Analyse syntaxique profonde à grande échelle : SXLFG

Pierre Boullier — Benoît Sagot

INRIA Rocquencourt - Projet ATOLL
Domaine de Voluceau
Rocquencourt, B.P. 105
F-78153 Le Chesnay cedex
{pierre.boullier,benoit.sagot}@inria.fr

RÉSUMÉ. *Cet article présente un nouvel analyseur syntaxique, nommé* SXLFG*, qui repose sur le formalisme des Grammaires Lexicales Fonctionnelles (*Lexical Functional Grammars, LFG*). Nous décrivons l'analyseur non contextuel sous-jacent ainsi que la façon dont les structures fonctionnelles sont efficacement calculées sur la forêt partagée résultant de l'analyse non contextuelle. Nous présentons ensuite les différentes techniques de rattrapage et de tolérance d'erreur que nous avons implémentées pour en faire un analyseur robuste. Enfin, nous donnons des résultats concrets de l'utilisation de* SXLFG *avec une grammaire du français à large couverture. Nous montrons que notre analyseur, tout en étant un analyseur profond non probabiliste, est à la fois efficace et robuste et permet l'analyse rapide de très gros corpus, bien que la grammaire utilisée pour l'évaluation soit très ambiguë.*

ABSTRACT. *In this paper, we introduce a new parser, called* SXLFG*, based on the* Lexical-Functional Grammars *formalism (*LFG*). We describe the underlying context-free parser and how functional structures are efficiently computed on top of the CFG shared forest thanks to computation sharing, lazy evaluation, and compact data representation. We then present various error recovery and tolerance techniques we implemented in order to build a robust parser. Finally, we offer concrete results when* SXLFG *is used with an existing grammar for French. We show that our parser, while being a deep and non-probabilistic parser, is both efficient and robust, allowing to parse huge corpora, although the grammar used for the evaluation is very ambiguous.*

MOTS-CLÉS : *analyse profonde, analyseur LFG, robustesse,eEfficacité, analyse de corpus volumineux.*

KEYWORDS: *deep parsing, LFG parser, robustness, efficiency, parsing of large corpora.*

1. Introduction

Pour surmonter les difficultés algorithmiques des analyseurs syntaxiques lorsqu'ils sont utilisés sur des corpus réels, il est désormais habituel d'utiliser des méthodes efficaces et robustes telles que les techniques markoviennes ou les automates finis. Ces méthodes sont parfaitement adaptées pour nombre d'applications qui n'ont pas explicitement besoin de représentations complexes de la phrase. Cependant, le pouvoir d'expression des analyses ainsi obtenues est bien en-deçà de ce qui est nécessaire pour représenter, par exemple, les syntagmes ou les dépendances à longue distance d'une façon qui soit en accord avec des définitions linguistiques fines de ces concepts. Pour cette raison, nous avons développé un analyseur syntaxique qui est compatible avec une théorie linguistique, à savoir LFG, tout en étant robuste et efficace en dépit de la grande variabilité des productions langagières.

Le développement d'un nouvel analyseur syntaxique pour le formalisme LFG (*Lexical-Functional Grammars*, cf. p. ex. (Kaplan, 1989)) n'est pas en soi très original. Il en existe déjà un certain nombre, comme ceux de (Andrews, 1990) ou (Briffault *et al.*, 1997). Mais le plus connu des systèmes d'analyse LFG est sans conteste le projet XLE (Xerox Linguistic Environment), qui est le successeur du Grammars Writer's Workbench (Kaplan *et al.*, 1994, Riezler *et al.*, 2002, Kaplan *et al.*, 2004). XLE est un projet d'importance, qui concentre un grand nombre de compétences linguistiques et techniques, repose sur un point de vue similaire au nôtre quant à l'équilibre entre analyse de surface et analyse profonde, et a été utilisé avec succès pour analyser des corpus bruts de taille importante.

Toutefois, ces analyseurs n'utilisent pas toujours de la manière la plus complète possible les différentes techniques algorithmiques de partage de calcul et de représentation compacte de l'information qui permettent d'écrire un analyseur efficace malgré le fait que le formalisme LFG, comme de nombreux formalismes qui reposent sur l'unification, est NP-complet. Naturellement, notre but n'est pas d'écrire un nouvel XLE mais d'étudier comment robustesse et efficacité peuvent être atteints dans l'analyse syntaxique LFG de corpus bruts.

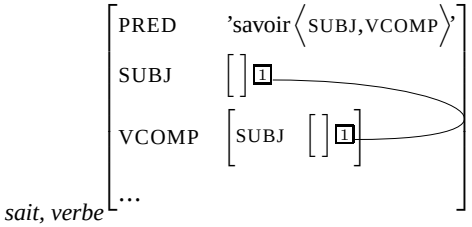
La construction des structures de constituance (ou c-structures) ne pose théoriquement[1] pas de problème particulier, car elles sont décrites par des grammaires non contextuelles (CFG) sous-jacentes aux LFG et appelées ici grammaires *squelettes*. En effet, les algorithmes généraux d'analyse pour les CFG sont bien connus (Earley, GLR, CKY,...). En revanche, la construction efficace des structures fonctionnelles (ou f-structures) est beaucoup plus problématique. Nous avons développé un module de calcul de ces structures qui partage les sous-structures communes à plusieurs analyses. De plus, des mécanismes de rattrapage d'erreur à tous les niveaux permettent d'obtenir un analyseur robuste.

En parallèle, nous avons adapté une grammaire LFG du français pour obtenir un analyseur du français à la fois efficace et raisonnablement couvrant.

1. Même si, en pratique, la disponibilité d'un *bon* analyseur est déjà plus délicate.

2. Formalisme employé

Le formalisme employé est une variante de LFG, qui se distingue du formalisme standard sur deux points principaux. Tout d'abord, les entrées lexicales ne contiennent ni équations fonctionnelles ni règles lexicales. Il ne s'agit là que d'optimisations : d'une part les équations fonctionnelles sont remplacées dans le lexique par la structure fonctionnelle correspondante, qui est le résultat de l'application de ces équations ; d'autre part on remplace les règles lexicales par le résultat de leur application (on aura donc des entrées lexicales correspondant par exemple à la forme impersonnelle, au lieu d'indiquer que la règle lexicale « impersonnel » est applicable sur l'entrée adéquate). Ainsi, les entrées lexicales sont des triplets contenant une forme fléchie, une catégorie (c'est-à-dire un terminal de la grammaire squelette) et une structure fonctionnelle sous-spécifiée avec partage possible de structures. Un exemple d'entrée lexicale, avec la façon dont elle est encodée et la règle LFG standard correspondante, est indiquée à la figure 1 (elle définit une forme fléchie *sait* qui est de la catégorie *verbe* et dont la structure fonctionnelle associée contient elle-même une structure fonctionnelle vide et partagée, repérée par [1]).




```
sait   verbe   [pred="savoir<subj,vcomp>", ... , @CtrlSubj] ;
@CtrlSubj = [subj=[]1, vcomp=[subj=[]1]] ;
```

Verbe → sait
(↑ pred) = 'savoir <subj,vcomp>'
(↑ vcomp subj) = (↑ subj)

Figure 1. *Entrée lexicale (partielle) pour une forme verbale à contrôle sujet, codage de cette entrée* via *une macro et règle correspondante en LFG standard*

La seconde différence importante est que les équations fonctionnelles sont interprétées en sorte qu'elles ne puissent pas *modifier* les structures fonctionnelles associées aux non-terminaux de partie droite d'une règle. D'une manière générale, le calcul de l'information est strictement synthétisé. Le contenu de la ou des structures fonctionnelles attachées au non-terminal de la partie gauche d'une règle de la grammaire squelette ne peut qu'être une constante ou se calculer (par unification) à partir des structures fonctionnelles attachées aux symboles de la partie droite de cette règle (structures directement données dans le lexique pour les symboles terminaux,

commme vu précédemment). Toutefois, le calcul des structures fonctionnelles associées au symbole de partie gauche peut utiliser des *copies locales* des structures fonctionnelles associées aux symboles de partie droite. Ces copies locales peuvent être modifiées, ce qui a pour effet que la puissance d'expression n'est pas diminuée par rapport au modèle LFG standard. La seule différence est que l'obtention des structures fonctionnelles associées aux nœuds internes de la forêt nécessiterait un nouveau parcours de la forêt. Nous n'avons pas implémenté ce parcours, considérant que ce que nous construisons est suffisant : la forêt représentant les structures en constituants, les structures fonctionnelles associées à la racine de la forêt, dites structures fonctionnelles *principales*, et des repères permettant d'associer des sous-structures des structures fonctionnelles principales à des non-terminaux de la forêt (voir plus bas). Cette vision purement synthétisée, qui ne restreint pas la puissance d'expression de LFG, est la clef de notre stratégie d'évaluation, car elle permet d'introduire du partage de calcul pendant la construction des structures fonctionnelles et de réaliser cette construction sur la forêt d'analyse et non sur les analyses individuelles prises successivement (voir plus bas).

