Framework modulaire de développement de ressources et d'évaluation diagnostique pour l'amélioration rapide d'un système de TAL

Gaël de Chalendar, Damien Nouvel
CEA, LIST, Multilingual Multimedia Knowledge Engineering Laboratory,
F-92265 Fontenay-aux-Roses, France.
{Gael.de-Chalendar,Damien.Nouvel}@cea.fr

Abstract

Les systèmes de Traitement automatique des Langues (TAL) sont des logiciels complexes dont le développement implique de nombreuses années-homme de travail, tant en termes de codage que de développement des ressources. Étant donnés un dictionnaire de 110k lemmes, quelques centaines de règles d'analyse syntaxique, des matrices de 20k n-grams et d'autres ressources, quel sera l'impact sur un analyseur syntaxique de l'ajout d'une nouvelle catégorie possible à un verbe donné ? Quelles seront les conséquences de l'ajout d'une nouvelle règle syntaxique ? Toute modification peut causer, au-delà de ce qui était attendu, des effets de bord difficilement prévisibles. La complexité du système rend difficile de prévoir l'impact global même de petits changements. Nous présentons ici un ensemble d'outils conçus pour permettre d'améliorer de façon efficace et itérative la qualité de notre analyseur linguistique LIMA par raffinement répêté de ses ressources linguistiques. Ces améliorations sont constamment jugées par une évaluation de la performance de l'analyseur sur un corpus de référence. Nos premiers résultats montrent que ces outils sont vraiment utiles pour atteindre cet objectif.

1 Introduction

1.1 Le framework d'évaluation

En Traitement Automatique des Langues (TAL), la robustesse et la fiabilité des analyseurs linguistiques est un problème de plus en plus présent, étant données la taille croissante des ressources et la quantité de code impliqués par la mise en œuvre de tels systèmes. Au-delà du choix d'une technologie d'analyse solide, on doit désormais adjoindre au système des outils efficaces et conviviaux, pour évaluer son efficacité. Comme l'a montré (Chatzichrisafis et al., 2007), où les développeurs reçoivent des rapports quotidiens de performance afin d'améliorer leur système, l'évaluation systématique avec des tests de régression est utile pour accélérer l'ingénierie grammaticale.

Les campagnes d'évaluation, où plusieurs participants évaluent les performances de leur système sur une tâche spécifique par rapport aux autres systèmes, sont un bon moyen pour rechercher des directions dans lesquelles un système peut être en mesure d'améliorer ses performances. Souvent, ces campagnes d'évaluation donnent aussi la possibilité aux participants d'exécuter leurs analyseurs sur les données de test et d'obtenir les résultats détaillés de l'évaluation. Dans ce contexte, les auteurs d'analyseurs peuvent compter sur les campagnes d'évaluation pour fournir des informations de performances, mais ils doivent également être en mesure d'évaluer et d'améliorer continuellement leurs analyseurs entre les campagnes d'évaluation. Nous visons à proposer un outil d'évaluation générique, en utilisant des données issues de campagnes d'évaluation pour évaluer les systèmes. Ce logiciel sera dé-

signé sous le terme “Benchmarking Tool”.

Les outils de TAL ont toujours besoin de plus de ressources pour l'analyse de textes. Ces ressources ont assez augmenté (en termes de volume et diversité), pour que ce soit désormais un défi que de les manipuler, même pour les utilisateurs expérimentés. En outre, il est nécessaire de permettre à des non-développeurs de pouvoir travailler sur ces ressources : il faut donc développer des outils accessibles à travers une interfaces graphique intuitive. Un tel outil graphique d'édition de ressources représente la seconde partie de notre contribution, appelée “Resource Tool”.

Le tableau général est de construire un framework de diagnostic permettant à un spécialiste des langues, tel un linguiste, d'établir en quasi-temps réel quels sont les conséquences d'une modification de ressources sur les performances de notre analyseur, et ce sur la plus grande quantité de données de test possible. Pour les analyseurs, chaque ressource peut avoir un effet sur la précision finale de l'analyse. Il est souvent nécessaire d'effectuer plusieurs itérations de tests avant de comprendre quelles ressources ou quelle partie du code doit être améliorée. Ceci est particulièrement le cas avec l'ingénierie grammaticale, dans laquelle il est difficile de prévoir les conséquences de la modification d'une simple règle. Idéalement, notre framework devrait permettre à l'utilisateur de modifier légèrement une ressource, de déclencher une évaluation et, presque instantanément, de voir les résultats et les interpréter. Avec ce framework, nous nous attendons à une importante accélération du processus d'amélioration de notre analyseur.

