interface de programmation et outils

ROOTS s'appuie sur une bibliothèque écrite en C++ et rassemble un ensemble d'utilitaires. La bibliothèque représente environ 33 000 lignes de code. Elle s'accompagne d'une API riche et largement documentée qui propose en particulier les fonctionnalités suivantes pour les différents éléments manipulés :

- **Items :** obtenir et définir les propriétés d'un item, comme le genre et le nombre pour un POS ; obtenir les items liés dans d'autres séquences.
- **Relation :** tester, obtenir les items liés ; lier ou délier des items ;
- **Séquence :** ajout, suppression, extraction et mise à jour d'un ou plusieurs items.
- ***Utterance* :** ajout, suppression, extraction et mise à jour des séquences et relations internes ; obtenir des relations directs ou composées entre séquences ; combiner, scinder et sauvegarder.
- **Corpus :** ajout, suppression, extraction et mise à jour d'un énoncé ; gestion de la structure logique et du stokage (voir la section 2.3) ; sauvegarde et chargement.

Pour permettre le prototypage simple et rapide de programmes, cette API est aussi disponible pour le langage Perl grâce au générateur automatique d'interface SWIG. Ainsi, ROOTS peut être porté à moindres frais vers beaucoup d'autres langages de programmation. En particulier, un portage vers le langage Python est prévu.

Des scripts utilitaires fondés sur l'API Perl sont disponibles pour les opérations les plus utiles (fusions et découpages de corpus, la recherche dans un corpus, la visualisation textuelle comme graphique des données...). D'autres scripts permettent également les imports et exports depuis et vers les outils de traitement de la parole HTK (Young *et al.*, 2009), Praat (Boersma, 2002), WaveSurfer (Sjölander et Beskow, 2000) et Transcriber (Barras *et al.*, 2001). ROOTS permet d'ajouter simplement d'autres formats, par exemple ceux de type CSV.

Toutes les sources, la documentation et des tutoriaux sont disponibles[1] en ligne sur `http://roots-toolkit.gforge.inria.fr`.

2.3 Stockage des données

Les informations sont sauvegardées sous forme de fichiers dans ROOTS, et non dans des bases de données. Ainsi les opérations rudimentaires sur des fichiers (déplacer, supprimer, copier...) sont connues de tous les utilisateurs et ne requièrent pas de compétence particulière. Par défaut, un corpus est simplement stocké comme un unique fichier contenant tous les énoncés, séquences, items et relations. Si le corpus est structuré en sous-corpus ou en couches, un fichier par subdivision est créé. Cette stratégie permet d'alléger les opérations de lecture et d'écriture en ne ciblant que certaines portions utiles du corpus entier. De plus, chaque sous fichier est lui-même

1. Sous licence *GNU Lesser General Public Licence* (LGPL) v3.0.

vu comme un corpus ROOTS et peut donc être utilisé indépendamment. Cela permet, par exemple, de tester et régler un programme sur une fraction seulement d'un vaste corpus avant de le lancer sur son intégralité. En reprenant l'exemple d'un corpus de journaux télévisés, nous pouvons imaginer une répartition des fichiers telle que présentée par la figure 3. Le fichier en point d'entrée (à gauche) représente le corpus entier. Celui-ci énumère simplement la liste des fichiers contenant les annotations du journal de chaque jour. De même, chacun de ces fichiers pointe vers les fichiers contenant ses différentes couches d'informations, sujet par sujet. En pratique, cette structuration des fichiers est transparente pour les utilisateurs qui souhaitent consulter le corpus. Enfin, l'utilisation des ressources est optimisée par un cache sur les *chunks* retardant au maximum le chargement des fichiers, et donc les allocations de mémoire correspondantes.

Deux formats de fichiers sont actuellement disponibles : XML et JSON. Ils sont lisibles sans outil particulier et sont appropriés pour la mise en place de services web. D'autres formats vont être ajoutés dans les versions futures, notamment un format binaire souhaitable pour développer des applications très rapides.

3 Comparaison avec les outils existants

D'autres outils permettent le traitement de données annotées. Parmi eux, le système GATE (Cunningham, 2002) propose d'interconnecter des flux d'annotations pour le développement de briques de TAL. Malgré cela, GATE s'appuie sur un environnement de développement intégré et ne permet donc pas facilement d'intégrer des outils tiers.