À titre d'exemple, considérons la règle permettant de décrire la disposition des clitiques d'un verbe fini en cas d'extraction clitique du génitif modifieur de l'objet (exemple : *Jean en mange une partie*). La règle complète et sa représentation en SXLFG sont données dans la figure 2. Comme (Clément *et al.*, 2001), nous écrivons toutes les équations fonctionnelles au même niveau, en remplaçant l'opérateur $\downarrow$ associé au n-ième symbole de partie droite par $\downarrow n$.

CLITICS → (cld) clg (cll)	`CLITICS -> (cld) clg (cll)`
$\downarrow 1$: ($\uparrow$ à-obj = $\downarrow 1$)	`$1 : ($$ à-obj = $1)`
($\uparrow$ obj de-obj) = $\downarrow 2$	`$$ obj de-obj = $2)`
$\downarrow 3 \in$ ($\uparrow$ adjunct)	`$3 < ($$ adjunct)`

Figure 2. *Les clitiques dans le cas d'une extraction du génitif modifieur de l'objet (règle simplifiée et codage). Les terminaux* `cld`*,* `clg` *et* `cll` *désignent respectivement les pronoms clitiques datif (*lui,*...), génitif (*en*) et « locatif » (*y*)*

Les opérateurs fonctionnels utilisés sont les suivants (un identifiant est un nom d'attribut, une expression régulière d'attributs pouvant utiliser les opérateurs de disjonction et d'itération de Kleene, ou un identifiant sous-spécifié de type (`$`i `pcas)-obj`) :

unification (opérateur « = ») : l'équation est vérifiée si et seulement si l'unification entre les deux structures opérandes est possible et si elle réussit ; un échec de l'unification fait échouer le calcul de la structure `$$` (c'est-à-dire $\uparrow$) associée au non-terminal de partie gauche,

parcours constructif de structure (opérateur « ») : avec un chemin vers une structure S en partie gauche et un identifiant i en partie droite (identifiant atomique ou expression régulière), rend la sous-structure de S de nom i en la créant si elle n'existe pas,

parcours non-constructif de structure (opérateur « . ») : comme le précédent, mais en cas de non-existence cela échoue ; cet opérateur est utilisé lorsque la partie droite est une expression régulière, pour parcourir toutes les structures existantes qui sont reconnues par l'expression régulière, sans créer de structure supplémentaire (s'il n'y en a pas, échec),

vérification de la présence (resp. absence) d'un attribut (opérateur « + » resp. « - ») : l'équation est vérifiée si la structure opérande existe (resp. n'existe pas),

unification « contrainte » (opérateur « =c ») : comme l'unification, mais la structure finale d'une analyse ne doit pas comporter d'attribut ayant subi une unification contrainte sans avoir subi également une unification,

ajout dans la liste d'adjoints (opérateur « < ») : ajoute le premier opérande au second, ce dernier devant être une liste d'adjoints, c'est-à-dire un chemin se terminant par un identifiant déclaré de type *liste d'adjoints* (classiquement `adjunct`),

conditionnement (opérateur « : ») : permet de contraindre la vérification d'une équation au fait qu'un symbole optionnel de partie droite soit effectivement réalisé ; par exemple, une équation de la forme `$1 :(éq)` est vérifiée soit parce que le premier symbole de la partie droite (`cld` dans l'exemple de la figure 2) dérive dans la chaîne vide, soit parce que l'équation `éq` est vérifiée.

Enfin, un mécanisme permet de représenter pour une structure fonctionnelle donnée les structures en constituants qui lui correspondent (voir plus bas).

3. Analyse standard

3.1. *Architecture globale*

Le cœur d'un analyseur[2] produit par SXLFG est un analyseur CFG général qui traite le squelette CFG de la grammaire LFG. C'est un analyseur à la Earley qui repose sur un automate coin-gauche sous-jacent et qui est une évolution de (Boullier, 2003). L'ensemble des analyses produites par cet analyseur est représenté par une forêt partagée. En réalité, cette forêt partagée peut être vue elle-même comme une grammaire non contextuelle dont les productions sont des productions instanciées du squelette[3]. L'évaluation des équations fonctionnelles est réalisée au cours d'un parcours de bas en haut dans cette forêt. Un module de désambiguïsation, qui élimine les f-structures non sélectionnées, peut être alors invoqué à n'importe quel nœud de la forêt, y compris bien sûr à sa racine.

---

2. Nous appelons *analyseur* un programme construit à partir d'une grammaire et fournissant une ou plusieurs analyses selon cette grammaire pour une entrée donnée. Par conséquent, un constructeur d'analyseurs produit deux analyseurs différents à partir de deux grammaires différentes.

3. Si A est un symbole non terminal de la grammaire squelette G, $A_{i..j}$ est un *symbole non terminal instancié* si et seulement si $A \xRightarrow[G]{+} a_i \dots a_{j-1}$, où $w = a_1 \dots a_n$ est la chaîne d'entrée et $\xRightarrow[G]{+}$ la fermeture transitive de la relation *dérive*.

L'entrée de l'analyseur est un DAG[4] de mots (tous les mots étant alors connus du lexique, y compris certains mots spéciaux représentant les tokens inconnus du corpus initial). Ce DAG est converti par le *lexeur* en un DAG de lexèmes (un lexème étant ici un symbole terminal du squelette CFG auquel est associée une structure fonctionnelle sous-spécifiée).

3.2. *L'analyseur non contextuel*

Les évolutions de l'analyseur Earley par rapport à celui décrit dans (Boullier, 2003) sont de deux types : il accepte les DAG en entrée et il dispose de mécanismes de rattrapage d'erreurs. Ce second point sera traité dans la section 4.1. Savoir prendre des DAG en entrée ne nécessite pas de changements considérables dans l'algorithme d'Earley, au moins du point de vue théorique[5]. Puisque l'analyseur Earley est guidé par un automate fini coin-gauche qui définit un sur-langage régulier du langage défini par le squelette CFG, cet automate accepte également les DAG en entrée (ce qui correspond à l'intersection de deux automates finis).

3.3. *Calcul des f-structures*

3.3.1. *Remarques générales*

Comme noté dans (Kaplan *et al.*, 1982), si le nombre d'analyses CFG (c-structures) croît exponentiellement par rapport à la longueur de l'entrée, la construction et la vérification des f-structures associées prend un temps exponentiel. Nos expériences montrent que le squelette des grammaires LFG à large couverture peut être hautement ambigu (cf. section 5). Ceci veut dire que l'analyse (complète) de longues phrases serait irréalisable. Bien qu'il soit possible, en analyse non contextuelle, de calculer et de stocker en un temps polynomial un nombre exponentiel (ou même non borné) d'arbres d'analyse dans une forêt partagée, ce résultat ne peut être transposé dans le cas des f-structures pour plusieurs raisons[6]. Cependant, ce comportement rédhibitoire (et bien d'autres) pourraient bien ne pas avoir lieu dans les applications pratiques de TAL, sans compter qu'un certain nombre de techniques, dont certaines

4. *Directed Acyclic Graph* (Graphe dirigé acyclique).