Dans le reste de cette introduction, nous allons décrire notre analyseur et Passage, un projet collaboratif comprenant une campagne d'évaluation et la production d'un treebank de référence pour le français grâce à une procédure de vote entre les analyseurs participants. La section 2.1 décrit notre framework d'évaluation, son architecture, ses deux modules principaux et les premiers résultats liés à son utilisation. La section 3 décrit quelques travaux proches. Nous concluons à la section 4 en décrivant les prochaines étapes de notre travail.

1.2 L'analyseur linguistique LIMA

Notre analyseur linguistique LIMA (LIC2M Multilingual Analyzer, (Besançon and Chalendar (de), 2005)), est implémenté comme un pipeline de modules indépendants appliqués successivement sur un texte. Il met en œuvre une grammaire de dépendance (Kahane, 2000) en ce sens que les analyses produites sont exclusivement représentées comme des relations de dépendance binaires entre les tokens. L'analyseur comprend, entre autres modules, un segmenteur en tokens reposant sur les signes de ponctuation, un étiqueteur morphosyntaxique, des extracteurs de dépendances à courte et longue portée fondés sur des automates à états finis définis par des règles contextuelles. Ces

```
@AttributSuj:(@VerbeCopule|@VerbePrincipal) \
    %AdvGroup::SYNTACTIC_RELATION:
+!GovernorOf(trigger.1,"ATB_S")
+!GovernorOf(trigger.1,"COD_V")
+!GovernorOf(trigger.1,"CPL_V")
+CreateRelationBetween(trigger.1,left.1,"ATB_S")
=>AddRelationInGraph()
=<ClearStoredRelations()
```

FIG. 1 – Une règle d'analyse syntaxique de LIMA

règles dont un exemple est donnée en Figure 1 expriment des successions de catégories décrites par un token déclencheur, un contexte gauche et un contexte droit. Les contextes gauche et droit sont décrits à l'aide d'expressions régulières dont les éléments peuvent être des lemmes, des catégories morphosyntaxiques, des étiquettes d'entités nommées ou encore des identifieurs marquant la reconnaissance de précédentes règles syntaxiaugmentéesques. De plus, ces expressions décrivant les successions de tokens peuvencontraintest être complétées par des contraintes sur les flexions des mots, la présence ou l'absence de relations dépendances, etc. (Cf. les lignes précédées d'un signe +). Les contraintes sont vérifiées et exécutées au fur et à mesure du parcours du graphe d'analyse. Si la règle est un succès, l'action de succès (=>) est exécutée, sinon c'est c'est le case de l'action d'échec (=<). L'analyseur comprend également des modules de reconnaissance des expressions idiomatiques et des entités nommées qui, une fois reconnues, sont fusionnées en un seul token, permettant ainsi de simplifier les règles de grammaire. En outre, certains modules sont dédiés

au traitement de phénomènes spécifiques à certaines langues, comme par exemple le découpage du chinois en tokens, le traitement des formes composées en allemand, etc. Actuellement, l'analyseur est en mesure de traiter plus ou moins profondément dix langues dont l'anglais, l'espagnol, le chinois, l'arabe, le français et l'allemand.

1.3 Le projet Passage

Ce travail s'inscrit dans le cadre du projet Passage (de la Clergerie et al., 2008). L'objectif de ce projet est double. Tout d'abord, il organise deux campagnes d'évaluation des analyseurs syntaxiques (environ 15 systèmes participants) du français. Ensuite, il a pour objectif de produire un treebank de référence pour le français de grande taille par la fusion des sorties de tous les analyseurs participants en utilisant une approche Rover (Recognizer Output Voting Error Reduction) (Fiscus, 1997).