Plus récemment, la boîte à outils NXT a été proposée pour la gestion de corpus multimodaux (Carletta *et al.*, 2005; Calhoun *et al.*, 2010). Celle-ci s'intéresse à de grands corpus multiplement annotés en proposant un modèle générique d'organisation des données. Ce modèle s'apparente à celui d'une base de données, avec un accès aux données par un langage de requête. À l'inverse, ROOTS laisse à l'utilisateur le soin de parcourir les données comme bon lui semble. Par ailleurs, les mécanismes permettant de mettre en relation les annotations de différents ordres ne sont pas optimisés dans NXT alors que cet aspect est au cœur de notre outil.

Dans une approche plus générale, UIMA (Ferrucci et Lally, 2004; Ferrucci *et al.*, 2006) propose des normes de génie logiciel pour la gestion de données non structurées, y compris leur annotation et leur traitement. Cependant, UIMA est, à notre sens, trop élaboré techniquement pour permettre de prototyper rapidement et facilement des idées de solutions, comme l'exige souvent un travail de recherche. Il est donc plutôt dédié à des développements industriels. En outre, UIMA prône un format commun pour les outils de traitement de données alors que la philosophie de ROOTS est de permettre la construction à moindre coût de ponts entre les formats spécifiques de ces outils.

Dans cet esprit, ROOTS se rapproche des travaux réalisés dans le cadre du système de synthèse de la parole Festival (Black *et al.*, 2002). Ce système s'appuie sur le formalisme HRG, pour *Heterogenous Relation Graphs*, qui vise à offrir un mode de représentation unique des différents niveaux d'informations intervenant dans un système de synthèse (Taylor *et al.*, 2001). HRG propose ainsi une bibliothèque C++ regroupant des classes pour de multiples structures linguistiques, phonologiques, acoustiques. . ., et leur mise en relation. Notre outil s'en distingue néanmoins car HRG est partie intégrante du système Festival alors que ROOTS est complètement autonome. Par ailleurs, ROOTS dispose d'une véritable interface de programmation (API), en C++ et en Perl.

Enfin, notons à toute fin utile que les problèmes engendrés par la manipulation de très grands

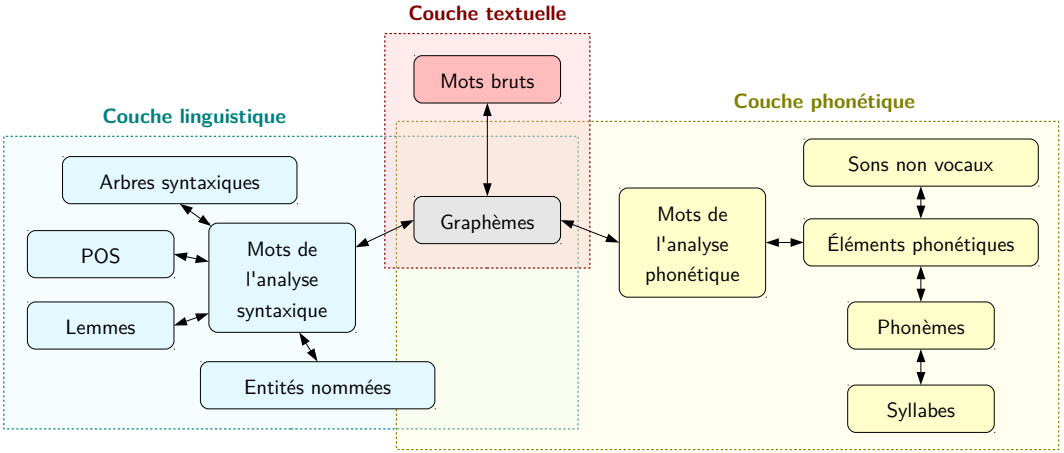

FIGURE 4 – Vue schématique des relations entre séquences.

volumes de données (Jacobs, 2009), de l'ordre du pétaoctet, ne sont volontairememnt pas pris en compte, ni dans cet article ni dans ROOTS plus généralement.