5. Si i est un nœud du DAG et si l'on a une transition sur le terminal t vers le nœud j (sans perte de généralité, on peut supposer $j > i$) et si l'item Earley $[A \rightarrow \alpha \bullet t\beta, k]$ est un élément de la table $T[i]$, alors on peut ajouter à la table $T[j]$ l'item $[A \rightarrow \alpha t \bullet \beta, k]$ s'il n'y est pas déjà. Il faut faire attention à ne commencer une phase PREDICTOR dans une table $T[j]$ que si toutes les phases Earley (PREDICTOR, COMPLETOR et SCANNER) sont déjà achevées dans toutes les tables $T[i]$, $i < j$.

6. Par exemple, il est possible, en LFG, de définir des f-structures qui encodent les analyses CFG individuelles. Si une forêt partagée de taille polynomiale en la taille de l'entrée représente un nombre exponentiel d'analyses, le nombre de f-structures différentes associées à la racine de la forêt partagée serait ce même nombre exponentiel d'analyses. En d'autres termes, il y a des cas où aucun partage de calcul des f-structures n'est possible.

sont décrites à la section 3.4, peuvent être appliquées pour restreindre cette explosion combinatoire.

Le calcul efficace de structures se combinant par unification est encore un champ de recherche actif. Cependant, ce problème est simplifié si le calcul des structures considérées peut se faire de façon cumulative, comme c'est le cas en LFG. Dans ce cas, le squelette (c'est-à-dire la forêt partagée) n'a pas besoin d'être modifié pendant la résolution des équations fonctionnelles. Si l'on adopte une stratégie de parcours de bas en haut dans la forêt partagée, les informations stockées dans les f-structures sont calculées de façon *synthétisée*. Ceci signifie que l'évaluation des f-structures sur une sous-forêt[7] n'est effectuée qu'une seule fois, même si cette sous-forêt est partagée par de nombreux nœuds pères. En réalité, l'effet d'une évaluation complète des équations fonctionnelles est d'associer à chaque nœud de la forêt partagée un ensemble de structures fonctionnelles partielles qui ne dépendent que des descendants dudit nœud (mais pas de ses nœuds pères ou frères).

Dans le cas où l'analyse est un succès, le résultat de l'analyseur LFG est l'ensemble des *structures fonctionnelles principales* (complètes et cohérentes si possible), c'est-à-dire l'ensemble des f-structures associées à la racine de la forêt partagée. De tels ensembles de f-structures (qu'il s'agisse ou non de f-structures principales) pourraient être factorisés en une seule f-structure contenant des valeurs disjonctives, comme dans XLE. Nous avons décidé de ne pas utiliser de telles valeurs disjonctives complexes, sauf pour les valeurs atomiques, mais plutôt d'associer à toute f-structure (principale ou non) un identifiant unique : deux f-structures identiques auront toujours le même identifiant tout au long du processus, et ne seront donc pas dupliquées. L'expérience montre que cette stratégie est payante et que le nombre total de f-structures distinctes construites au long d'une analyse complète reste tout à fait raisonnable, sauf peut-être dans certains cas pathologiques.

3.3.2. *Mise en œuvre*

Le calcul des f-structures se fait au cours d'un ou de plusieurs parcours de bas en haut de la forêt produite par le squelette non contextuel (stratégie ascendante). Chacun de ces parcours est appelé une *passe*. À chaque passe, on fait toujours en sorte que le calcul des f-structures associées au symbole instancié[8] de partie gauche d'une production instanciée ne commence que lorsque l'on a terminé le calcul de toutes les f-structures associées à tous les symboles de partie droite de cette production instanciée. Dans le cas où un symbole de partie droite est un symbole terminal, les f-structures qui

---

7. Si le squelette non contextuel G est cyclique (c'est-à-dire $\exists A$ t.q. $A \overset{+}{\underset{G}{\Longrightarrow}} A$), la forêt peut être un graphe général et non seulement un DAG. Bien que le générateur d'analyseurs non contextuels utilisé sache traiter ce cas, nous l'excluons explicitement dans SXLFG. Naturellement, cette (petite) restriction ne veut pas dire que les structures fonctionnelles cycliques soient interdites. SXLFG sait gérer les f-structures cycliques, qui peuvent être un moyen élégant de représenter certaines relations linguistiques.

8. Rappelons qu'un symbole instancié est de la forme $A_{i..j}$, où i et j sont des positions si l'entrée de l'analyseur est une chaîne, ou des numéros d'états si c'est un DAG.

lui sont associées proviennent directement du lexique. Une fois le calcul fini, toutes les f-structures associées au symbole instancié de partie gauche sont regroupées dans un ensemble, auquel viennent s'ajouter toutes les f-structures calculées sur d'autres productions instanciées ayant le même symbole instancié en partie gauche.

L'évaluation des équations fonctionnelles associées à une production instanciée de la forêt partagée se fait comme suit. Une évaluation particulière des équations se fait en sélectionnant, pour chaque symbole instancié de la partie droite, une seule f-structure parmi l'ensemble des f-structures qui lui sont associées. Sur une production instanciée donnée, on effectue cette évaluation pour toutes les combinaisons de f-structures de partie droite possibles[9]. Chaque évaluation particulière peut ne produire aucun résultat (en cas d'échec d'unification), un résultat unique, ou plusieurs résultats, ce dernier cas n'ayant lieu qu'en présence d'équations comportant des expressions régulières sur les chemins, puisque plusieurs chemins peuvent conduire à un succès. Autrement dit, si l'on sélectionne un seul chemin pour chaque expression régulière qui intervient dans les équations, on obtient soit aucune f-structure résultat, soit une seule.

L'ordre dans lequel les équations fonctionnelles sont évaluées n'a aucune importance, à l'exception des équations comportant l'opérateur non-standard de *parcours non-constructif de structures* (opérateur « . »). Ces équations sont évaluées après toutes les autres, ce qui permet aux attributs référencés par cet opérateur d'avoir été construits par d'autres équations de la même règle[10].

L'évaluation se fait dans des structures de travail. Si la f-structure associée à un symbole instancié de partie droite est utilisée, elle est potentiellement copiée dans une structure de travail. En réalité, un mécanisme d'unification paresseuse a été mis en place, à l'image de ce qui se passe dans XLE[11]. Ce mécanisme permet de ne copier une f-structure dans la structure de travail que si elle est susceptible d'être modifiée. À la fin d'une évaluation réussie, la structure de travail associée au symbole instancié de partie gauche est sauvegardée dans une structure globale et reçoit alors son identifiant unique (voir ci-dessous). Cet identifiant est ajouté, s'il ne s'y trouve pas déjà, à l'ensemble des identifiants déjà calculés précédemment pour ce symbole instancié (sur la même production instanciée ou sur d'autres). Les structures de travail associées aux symboles instanciés de partie droite ne sont pas sauvegardées, et ce même si elles ont été modifiées, pour éviter qu'une autre production instanciée ayant en partie droite ce même symbole instancié ne récupère ces modifications qui ne le concernent pas

9. Naturellement, tout échec en cours d'évaluation disqualifie toutes les combinaisons comprenant les f-structures en jeu dans cet échec.

10. On peut construire des jeux d'équations où l'ordre dans lequel sont évaluées les équations comportant l'opérateur « . » change les résultats. Mais d'une part le développeur de grammaire peut contrôler l'ordre d'évaluation des équations, et d'autre part une telle situation semble ne pas arriver dans la pratique lorsque l'on développe des grammaires réelles.