Dans ce projet, les annotations syntaxiques sont produites dans un format commun, suffisamment riche pour représenter la plupart des phénomènes syntaxiques et assez simple pour permettre à tous les participants (qui utilisent des techniques d'analyse très différentes) de représenter leurs analyses dans ce format. Il s'agit d'une évolution du format développé dans le cadre de la campagne EASy, qui regroupe des syntagmes simples non récursifs et des relations de dépendence entre syntagmes ou tokens. L'analyse syntaxique d'un corpus en format Passage fournit des informations sur :

- la segmentation du corpus en phrases ;
- la segmentation des phrases en formes ;
- les syntagmes typés non récursifs (types listés en Table 1) incluant les formes ;
- les relation de dépendance typées (listées en Table 2) ancrées ou bien sur les formes ou bien sur les syntagmes.

Type	Explication
GN	Groupe Nominal
NV	Noyau Verbal
GA	Groupe Adjectival
GR	Groupe Adverbial
GP	Groupe Prépositionnel
PV	Noyau Verbal Prépositionnel

TAB. 1 – Types de syntagmes

Dans le projet EASy qui a précédé Passage, les analyseurs ont été évalués par rapport à une référence, qui était elle-même une petite partie du corpus. La référence a été créée par l'annotation manuelle de phrases choisies aléatoirement dans le corpus. Après la fin de la campagne, le corpus annoté de référence a été porté à la connaissance des participants. Actuellement, nous utilisons cette référence pour l'analyse comparative de notre analyseur.

Type	Explication
SUJ-V	Sujet-verbe
AUX-V	Auxiliaure-verbe
COD-V	Oject Direct-verbe
CPL-V	Autre complément-verbe
MOD-V	Modifieur-verbe(p.ex. adverbes)
COMP	Phrases Subordonnées
ATB-SO	Attribut du sujet ou de l'objet
MOD-N	Modifieur de nom
MOD-A	Modifieur d'adjectif
MOD-R	Modifieur d'adverbe
MOD-P	Modifieur de préposition
COORD	Coordination
APPOS	Apposition
JUXT	Juxtaposition

TAB. 2 – Types de dépendences

1.4 Métriques pour l'évaluation de l'analyse syntaxique

Les métriques d'évaluation et les méthodologies évoluent et sont soumises à un intense travail de recherche et d'innovation (Carroll et al., 2002). Discuter ces métriques n'est pas l'objectif de cet article. Nous devons seulement être capable d'en appliquer le plus grand nombre possible sur le corpus entier ou n'importe quelle partie de celui-ci. L'évaluation est supposée, pour chaque document d et pour chaque type (de syntagme ou de dépendance) t, retourner les décomptes suivants :

- Nombre d'éléments trouvés et corrects - $fc(d,t)$
- Nombre d'éléments trouvés - $f(d,t)$
- Nombre d'éléments corrects - $c(d,t)$

Ainsi, nous pouvons calculer les métriques standard en Recherche d'Information (van Rijsbergen, 1079) : précision, rappel, F-mesure. Nous introduisons aussi une nouvelle mesure qui donne une indication de quels sont les types qui ont l'impact le plus négatif sur les performances générales. Cette mesure est appelée "global error impact" :

$$\frac{f(d,t) + c(d,t) - 2.fc(d,t)}{\sum^{t \in T} f(d,t) + c(d,t) - 2.fc(d,t)} \quad (1)$$

Cette métrique compte le nombre d'erreurs et d'oublis pour un type donné par rapport au

nombre total d'erreurs et d'oublis. Elle nous permet de quantifier de quel ordre une amélioration sur un certain type améliorera le score global du système. Dans notre cas, les scores sont calculés sur les syntagmes d'un côté et sur les dépendances de l'autre. Par exemple, les erreurs de GN représentent 34,6% des erreurs de syntagmes, tandis que celles sur les PV ne représentent que 2,2% : nous aurons ainsi plus d'intérêt à travailler sur les GN que sur les PV dans le cadre de la campagne d'évaluation actuelle.

2 Le framework d'évaluation

2.1 Architecture

Notre framework doit être portable et implémenté selon une approche agile : chaque nouvelle version doit être pleinement fonctionnelle tout en ajoutant quelques fonctionnalités supplémentaires. Il doit aussi être facile à utiliser, en permettant d'ajouter des fonctionnalités visuellement plaisantes et bien intégrées dans l'environnement de bureau. C'est pourquoi nous avons choisi d'implémenter ces outils en C++ en utilisant la bibliothèque Qt 4.5[1]. Cette bibliothèque satisfait à nos exigences tout en étant stable et en étant un Logiciel Libre (LGPL), nous laissant la possibilité de diffuser le moment venu notre framework sous une licence libre.