4 Exemples d'applications

ROOTS a été développé pour créer et traiter facilement et efficacement de grands corpus recensant des informations nombreuses, hétérogènes et issues d'outils divers. Cette section démontre ces capacités sur un exemple de construction d'un tel corpus, puis sur une tâche de calcul de statistiques à partir d'un corpus créé *via* ROOTS. Les expériences ont été réalisées sur une machine équipée de 64 cœurs Intel Xeon cadencés à 2 GHz et de 120 Go de RAM.

4.1 Construction d'un corpus annoté de livres numériques

Nous avons conçu un processus non supervisé qui, à partir d'un texte brut en français, produit et relie les informations suivantes : graphèmes, POS, lemmes, entités nommées, arbres syntaxiques, phonèmes, sons non vocaux et syllabes. En utilisant les concepts de ROOTS et comme illustré par la figure 4, ce processus consiste à compléter une couche d'informations textuelles rudimentaires, composées des seules séquences de mots bruts et des graphèmes correspondants, avec une couche linguistique et une couche phonologique. Ces deux dernières couches sont produites respectivement à partir des résultats de deux outils spécialisés : une suite logicielle pour le TAL de la société Synapse Développement et un service web de phonétisation, de la société Voxygen. Comme souvent, chaque outil s'appuie sur un découpage en mots qui lui est propre [2]. Aussi, trois séquences différentes de mots sont stockées (mots bruts, issus de l'analyse linguistique ou issus du phonétiseur) et mises en relation en utilisant les graphèmes comme pivot.

Les sorties des outils d'annotations sont converties en objets ROOTS à l'aide de scripts Perl écrits pour l'occasion (environ 2 000 lignes de code).

Ce processus d'annotation a été appliqué sur 1 300 livres numériques, libres et en français,

2. Par exemple, l'un peut conserver les acronymes en un mot alors que l'autre les explose en séquences d'initiales.

Type	Nombre (millions)
Mots (bruts)	94
Graphèmes	534
Mots (phonologiques)	112
Syllabes	134
Sons non vocaux	21
Phonèmes	309

Type	Nombre (millions)
Mots (linguistiques)	111
POS	111
Lemmes	111
Entités nommées	4
Arbres syntaxiques	6

TABLE 1 – Nombre d'items automatiquement générés dans le corpus de livres numériques.

Syllabe	Nombre	Syllabe	Nombre	Syllabe	Nombre
/mɑ̃/	1 098 968	/sjɔ̃/	649 986	/rɛ/	417 144
/te/	1 041 858	/se/	495 601	/ve/	338 582
/tɛ/	885 349	/ne/	430 312	/re/	303 055
/vɛ/	822 038	/si/	418 720	/tɑ̃/	288 008

TABLE 2 – Liste des 12 syllabes les plus fréquentes en fin de mot dans le corpus de livres.

collectés depuis `http://www.ebooksgratuits.com`. Le texte brut de ces livres a ainsi été transformé en un corpus ROOTS dont les statistiques sont présentées dans la table 1. Ce corpus est réparti en 35 388 fichiers JSON qui occupent un total de 220 Go (environ 27 fichiers et 173 Mo par livre). Cette taille pourrait être largement réduite par compression puisque JSON est un format très verbeux. Le temps d'exécution du processus d'annotation a été de quinze heures et est principalement dû aux appels aux outils externes comme le service en ligne de phonétisation.

4.2 Extraction de statistiques

Cette section démontre maintenant la simplicité d'utilisation de l'API de ROOTS. Pour cela, et afin d'illuster au mieux les fonctionnalités élémentaires, nous supposons vouloir répertorier et compter les syllabes en fin de mot dans un corpus, excepté pour les mots mono-syllabiques.