11. Lorsque deux f-structures sont unifiées, nous ne copions que leurs parties *communes* qui sont nécessaire à la vérification de l'unifiabilité des deux f-structures. Ceci restreint la quantité de copies entre deux nœuds frères aux seules parties où ils interagissent. Naturellement, les nœuds-frères originaux sont laissés inchangés (et peuvent ainsi être utilisés dans un autre contexte).

du tout. C'est ce qui fait le caractère *synthétisé* de l'évaluation des f-structures. C'est aussi ce qui fait que SXLFG n'associe pas directement les mêmes f-structures aux nœuds internes de la forêt que le modèle LFG standard[12] (au niveau de la racine, cette différence n'existe pas). Nous considérons en effet qu'il est plus pertinent d'associer à ceux des nœuds internes de la forêt qui le méritent non pas une f-structure particulière mais plutôt des références à des sous-structures des f-structures associées à la racine. Pour cela, nous permettons aux (sous-)f-structures de référencer des nœuds de la forêt (c'est-à-dire de la c-structure). Ceci est fait grâce à un attribut spécial, nommé A_{ij}, qui accumule dans une liste des identifiants de nœuds de la forêt (plus spécifiquement, des identifiants de productions instanciées), à l'aide d'une équation spéciale qui demande l'ajout de l'identifiant de la production instanciée courante à l'attribut A_{ij} de la f-structure associée à son symbole instancié de partie gauche[13].

En général, l'association d'un identifiant unique à une f-structure quelconque ne pose pas de problème : c'est son contenu qui permet de déterminer un identifiant. Cependant, c'est plus problématique dans le cas de f-structures cycliques. Il faut évidemment assurer une identification cohérente (une f-structure cyclique d'identifiant f qui fait référence à elle-même, directement ou à travers des sous-structures, se fera référence par l'identifiant f). Nous avons opté pour un compromis qui garantit, grâce à l'évaluation paresseuse, que deux ensembles identiques de f-structures cycliques portent les mêmes identifiants dans de nombreux cas. Cependant, si un ensemble de f-structures cycliques est modifié, chaque f-structure de cet ensemble recevra un nouvel identifiant, même si ce nouvel ensemble préexistait dans la structure globale.

3.4. *Désambiguïsation interne et globale*

Les applications des systèmes d'analyse ont souvent besoin d'un résultat désambiguïsé, nécessitant ainsi l'application de techniques de désambiguïsation sur les sorties ambiguës des analyseurs tels que ceux générés par SXLFG. Dans notre cas, ceci implique le développement de procédures de désambiguïsation permettant de choisir l'analyse la plus probable (ou les analyses les plus probables) parmi les f-structures principales. Ensuite, la forêt partagée est élaguée, en sorte de ne garder que les c-structures qui sont compatibles avec la ou les f-structures principales choisies (si l'on

12. Bien que nous ne l'ayons pas réalisé, et comme suggéré précédemment, il serait facile de dupliquer les f-structures des nœuds internes et de compléter ces copies par les informations propres à chaque contexte. On obtiendrait alors une implémentation complète du modèle LFG standard. Le mécanisme des A_{ij}, évoqué ci-dessous, joue un rôle équivalent, et de façon mieux contrôlable.

13. Si l'évaluation de ces équations permettant le calcul des valeurs des A_{ij} était effectuée conjointement au calcul des autres équations, on pourrait se retrouver avec un nombre exponentiel de f-structures qui ne différeraient que par la valeur de leur champ A_{ij}. SXLFG permettant de spécifier pour chaque attribut un numéro de passe, on ne fait le calcul des A_{ij} que dans une passe qui suit tous les autres calculs. La ou les autres passes ayant alors éliminé un très grand nombre d'arbres invalides, le calcul des A_{ij} se fait sur un nombre restreint d'arbres, assurant par conséquent un gain potentiellement exponentiel en temps d'analyse.

ne choisit qu'une seule f-structure, on n'obtient en général qu'une seule c-structure, mais ce n'est pas nécessairement le cas). C'est ce que l'on appelle la *désambiguïsation globale*.

D'un autre côté, sur tout nœud interne de la forêt, un nombre potentiellement très grand de f-structures peut être obtenu. Si rien n'est fait, ces nombreuses structures peuvent conduire à une explosion combinatoire qui empêche l'analyse de se terminer en un temps raisonnable. C'est la raison pour laquelle il semble approprié de permettre au développeur de la grammaire de répertorier un ensemble de symboles non terminaux qui disposent d'une certaine forme de saturation linguistique qui rend possible l'application sur ces non-terminaux de techniques de désambiguïsation[14]. Ainsi, certains non-terminaux du squelette CFG qui correspondent à des portions de phrases linguistiquement saturées (par exemple des propositions) peuvent se voir attribuer une liste (ordonnée) de méthodes de désambiguïsation, chaque non-terminal pouvant recevoir sa propre liste. Ceci permet un filtrage immédiat en cours d'analyse des f-structures associées aux nœuds internes concernés qui auraient certainement (ou très probablement) conduit à des f-structures principales incomplètes, incohérentes, ou non sélectionnées par les mécanismes de désambiguïsation globale. Par ailleurs, cela conduit inévitablement à une réduction parfois très importante des temps d'analyse. Ce point de vue est en réalité une généralisation de la désambiguïsation classique (la désambiguïsation globale), en ce qu'elle en permet l'application sur d'autres nœuds que le nœud racine. Nous l'appelons *désambiguïsation interne*. On notera que nous employons le terme *désambiguïsation* dans un sens large, puisqu'une désambiguïsation, globale ou interne, peut conserver plus d'une analyse. En ce sens, *désambiguïsation* est à prendre comme un raccourci pour *désambiguïsation totale ou partielle*.

Les techniques de désambiguïsation sont en général partagées entre techniques probabilistes et techniques à règles. L'architecture de SXLFG permet l'implémentation des deux types de techniques, pourvu que les calculs puissent être réalisés sur les structures fonctionnelles. Elle permet d'associer un poids à chaque f-structure associée au non-terminal instancié considéré[15]. Appliquer une règle de désambiguïsation revient à éliminer toutes les f-structures qui ne sont pas optimales au sens de cette règle. Les règles associées au non-terminal courant sont appliquées en cascade (comme indiqué ci-dessus, la liste des règles et par conséquent leur nombre et leur ordre d'application peut varier d'un non-terminal à un autre).

Après l'application de ce mécanisme de désambiguïsation sur les structures fonctionnelles, la forêt partagée (qui représente les structures de constituants) est filtrée

14. Une telle approche est plus satisfaisante qu'un *skimming* à l'aveugle qui stoppe l'analyse normale de la phrase lorsque le temps ou l'espace utilisé dépasse une certaine limite fixée par avance, et qui la remplace par une analyse dégradée qui s'impose de faire une quantité de travail bornée sur chaque non-terminal restant (Riezler *et al.*, 2002, Kaplan *et al.*, 2004).

15. On se reportera à (Kinyon, 2000) pour une argumentation sur l'importance de désambiguïser sur des structures telles que les structures de dérivation TAG ou les f-structures LFG et non sur les structures de constituance.

afin de correspondre exactement à la f-structure ou aux f-structures conservée(s). En particulier, si la désambiguïsation est complète (seule une f-structure a été conservée), ce filtrage conduit en général à une c-structure unique (un arbre).

4. Techniques d'analyse robuste

4.1. *Rattrapage d'erreurs dans l'analyseur non contextuel*

La détection d'une erreur dans l'analyseur Earley[16] peut être due à deux phénomènes différents qui peuvent se cumuler : le squelette CFG n'a pas une couverture suffisante, ou l'entrée n'est pas une phrase correcte de la langue. Naturellement, bien que l'analyseur ne puisse pas faire la différence entre ces deux causes, les développeurs de analyseurs et de grammaires doivent les gérer différemment. Dans les deux cas, l'analyseur doit être en mesure de faire du *rattrapage* afin de continuer l'analyse et d'analyser correctement, autant que faire se peut, les portions correctes et incorrectes en préservant des relations raisonnables entre elles. La gestion des erreurs dans les analyseurs est un domaine de recherche qui a été principalement étudié dans le cas déterministe et rarement dans le cas général d'analyseurs non-déterministes, y compris pour les analyseurs non contextuels.