Cette approche nous permet de livrer rapidement un logiciel fonctionnel, tout en le testant et en poursuivant son développement. Les itérations de ce processus se poursuivent même si la version actuelle, avec ses fonctionnalités principales, nous permet déjà d'exécuter les évaluations et de commencer les améliorations de nos ressources linguistiques tout en livrant des versions améliorées du framework. Les premiers résultats de ce travail vont être présentés ci-dessous.

L'architecture ouverte que nous avons choisie implique d'utiliser des outils externes, pour l'analyse et l'évaluation d'une part, pour compiler et installer les ressources d'autre part. Ces outils peuvent être considérés comme des boîtes noires, étant des commandes externes dôtés des paramètres ad-hoc. En particulier, le *Benchmarking Tool* dépend de deux commandes : celle d'analyse qui prend en entrée le fichier à analyser et qui produit le fichier analysé et celle d'évaluation qui prend en entrée le fichier analysé et celui de référence et qui produit en sortie les décomptes d'éléments trouvés, corrects et trouvés et corrects pour chaque dimension. Ceci permet, par exemple, de remplacer notre analyseur par un autre en plaçant simplement celui-ci dans une fine capsule de conversion pour convertir ses entrées et ses sorties.

[1] http ://www.qtsoftware.com/

2.2 Benchmarking Tool

Le *Benchmarking Tool*, dont l'architecture est décrite par la Figure 2, est responsable de l'exécution des analyses et des évaluations sur des paires de fichiers de référence et de données, en utilisant les commandes décrites par son fichier de configuration. Pour chaque paire de fichiers, la commande d'analyse est exécutée suivie par celle d'évaluation.

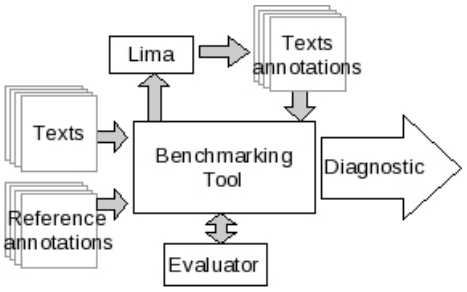

FIG. 2 – Flot de données pour le *Benchmarking Tool*

On peut considérer les types de syntagmes et de dépendances comme les dimensions d'une évaluation. D'une certaine manière, ces dimensions peuvent être associées au test de phénomènes linguistiques, comme proposé par le projet TSNLP (Balkan et al., 1994) ou, plus récemment, pour les systèmes de Question/Réponse par (de Paiva and King, 2008). Mais dans ces projets, l'accent est mis sur les outils d'évaluation eux-mêmes, tandis que nous n'implémentons pas nous-mêmes ces outils mais dépendons d'un programme externe pour l'évaluation.

Les paires de fichiers de référence sont insérées dans une file qui peut être modifiée (en ajoutant, retirant ou en réordonnant des éléments) avec des éléments d'interface standards. Après la création de la file, l'utilisateur peut lancer une évaluation dont la progression est indiquée en colorant les éléments selon

leur état (en attente, analyse, évaluation, terminé). L'évaluation peut être à tout moment suspendue, reprise ou définitivement abandonnée. L'évaluation peut exploiter ou bien les processeurs locaux ou bien les noeuds d'un cluster. Dans les deux cas, le nombre de processus concurrents à lancer est configurable. Au fur et à mesure de l'arrivée des résultats, les tables et les graphiques sont mis à jour dans des vues montrant l'évaluation courante ainsi que les précédentes pour chaque dimension évaluée. Pour affiner le diagnostic, l'utilisateur peut choisir les dimensions à afficher ainsi que la métrique à calculer et quels éléments de la file montrer. Enfin, une évaluation peut être supprimée si les modifications apportées n'ont pas modifié les performances de la manière attendue.

L'outil souvegarde automatiquement la configuration et les résultats dans un fichier XML. Les résultats de l'évaluateur sont stockés pour chauqe unité et chaque dimension de façon à pouvoir calculer les métriques pour chaque unité ou bien pour le corpus dans son ensemble. De plus, la date et un commentaire utilisateur sont attachés à chaque évaluation. Les commentaires se sont avérés très utiles pour garder trace des changements effectués sur le code, les ressources linguistiques, les paramètres, etc.