Cette tâche est réalisée par un court script dont le code complet est donné par l'algorithme 1. Il est composé de deux parties : la fonction `compter_dernieres_syllabes` (lignes 3-16) et le code principal (l. 18-27). Le code principal déclare une table associative pour stocker le nombre d'occurrences des syllabes (l. 18), puis charge un corpus ROOTS à partir d'un fichier JSON (l. 19). Le corpus est ensuite parcouru, phrase par phrase, de la manière suivante : une copie de chaque phrase est récupérée (l. 21), la table est mise à jour par la fonction `compter_dernieres_syllabes` (l. 22), puis la copie est détruite pour libérer la mémoire (l. 23). Enfin, toutes les syllabes rencontrées et leur nombre d'occurrences respectif sont affichés (l. 25-27). La fonction `compter_dernieres_syllabes` prend en entrée la référence vers la table des comptes et une référence vers un objet *utterance*, contenant une phrase (l. 4-5). La séquence de mots de cette phrase est récupérée (l. 6) et parcourue (l. 7). Pour chaque mot, la liste des syllabes correspondantes est calculée (l. 8). Après avoir vérifié que le mot ne contient pas qu'une seule syllabe (l. 9), la dernière syllabe est lue et convertie en une chaîne de caractères (l. 10), puis son nombre d'occurrences est soit initialisé soit incrémenté (l. 11-13).

Ce script a été appliqué en parallèle sur chaque livre numérique du corpus décrit à la section 4.1.

```perl
use roots;

sub compter_dernieres_syllabes {
    my $p_nb_occurrences = shift;
    my $phrase = shift;
    my $seq_mots = $utterance -> get_sequence("Mots");
    foreach my $mot (@{$seq_mots -> get_all_items()}) {
        my @syllabes = @{$mot -> get_related_items("Syllabes")};
        if($#syllabes > 0) {
            my $derniere_syllabe = pop(@syllabes) -> to_string();
            if(!exists($$p_nb_occurrences{$derniere_syllabe}))
                { $$p_nb_occurrences{$derniere_syllabe} = 1; }
            else { $$p_nb_occurrences{$derniere_syllabe} += 1; }
        }
    }
}

my %nb_occurrences;
my $corpus = new roots::Corpus("MonFichierRoots.json");
for (my $i = 0; $i < $corpus -> count_utterances(); $i++) {
    my $phrase = $corpus -> get_utterance($i);
    compter_dernieres_syllabes(\%nb_occurrences, $phrase);
    $phrase -> destroy();
}
while (my ($syllabe, $nombre) = each(%nb_occurrences)) {
    print $syllabe."\t".$nombre."\n";
}
```

Algorithme 1 – Script Perl utilisant l'API de ROOTS pour compter les syllabes.

L'ensemble a duré 20 min, soit environ 1 min par livre. En complément, l'utilisation moyenne de la mémoire centrale a été d'environ 300 Mo par livre. Le résultat des 12 syllabes les plus fréquentes est donné par la table 2. Il ressort de son analyse rapide que la plupart des mots terminent par /mɑ̃/, suffixe typique des adverbes, ou par /e/, /ɛ/ ou /ɑ̃/, marques des flexions verbales du participe passé, de l'imparfait et du gérondif. Ainsi, ces résultats sont cohérents avec le style narratif du corpus.

5 Conclusion et développements futurs

Dans cet article, nous avons présenté l'outil ROOTS, qui permet de générer, gérer et traiter facilement, efficacement et avec cohérence de grands corpus de séquences annotées, principalement à partir de données orales ou écrites. Après avoir présenté l'outil et l'avoir comparé à l'existant, ses performances ont été démontrées sur deux exemples applicatifs. ROOTS est entièrement opérationnel, libre et facile à étendre. Pour ces multiples raisons, nous pensons qu'il devrait être profitable à la communauté.

Dans les versions futures, une interface graphique permettant une édition manuelle d'annotations et de nouvelles fonctionnalités d'import/export seront ajoutées. Il est également envisagé d'étendre la notion de séquence à celle de treillis, utile pour modéliser l'incertitude de certaines informations. Enfin, les corpus générés avec ROOTS à partir de données libres seront prochainement mis à disposition, notamment le corpus de livres présenté dans nos expériences.

Références

Barbot, N., Barreaud, V., Boëffard, O., Charonnat, L., Delhay, A., Le Maguer, S. et 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. et 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. et Clark, R. (2002). The Festival speech synthesis system. Rapport technique, 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. et 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. et 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. et Kilgour, J. (2005). The NITE XML toolkit : Data model and query language. *Language Resources and Evaluation*, 39(4):313–334.

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

Ferrucci, D. et 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. *IBM Research Report*.

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

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

Taylor, P., Black, A. W. et 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. et Woodland, P. C. (2009). *The HTK Book, version 3.4*. Cambridge University Engineering Department.