L'idée que nous avons mise en œuvre est la suivante. Lorsque l'on arrive à un point j où l'analyse est bloquée, on va rechercher un intervalle $p..q$ incluant j ($p \leq j \leq q$), qui permettrait de poursuivre l'analyse s'il recouvrait des terminaux différents de ceux qui sont effectivement présents. Autrement dit, on cherche, avec différentes stratégies de recherche possibles, un intervalle de la chaîne ou du DAG d'entrée pour lequel il existe (au moins) une correction permettant la poursuite de l'analyse. Les entiers p et q sont alors appelés *bornes* du rattrapage. On voit que la philosophie sous-tendant cette méthodologie est que l'on construit dans tous les cas des analyses complètes, qui respectent la grammaire, quitte à « corriger » la chaîne ou le DAG d'entrée (c'est-à-dire en réalité quitte à faire comme s'il était différent de ce qu'il est)[17].

Soit une chaîne ou un DAG source, d'état initial 1 et d'état final n. Supposons que l'analyse est bloquée dans l'analyseur Earley à la position j, c'est-à-dire dans la

16. Rappelons ici que l'algorithme d'Earley, comme l'algorithme GLR, a la propriété du préfixe valide. Ceci est encore vrai lorsque l'entrée est un DAG.

17. Il aurait été possible également de construire des analyses qui respectent la chaîne ou le DAG d'entrée, quitte à produire des forêts ne respectant pas la grammaire ou le lexique. Dans le cas de LFG, il aurait alors fallu définir le comportement des équations fonctionnelles sur les nœuds agrammaticaux de la forêt obtenue, ce qui n'est pas nécessairement satisfaisant. Un dernier choix consiste à toucher non pas à l'entrée (chaîne ou DAG) ou à la grammaire, mais à la sortie de l'analyseur non contextuel, c'est-à-dire à produire une structure qui n'est pas une forêt, mais un ensemble de forêts constituant des analyses partielles. On peut considérer cette dernière possibilité comme dégradant l'analyseur non contextuel en un analyseur de surface (*shallow parser*) en cas d'échec de l'analyse globale. Mais notre objectif étant de construire des analyses profondes, nous avons jugé préférable de construire dans tous les cas une analyse globale.

avant	(j,j), $(j,j+1)$, …, (j,n), $(j-1,j+1)$, …, $(j-1,n)$, …, $(1,n)$
arrière	(j,j), $(j-1,j)$, …, $(1,j)$, $(j-1,j+1)$, …, $(1,j+1)$, …, $(1,n)$
mixte-avant	(j,j), $(j,j+1)$, $(j-1,j)$, $(j,j+2)$, $(j-1,j+1)$, $(j-2,j)$, …, $(1,n)$
mixte-arrière	(j,j), $(j-1,j)$, $(j,j+1)$, $(j-2,j)$, $(j-1,j+1)$, $(j,j+2)$, …, $(1,n)$

Tableau 1. *Description des stratégies de rattrapage dans l'analyseur non contextuel. Les positions (ou états) initiale et finale sont notées 1 et n, et la position où l'analyse est bloquée est notée j*

table $T[j]$[18]. On appelle *signature* d'une tentative de rattrapage tout couple d'entiers (p,q) qui entoure ce point j ($1 \leq p \leq j \leq q \leq n$), p et q étant appelées *bornes* de cette tentative, comme indiqué précédemment. On appelle *stratégie de rattrapage* toute suite de signatures : l'application d'une stratégie consiste à essayer de se rattraper en essayant successivement les signatures de la stratégie, jusqu'à trouver un rattrapage ayant la bonne signature, ce qui met un terme à l'application de la stratégie. Signalons que si la signature $(1,n)$ est dans la stratégie, le succès est certain (voir ci-dessous).

L'écrivain de la grammaire peut choisir une stratégie parmi celles proposées par défaut dans SXLFG, mais peut également écrire sa propre stratégie. Comme nous allons le comprendre, une signature (p,q) telle que $q-j > j-p$ a tendance à favoriser la préservation de la partie du texte déja analysée au détriment de la correction de la portion du texte non encore examinée. On a l'inverse si $q-j < j-p$. Si deux signatures (p,q) et (p',q') sont telles que $q < q'$, la deuxième tentative va corriger davantage de symboles que la première. Si $p' < p$ elle réutilisera une plus faible quantité des analyses (partielles) déjà réalisées. Notons qu'une signature (p,q) peut être essayée même s'il n'existe pas de chemin dans le DAG entre les noeuds p et q.

Les quatre stratégies proposées par SXLFG sont qualifiées de *avant*, *arrière*, *mixte-avant* et *mixte-arrière* (cf. tableau 1). La stratégie avant privilégie les choix déjà réa-

18. Nous laissons de côté dans cette description le cas où l'entrée est un DAG et où l'on est bloqué simultanément à plusieurs endroits du DAG, qui sont non-comparables par la relation de précédence linéaire. Ce cas est cependant géré par SXLFG.

lisés par l'analyseur (et a donc tendance à sauter du texte source sans l'analyser). La stratégie arrière remet en cause les choix déjà réalisés par l'analyseur. Les stratégies mixtes visent à minimiser les portions du source qui ne seront finalement pas analysées, avec une priorité aux choix déjà effectués par l'analyseur dans la stratégie mixte-avant.

Pour une signature (p, q), le rattrapage procède comme suit. On examine tous les items $[A \rightarrow \alpha \bullet \beta, i]$ de la table $T[p]$. Pour chacun de ces items on va examiner le suffixe $\beta = X_1 \dots X_k$ de gauche à droite, jusqu'à trouver un X_h dont une chaîne dérivée commence par un symbole terminal r tel qu'il existe une transition sortante sur r depuis le noeud q du DAG. Si c'est le cas, l'item $[A \rightarrow \alpha X_1 \dots \bullet X_h \dots X_k, i]$ est ajouté à la table $T[q]$. Ceci garantit au moins une transition valide de $T[q]$ sur r. Sinon, on a atteint la position réduction de la production courante et on examine alors un autre item de la table $T[p]$[19].

Lorsque tous les items de $T[p]$ ont été examinés, si la table $T[q]$ est non vide, la signature (p, q) est déclarée *productive*, et le rattrapage est un succès. L'analyse normale reprend donc sur la table $T[q]$. On est sûr qu'aucune nouvelle erreur ne sera détectée dans $T[q]$. En revanche, une autre erreur peut très bien être détectée en $T[q+1]$ (ou dans les tables suivantes) et donc déclencher un nouveau rattrapage. Ce nouveau rattrapage peut bien sûr avoir une signature productive (p', q') qui englobe la borne précédente (p, q) ($p' \leq p$ et $q' > q$) et qui, par conséquent, invalide et remplace le rattrapage précédent. Si $p' > q$, les deux rattrapages coexistent. Le rattrapage avec la borne $q' > q$ n'est pas essayé si $p < p' \leq q$ (et produit donc un échec pour cette signature). La raison est double. Le cas $p < p' < q$ nous ferait essayer un rattrapage dans une table $T[q']$ inaccessible (car sautée par le rattrapage précédent). Le cas $p' = q$ correspond en fait à une *composition* de deux signatures (p, q) et (p', q') en une nouvelle signature (p, q') plus large, laquelle sera, si la stratégie le permet, essayée plus tard.

Nous avons indiqué que si la signature $(1, n)$ est dans la stratégie, le succès de cette stratégie est certain. Cela provient du fait que par construction de l'analyseur Earley, la table $T[1]$ contient l'item $[S' \rightarrow \bullet S\,\$, 0]$, où $\$$ désigne le marqueur de fin de chaîne, S est l'axiome de la grammaire, et S' le super-axiome (si w est une phrase, la production $S' \rightarrow S\,\$$ permet donc de reconnaître la chaîne $w\$$). Le mécanisme décrit précédemment permet de mettre l'item $[S' \rightarrow S \bullet \$, 0]$ dans la table $T[n]$ car $\$$ est par définition une transition sortante valide pour le noeud n. Cette situation est appelée *rattrapage trivial*.