Par exemple, durant le projet Passage, nous avons noté à l'utilisation du *Benchmarking Tool* que nous avions tout à coup de mauvais résultats sur la relation auxiliaire-verbe. Au vu des résultats précédents, nous avons constaté que le problème était apparu suite à l'introduction d'un ensemble de formes idiomatiques concernant les verbes pronominaux. Des test unitaires ont alors montré que l'analyse des participes passés pour les verbes pronominaux comportait un bug. Corriger le code nous a alors permis un gain de 10 points de F-mesure pour la dimension $AUX\text{-}V$ et de 0,3 pour l'ensemble des dépendances ($AUX\text{-}V$ présentant un *global error impact* de 2,6% au sein des relations de dépendance). Les résultats d'évaluation ont donc été enregistrés avec un commentaire approprié et d'autre problèmes ont pu être abordés.

Grâce à ces caractéristiques, l'outil offre une vue globale des résulats d'évaluation et de leur évolution dans le temps, étant donnés des données, des dimensions d'analyse et des métriques multiples. Ce faisant, il nous aide, sans manipulation complexe, à obtenir un résumé visuel des conséquences d'une modification sur le processus d'analyse. Bien plus, ces test permettent de rechercher les erreurs dans les ressources mais aussi dans le code, de manière à trouver comment enrichir nos ressources linguistiques ou identifier les problèmes dans notre code.

La Figure 3 montre l'outil d'évaluation. Le tableau central montre les mesures correspondant aux évaluations successives, les dimensions à afficher étant sélectionnables avec les cases à cocher situées en haut de l'interface. Les graphiques à gauche montrent les mêmes données, les évaluations successives étant affichées en abscisse et les valeurs en ordonnée. La place manque ici pour décrire deux autres fonctionnalités (visibles sur les onglets au dessus du tableau) : d'une part la possibilité d'explorer les erreurs individuelles et d'afficher avec des graphes les différences entre évaluations successives ou encore entre une évaluation et la référence ; et d'autre part, la possibilité de rentrer une phrase à la main et de visualiser graphiquement les résultats pour pouvoir tester un phénomène particulier.

2.3 Resource Tool

L'outil *Resource Tool*, dont la conception modulaire est décrite par la figure 4, a pour but de rendre l'édition de ressources linguistiques accessible à des personnes n'ayant pas une connaissance approfondie du fonctionnement interne de notre analyseur ni des capacités de programmeur. Enrichir les ressources nécessite de pouvoir employer des personnes spécialisées en linguistique même si elles ne sont pas formées aux spécificités de nos formats de stockage des ressources.

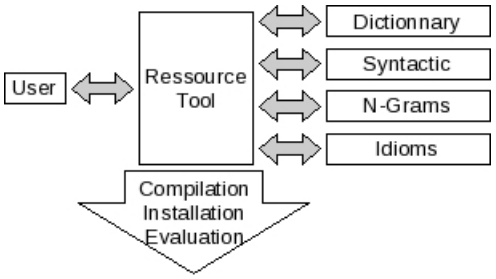

FIG. 4 – Conception du Resource Tool

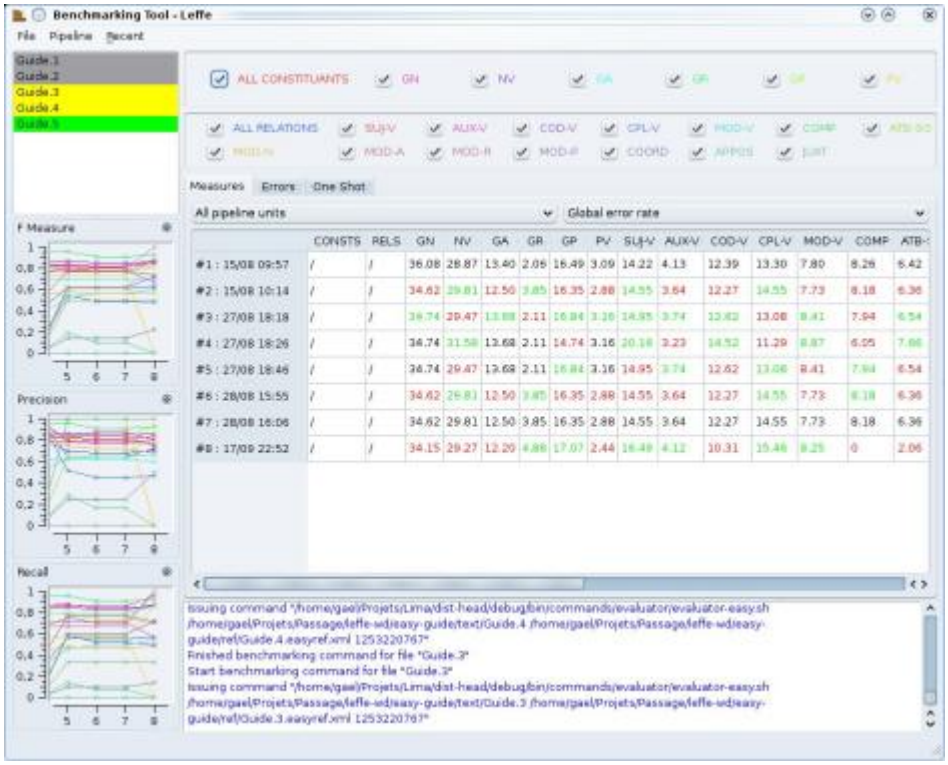

FIG. 3 – Une vue de l'outil d'évaluation

Dans sa version actuelle, le *Resource Tool* permet d'éditer les ressources suivantes :

- Dictionnaire de lemmes : entrées et leurs catégories
- Règles syntaxiques : expressions rationnelles enrichies de contraintes
- Corpus d'apprentissage pour l'étiqueteur morpho-syntaxique : exemples annotés de *ngrams*
- Expressions idiomatiques : expressions figées

Les ressources sont présentées dans des onglets, chacune ayant une interface dédiée. Pour chaque ressource une option de recherche est implémentée. L'outil permet de sauver, compiler et installer facilement les ressources une fois celles-ci éditées. Ceci doit aussi être très transparent pour l'utilisateur et nous avons simplement implémenté un bouton "sauver" et un autre "compiler et installer". La version actuelle est assez réduite du point de vue des possibilités d'édition. Le dictionnaire a une interface dédiée pour l'édition des lemmes et de leurs catégories, mais le corpus d'apprentissage, les règles syntaxiques et les expressions idiomatiques ne peuvent pour le moment être éditées qu'avec un éditeur de texte basique.

Le *Resource Tool* et le *Benchmarking Tool* communiquent entre eux avec des signaux : d'une part quand les ressources sont installées, le *Resource Tool* peut déclencher une évaluation dans le *Benchmarking Tool* et d'autre part, quand l'évaluation est terminée, le *Resource Tool* en est notifié et prévient l'utilisateur. Étant informés de leur état respectif, les outils peuvent mettre en garde l'utilisateur contre des opérations dangereuses comme tenter d'installer une ressource quand une évaluation est en cours ou quitter l'application avant que la dernière évaluation ne soit finie.

3 Travaux connexes

Nous avons plus haut décrit l'approche suite de tests, ainsi que le projet TSNLP. Cette approche a été concernée par l'identification et le test systématiques des phénomènes linguistiques. En conclusion de TSNLP, (Oepen and Flickinger, 1998) souligne la nécessité "pour évaluer l'impact des contributions individuelles, d'évaluer régulièrement la qualité de l'ensemble de la grammaire, et de le comparer aux versions précédentes". Ce projet a ainsi montré comment il est positif de constater les problèmes et d'améliorer les grammaires par itérations successives. C'est l'objectif que nous avons l'intention d'atteindre avec notre frame-