19. Notons que nous n'effectuons pas la réduction précédente car cette opération nous ferait considérer une signature de la forme (i, q), avec $i \leq p$. Si $i < p$, on serait amené à tenter un rattrapage qui n'a pas la signature actuellement considérée pour tenter un rattrapage. De plus, si la stratégie de rattrapage est raisonnable, il y a de bonnes chances pour que l'on tente plus tard cette signature (i, q). Si en revanche $i = p$, la signature est la bonne, mais l'item $[B \rightarrow \gamma \bullet A\delta, m]$ est nécessairement un item de $T[p]$, et à ce titre a déjà été ou sera étudié par ailleurs.

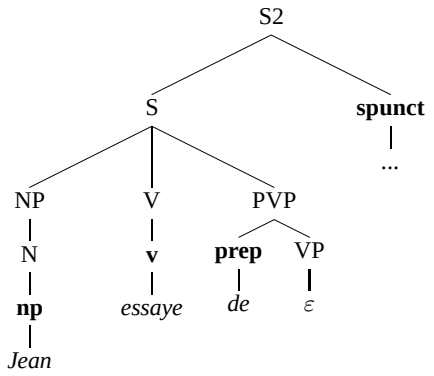


Figure 3. *Structure en constituants simplifiée pour la phrase incomplète « Jean essaye de... ». La dérivation de VP dans la chaîne vide est le résultat d'un rattrapage avant, et conduira à une structure fonctionnelle incomplète (pas de « pred » dans la sous-structure correspondant au nœud VP)*

Un exemple d'analyse produite à l'aide d'un rattrapage est montrée dans la figure 3 : dans ce cas, aucune transition sortante sur **spunct** n'est disponible après avoir reconnu **prep**. Un rattrapage avant est donc tenté, et réussit moyennant l'insertion d'un VP « vide » après le symbole **prep**, permettant ainsi la construction d'une analyse complète.

4.2. *Structures fonctionnelles incohérentes, incomplètes ou partielles*

Le calcul des structures fonctionnelles échoue si et seulement si aucune structure fonctionnelle cohérente et complète n'est trouvée. Ceci peut arriver parce que les contraintes d'unification spécifiées par les équations fonctionnelles n'ont pas pu être vérifiées, ou parce que les structures fonctionnelles résultantes sont incohérentes ou incomplètes. Sans entrer dans les détails, l'incohérence est principalement due à ce que des contraintes de sous-catégorisation ont échoué.

La première tentative de calcul des f-structures se fait en respectant toutes les contraintes de cohérence et de complétude, sauf lors du calcul des structures principales (c'est-à-dire associées à l'axiome). Ainsi, le résultat peut être soit un ensemble non vide de structures cohérentes et complètes, soit un ensemble non vide de structures incohérentes ou incomplètes dont toutes les sous-structures sont cohérentes et complètes, soit aucune structure.

Un échec de cette première tentative conduit à une seconde évaluation des f-structures sur la forêt partagée pendant laquelle les vérifications de cohérence et de complétude sont relâchées (un exemple en est donné figure 4). En cas de succès, on

$$\left[\begin{array}{l}
\text{pred = \textbf{'essayer <subj, de-vcomp>'}, v[2..3]} \\
\\
\text{subj} = \left[\begin{array}{l}\text{pred = \textbf{'Jean <(subj)>'}, np[1..2]} \\ \text{det} = + \\ \text{hum} = + \\ A_{ij} = \left\{R_9^{182}, R_{26}^{177}, R_{28}^{170}\right\}\end{array}\right]_{F68} \\
\\
\text{de-vcomp} = \left[\begin{array}{l}\text{pred = \textbf{'de <obj|vcomp>'}, prep[3..4]} \\ \text{vcomp} = \left[\begin{array}{l}\text{subj} = []_{F68} \\ A_{ij} = \left\{R_{84}^{162}\right\}_2\end{array}\right]_{F69} \\ \text{pcase = de} \\ A_{ij} = \{\}_2\end{array}\right]_{F70} \\
\\
\text{number = sg} \\
\text{person = 3} \\
\text{mode = indicative} \\
\text{tense = present} \\
A_{ij} = \left\{R_{33}^{130}, R_{48}^{119}, R_{49}^{134}\right\}
\end{array}\right]$$

Figure 4. *Structure fonctionnelle incomplète simplifiée pour la phrase incomplète « Jean essaye de... ». Les identifiants de sous-structures sont indiqués en indice (comme $F70$). Comme indiqué précédemment, une règle peut indiquer au analyseur de stocker un identifiant de la production instanciée courante dans le champ spécial A_{ij} de la structure associée au non-terminal de sa partie gauche. Les valeurs atomiques de la forme R_p^q représentent donc des productions instanciées, permettant ainsi de faire le lien entre sous-structures fonctionnelles et non-terminaux de la c-structure*

obtient des structures principales incohérentes ou incomplètes. Bien sûr, cette seconde tentative peut elle-même échouer. On cherche alors dans la forêt partagée un ensemble de *nœuds maximaux* qui ont des f-structures (éventuellement incomplètes ou incohérentes) et dont les nœuds pères n'ont pas de f-structure. Ils correspondent à des analyses partielles disjointes. Le processus de désambiguïsation globale présenté à la section 3.4 s'applique donc à tous les nœuds maximaux.

4.3. *Sur-segmentation des phrases inanalysables*

En dépit de toutes ces techniques de rattrapage, l'analyse échoue parfois, et aucune analyse n'est produite. Ceci peut arriver parce qu'un délai maximal donné en paramètre a expiré avant la fin du processus, ou parce que l'analyseur Earley a réalisé un rattrapage trivial (en raison d'une couverture insuffisante de la grammaire, ou parce que la phrase d'entrée est tout simplement trop loin d'être correcte : fautes de grammaire, phrases incomplètes, phrases trop bruitées…).

Pour cette raison, nous avons développé une sur-couche de SXLFG qui réalise une *sur-segmentation* des phrases inanalysables. L'idée est qu'il arrive fréquemment que des portions de la phrase d'entrée soient analysables comme phrases, bien que la phrase dans son ensemble ne le soit pas. Pour cette raison, nous redécoupons en *segments* les phrases inanalysables (segmentation de niveau 1) ; puis, si nécessaire, nous redécoupons les segments de niveau 1[20] (segmentation de niveau 2), et ainsi de suite avec 5 niveaux de segmentation, décrits ci-dessous[21]. Une telle technique suppose naturellement que la grammaire reconnaît aussi les *chunks* et les syntagmes (ce qui est linguistiquement justifié, pour reconnaître par exemple les phrases nominales) mais aussi les terminaux isolés (ce qui est moins pertinent linguistiquement). En un sens, c'est une généralisation de l'utilisation d'une grammaire de FRAGMENTS telle que décrite dans (Riezler *et al.*, 2002, Kaplan *et al.*, 2004).

Nos 5 niveaux de segmentation sont les suivants :

niveau 1 : segmentation sur les (quasi-)frontières de phrases : ceci inclut les frontières de phrases détectées par un segmenteur s'il n'a pas été utilisé précédemment (par exemple parce que le texte était déjà segmenté à l'aide d'un segmenteur différent), ou les quasi-frontières de phrases (« ; », identifiants d'éléments dans une liste, . . .),

niveau 2 : segmentation sur les ponctuations fortes (tirets isolés, points d'exclamation ou d'interrogation, guillemets, etc.),

niveau 3 : segmentation sur les ponctuations faibles (typiquement « , »),

niveau 4 : segmentation sur les coordinations (telles que « *et* » ou « *ou* »),

niveau 5 : segmentation sur les frontières de mots.

5. Résultats quantitatifs

Le but principal de cet article est de montrer que le générateur d'analyseurs SXLFG produit des analyseurs dont la robustesse et l'efficacité permettent l'analyse de gros corpus à l'aide de grammaires à large couverture écrite dans un formalisme syntaxique profond linguistiquement motivé. Pour ce faire, nous avons analysé 208 692 phrases de corpus journalistique brut (extrait du *Monde diplomatique*), soit environ 5 millions de tokens[22], à l'aide de l'analyseur issu d'une grammaire et d'un lexique à large cou-

20. Une phrase peut être segmentée en deux segments de niveau 1, le premier étant analysable. Dans ce cas, seul le second sera sur-segmenté en segments de niveau 2. Et seuls les segments de niveau 2 qui ne sont pas analysables seront sur-segmentés, et ainsi de suite.

21. Le dernier niveau de segmentation segmente la phrase d'entrée en terminaux isolés, afin de garantir que toute entrée est analysée, et en particulier pour ne pas abandonner l'analyse de phrases pour lesquelles des segments de niveau 1 ou 2 sont analysables, mais qui ont des portions qui ne sont « analysables » qu'au niveau 5.

22. Le nombre exact est de 5 508 467 transitions dans l'ensemble des DAG de mots produits par la phase de traitement pré-syntaxique, soit 1, 1 fois le nombre de tokens. On notera qu'il s'agit bien de DAG de *mots* et non de *lexèmes*, comme indiqué en 3.1 : *pomme de terre* n'induit que quatre transitions, l'une pour *pomme de terre*, et une pour chacun des trois mots *pomme*,

verture. Le temps d'analyse total pour ces 4 millions et demi de mots n'est que de moins de 12 heures.

À ce stade, il est nécessaire d'expliciter une distinction importante sur laquelle nous reviendrons plus bas. SXLFG n'est pas un analyseur, c'est un générateur d'analyseurs. Il prend en entrée une grammaire et un lexique et produit en sortie un analyseur. Par conséquent, les performances de cet analyseur produit dépendent à la fois des performances du générateur SXLFG et des caractéristiques de la grammaire[23]. Ceci signifie qu'évaluer SXLFG n'est pas trivial, dans la mesure où l'on ne peut évaluer que les analyseurs qu'il génère pour des grammaires particulières. Le but de cet article étant de montrer que les performances de SXLFG permettent d'analyser de gros corpus avec des grammaires LFG à large couverture, nous évaluerons donc SXLFG en montrant simultanément :

– que l'analyseur produit par SXLFG à partir d'une grammaire à large couverture est suffisamment efficace et robuste pour analyser plusieurs millions de mots en quelques heures,

– que les performances de l'analyseur produit sont très satisfaisantes, eu égard aux caractéristiques de la grammaire dont il est issu, et en particulier son ambiguïté et sa couverture.

En revanche, notre but n'est pas d'évaluer la grammaire elle-même (et les heuristiques de désambiguïsation qui l'accompagnent) ou le lexique. Nous ne les décrirons que pour permettre la mise en rapport de leurs caractéristiques avec les performances de l'analyeur produit par SXLFG.

5.1. *Grammaire, règles de désambiguïsation, lexique*

Pour évaluer SXLFG, nous avons utilisé une grammaire du français dont le développement s'est appuyé sur une grammaire LFG développée par L. Clément pour son système XLFG (Clément *et al.*, 2001)[24]. Parmi les phénomènes complexes couverts par la grammaire, on peut citer les coordinations sans ellipse, les juxtapositions (de phrases ou de groupes), les interrogatives (non clivées), les sujets inversés ou doubles (*Pierre dort-il ?*), tous les types de noyaux verbaux (y compris les clitiques,

de et *terre*. Mais le nom commun *terre* et la forme verbale homonyme ne sont pas distingués : c'est le lexeur, premier composant de l'analyseur, qui transforme ce DAG de mots en un DAG de lexèmes.

23. Dans le cas général, une grammaire très ambiguë donnée à un générateur d'analyseurs performant peut induire un analyseur peu efficace. Mais une grammaire très simple donnée à un générateur d'analyseurs rudimentaire donnera également un analyseur peu efficace.

24. Le projet initial XLFG remonte à 1996. Écrit en langage C, il comprend une interface graphique de développement de grammaires LFG et un analyseur qui repose sur un algorithme SLR non déterministe. Depuis le départ, cet analyseur est destiné plutôt à l'enseignement et à l'expérimentation de nouveaux concepts linguistiques, mais pas à une utilisation sur de gros corpus ou de très longues phrases.

les temps composés, le passif et la négation), les complétives (sous-catégorisées ou adjointes), les infinitives (y compris les verbes à montée et les différents types de verbes à contrôle) ou les relatives et les interrogatives indirectes (même enchâssées). En revanche, les comparatives, les clivées, les coordinations à ellipse ne sont pas couvertes (entre autres). D'autre part, nous avons pu constater que la grammaire squelette (cf. section 5.2) est excessivement ambiguë. Toutefois, notre expérience montre que toute grammaire à large couverture reposant sur une grammaire squelette (non contextuelle ou TAG, par exemple) est inévitablement très ambiguë[25]. À titre indicatif, cette grammaire comporte 253 règles, 911 équations fonctionnelles, 40 symboles terminaux et 109 symboles non terminaux, dont environ la moitié sont récursifs (la plus grande boucle de récursivité implique 30 symboles non terminaux).

En complément de la grammaire proprement dite, nous avons défini un ensemble d'heuristiques de désambiguïsation[26], dont le rôle premier est de spécifier des préférences au sein d'un ensemble d'analyses. Suivant sur ce point (Clément *et al.*, 2001), nous utilisons un ensemble de règles qui sont une adaptation et une extension des trois principes simples qu'ils décrivent et qui sont appliquées sur les f-structures, plutôt qu'un modèle stochastique[27]. Nos règles reposent sur des considérations linguistiques. Parmi elles, deux règles spéciales sont disponibles, qui éliminent les f-structures incohérentes et incomplètes soit dans tous les cas soit dans le seul cas où au moins une structure cohérente et complète existe pour le même non-terminal instancié (ces règles ne sont pas appliquées lors de la seconde tentative, si celle-ci est nécessaire). Comme expliqué précédemment, nous avons attribué à certains non-terminaux du squelette CFG, qui correspondent à des groupes « saturés » linguistiquement, une liste (ordonnée) partielle de ces heuristiques, chaque non-terminal concerné ayant sa propre liste.

25. Naturellement, l'ambiguïté d'une grammaire n'est pas la seule cause de la complexité de l'analyseur associé. Comme nous l'a signalé l'un des relecteurs, la taille de la grammaire (donnée particulièrement pertinente dans le cas de grammaires lexicalisées ou de grammaires générées automatiquement à partir de corpus arborés) et le nombre et les types de contraintes exprimées dans cette grammaire participent de façon importante à la complexité de l'analyseur associé (ainsi, dans le cas de LFG, l'utilisation excessive d'expressions régulières sur les chemins d'attributs peut être coûteuse). Par ailleurs, la grammaire utilisée dans les expériences décrites ici n'est évidemment pas la plus complexe possible, bien qu'elle soit à large couverture et qu'elle construise des structures pertinentes. Mais il n'est donc pas impossible que SXLFG, à partir d'une autre grammaire raisonnable et linguistiquement motivée, produise un analyseur aux performances décevantes.

26. Précisons bien ici que ces heuristiques ne sont pas définies dans SXLFG, mais sont fournies, avec la grammaire, en *entrée* à SXLFG.

27. Comme évoqué plus haut, un modèle stochastique pourrait cependant facilement être intégré, en définissant une règle qui utiliserait un tel modèle pour pondérer les f-structures (cf. par exemple (Miyao *et al.*, 2002)). Comme les autres règles, cette règle appliquée à un non-terminal éliminerait toutes les f-structures qui ne sont pas optimales au sens de cette règle, c'est-à-dire dont la pondération par le modèle stochastique ne serait pas maximale. Cependant, nos expériences montrent que des règles structurelles peuvent être suffisamment discriminantes pour permettre à l'analyseur d'être efficace, sans que soient nécessaires des données statistiques qui doivent être acquises sur des corpus annotés rares et coûteux, en particulier si la langue considérée n'est pas l'anglais.