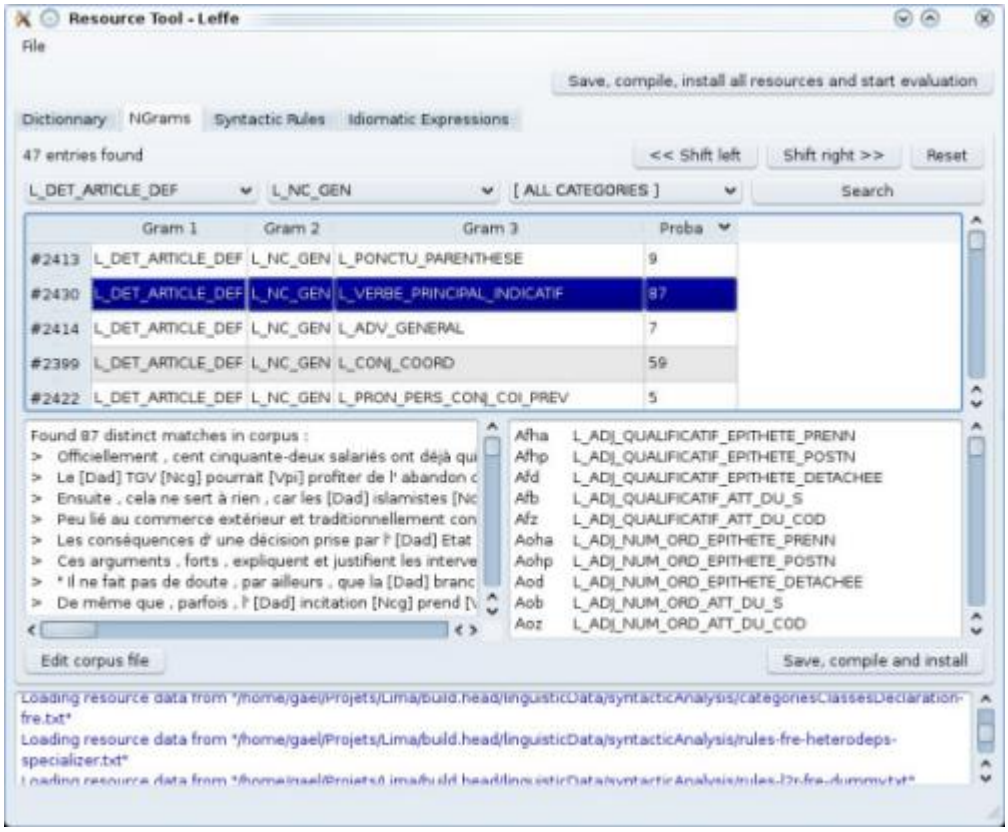

FIG. 5 – Vue et édition des matrices de désambiguïsation

work.

Plus récemment, dans le domaine biomédical, (Baumgartner et al., 2008) décrivent la mise en œuvre d'un framework et, même si elle est appliquée à une tâche de fouille de texte, la démarche reste assez proche de la nôtre dans ses fondements (orientée évaluation, tests itératifs, structure modulaire, open source, orientée corpus, etc) et encourage ce genre d'initiative en montrant l'importance de l'évaluation continue tout en codant l'analyseur et en éditant les ressources. Ce travail présente l'intérêt de s'appuyer sur le framework UIMA, permettant ainsi une bonne modularité. Dans l'avenir, nous comptons étudier l'intérêt de donner la possibilité à notre framework d'intégrer des modules UIMA.

4 Conclusion et perspectives

Notre utilisation préliminaire du framework nous a onvaincu de l'importance de tels outils diagnostiques pour accélérer l'amélioration de notre analyseur, en rendant les ressources linguistiques accessibles et en pouvant itérer les tests. Nous pensons aussi que ce framework pourrait être utile pour d'autres applications où l'évaluation est importante, comme la Recherche d'Information par exemple. En particulier, la recherche d'images est un champ de recherche très actif et nous envisageons actuellement d'utiliser notre outil d'évaluation pour accélérer le développement du moteur de recherche d'images développé au sein de notre laboratoire (Joint et al., 2004).

Ce travail met aussi en évidence la grande différence entre l'évaluation des performances et l'évaluation diagnostique. Dans notre cas, l'association des outils *Benchmarking Tool* et *Resource Tool* utilisés en conjonction avec des tests unitaires et des tests de régression nous aide à identifier les parties du processus d'analyse sur lesquelles il convient de travailler et, pour ce qui est de l'ingénierie de la grammaire, quelle règle ou ensemble de règles doivent être retravaillées pour améliorer la performance globale du système.

Les travaux futurs incluent l'amélioration de la parallélisation de l'analyse sur cluster. La répartition actuelle est sous-optimale, étant gérée par de simples scripts nécessitant de nombreuses synchronisations de fichiers et le rechargement de l'ensemble des ressources linguistiques à chaque exécution d'une analyse. Une telle amélioration nous permettrait d'utiliser les résultats d'évaluation sur de plus gros corpus. Nous comptons aussi poursuivre le dé-

veloppement de l'interface de consultation des erreurs, de façon à pouvoir accéder directement aux analyses erronnées. Une nouvelle itération de développement comprendra aussi celui d'interfaces d'édition de ressources améliorées.

Nous prévoyons aussi de travailler sur l'inférence automatique de règles d'analyse, en se fondant sur des travaux précédents dans notre laboratoire (Embarek and Ferret, 2008). Dans ce cadre, l'évaluation permanente sera encore plus importante, le système reposant sur des experts réglant les paramètres des algorithmes d'apprentissage et devant valider les règles apprises, celles-ci n'étant pas nécessairement éditables directement.

Enfin, nous comptons rendre encore plus modulaires les logiciels en développant un système de plugings facilitant le remplacement d'un outil d'évaluation par un autre ou encore ceux de visualisation des erreurs. Ainsi, le framework pourra être plus facilement appliqué à d'autres tâches que l'analyse syntaxique.

Remerciements

Ce travail est partiellement financé par l'Agence Nationale de la Recherche (ANR), dans le cadre du projet Passage.

Bibliographie

Lorna Balkan, Klaus Netterz, Doug Arnold, and Siety Meijer. 1994. Test suites for natural language processing. In *Proceedings of the Language Engineering Convention (LEC'94)*, pages 17–22.

William A Baumgartner, Kevin Bretonnel Cohen, and Lawrence Hunter. 2008. An open-source framework for large-scale, flexible evaluation of biomedical text mining systems. *Journal of Biomedical Discovery and Collaboration*, 3(1) :1.

Romaric Besançon and Gaël Chalendar (de). 2005. L'analyseur syntaxique de lima dans la campagne d'évaluation easy. In *actes de la 12e conférence annuelle sur le Traitement Automatique des Langues Naturelles, TALN 2005*, Dourdan, France, June.

John Carroll, Anette Frank, Dekang Lin, Detlef Prescher, and Hans Uszkoreit. 2002. Proceedings of the workshop beyond parseval - toward improved evaluation measures for parsing systems. In *Proceedings of the 3rd International Conference on Language Resources and Evaluation (LREC'02).*

Nikos Chatzichrisafis, Dick Crouch, Tracy Holloway King, Rowan Nairn, Manny Rayner, and Marianne Santaholma. 2007. Regression testing for grammar-based systems. In *Proceedings of the GEAF07 Workshop*, pages 128–143.

Eric V. de la Clergerie, Christelle Ayache, Gaël Chalendar (de), Gil Francopoulo, Claire Gardent, and Patrick Paroubek. 2008. Large scale production of syntactic annotations for french. In *Proceedings of the international workshop on Automated Syntactic Annotations for Interoperable Language Resources, Hong-Kong.*

Valeria de Paiva and Tracy Holloway King. 2008. Designing testsuites for grammar-based systems in applications. In *Proceedings of the GEAF08 Workshop*, pages 49–56.

Mehdi Embarek and Olivier Ferret. 2008. Learning patterns for building resources about semantic relations in the medical domain. In *Proceedings of the 6th Conference on Language Resources and Evaluation (LREC'08).*, Marrakech, Morocco.

Jonathan G. Fiscus. 1997. A post-processing system to yield reduced word error rates : Recognizer output voting error reduction (rover). In *Proceedings IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU97)*, pages 347–352.

Magali Joint, Pierre-Alain Moellic, Patrick Hede, and Pascal Adam. 2004. Piria : a general tool for indexing, search, and retrieval of multimedia content. In *Proceedings of SPIE*, volume 5298, San Jose, CA, USA.

Sylvain Kahane. 2000. Les grammaires de dépendance. *Traitement Automatique des Langues*, 41.

Stephan Oepen and Daniel P. Flickinger. 1998. Towards systematic grammar profiling. test suite technology ten years after. *Journal of Computer Speech and Language. Special Issue on Evaluation.*, 12(4) :411–436, June.

C. J. van Rijsbergen. 1079. Designing testsuites for grammar-based systems in applications. In *Information Retrieval, 2nd edition.*, pages 49–56.