Nos règles heuristiques, dans leur ordre d'application lorsque l'on effectue la désambiguïsation globale, c'est-à-dire à la racine de la forêt, sont les suivantes (on rappelle encore une fois que sur les non-terminaux internes sur lesquels on effectue une désambiguïsation interne, certaines règles peuvent ne pas être appliquées, ou bien dans un ordre différent) :

Règle 1 : *Éliminer les structures incohérentes et incomplètes si toutefois il y en a au moins une qui est cohérente et complète*.

Règle 2 : *Préférer les analyses qui maximisent la somme des poids des lexèmes utilisés* ; le lexique utilisé associe en effet à certaines entrées un poids ; parmi les lexèmes qui ont un poids plus élevé que la normale se trouvent les mots composés,

Règle 3 : *Préférer les analyses qui minimisent le nombre de groupes nominaux sans déterminant.*

Règle 4 : *Préférer les analyses maximisant le nombre d'arguments par rapport au nombre de modifieurs, et le nombre de relations auxilliaire-participe par rapport au nombre d'arguments* (ce calcul est effectué récursivement sur toutes les sous-structures, une sous-structure contribuant d'autant moins à la mesure globale qu'elle est enchâssée).

Règle 5 : *Préférer les analyses minimisant la distance entre les prédicats (verbaux ou non) et leurs arguments.* (même remarque).

Règle 6 : *Préférer les structures les plus profondes*[28].

Règle 7 : *Ordonner les structures suivant le mode de leur tête verbale* (on préfère récursivement les structures à l'indicatif à celles au subjonctif, et ainsi de suite).

Règle 8 : *Ordonner selon la catégorie des gouverneurs des adverbes.*

Règle 9 : *Choisir une analyse au hasard* (l'utilisation de cette règle permet de garantir que la sortie est une f-structure unique).

Le lexique que nous avons utilisé est la dernière version du L*efff* (Lexique de formes fléchies du français, (Sagot *et al.*, 2005b)), qui contient des informations morphosyntaxiques et syntaxiques pour plus de 500 000 entrées correspondant à environ 400 000 tokens différents (mots simples ou composants de mots composés).

Mais le but de cet article n'est pas de valider la grammaire et les règles de désambiguïsation, la grammaire ayant pour seul rôle de permettre l'évaluation en situation réelle des techniques d'analyse développées pour SXLFG.

5.2. *Résultats*

Comme indiqué précédemment, nous avons analysé près de 5 millions de mots de corpus journalistique brut, après l'avoir passé à travers la chaîne SXPipe de traitement pré-syntaxique (Sagot *et al.*, 2005a), obtenant ainsi des DAG comportant au total plus

28. On notera que cette règle est prévue pour être utilisée après la précédente, ce qui signifie que l'on a déjà éliminé toutes les analyses où les arguments sont en moyenne moins proches de leur prédicat. C'est la combinaison de cette règle et de la précédente qui a pour effet de sélectionner les constructions les plus « simples » structurellement.

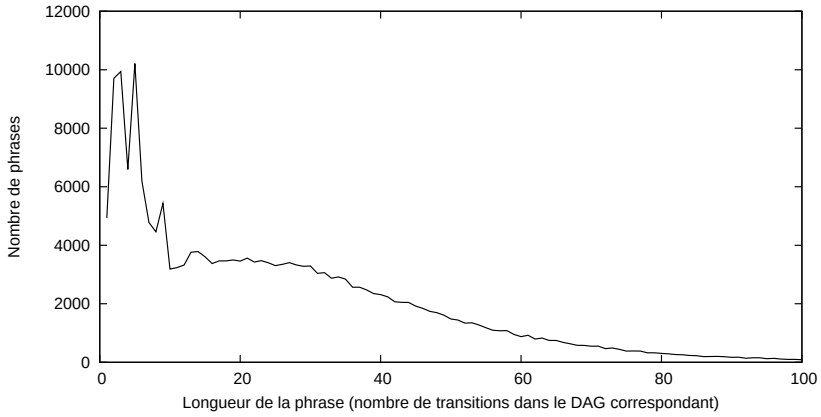


Figure 5. *Répartition des phrases du corpus de test en fonction de leur longueur*

de 5 millions et demi de transitions. La répartition des phrases en fonction de leur longueur est indiquée par la figure 5. Dans cette figure et celles qui suivent, l'abscisse est bornée pour ne montrer de résultats que sur des données statistiquement significatives, mais toutes les phrases sont analysées, la plus longue étant de longueur 384.

Cependant, pour évaluer les performances de notre générateur d'analyseurs SXLFG, il nous fallait isoler autant que possible l'influence de la grammaire dans nos résultats quantitatifs. En effet, et comme dit plus haut, l'efficacité du générateur d'analyseurs SXLFG est orthogonale à la qualité et à l'ambiguïté de la grammaire, qui est une donnée d'entrée pour SXLFG[29]. Au contraire, notre but est de développer un générateur d'analyseurs qui produise un analyseur aussi efficace et robuste que possible pour une grammaire donnée qu'on lui donne en entrée, y compris dans le cas où ladite grammaire est très ambiguë (au niveau du squelette CFG ou globalement) ou peu couvrante[30].

5.2.1. *Évaluation de l'analyseur non contextuel*

Pour cette raison, la figure 6 indique le niveau d'ambiguïté du squelette non contextuel en montrant le nombre médian d'analyses non contextuelles en fonction

29. Comme indiqué précédemment, ceci ne veut évidemment pas dire que les performances des analyseurs produits par SXLFG ne sont pas directement liées aux grammaires sous-jacentes.
30. Nous n'avons pas pu comparer les résultats quantitatifs de SXLFG par rapport à d'autres systèmes. Le site internet de l'environnement XLE donne toutefois quelques indications (voir http://www2.parc.com/istl/groups/nltt/xle/), qui sont difficilement interprétables et ne semblent pas extrêmement récentes. Notre analyseur à large couverture produit par SXLFG donne des résultats plus rapides de plusieurs ordres de grandeur, sans qu'il semble possible d'en tirer des conclusions précises.

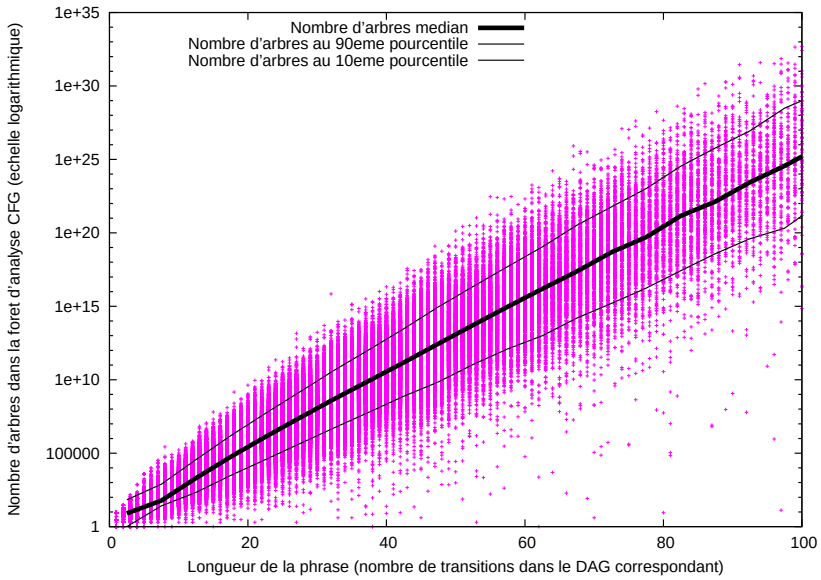


Figure 6. *Ambiguïté selon le squelette non contextuel en fonction de la longueur des phrases (les médianes sont calculées sur des classes de longueur allant de* $5i$ *à* $5(i+1)-1$ *et se voient attribuer une abscisse centrée (*$5(i+1/2)$*)*

du nombre de transitions dans le DAG de mots représentant une phrase donnée. Bien que le nombre d'arbres soit extrêmement élevé, la figure 7 montre l'efficacité de notre analyseur CFG[31] (le nombre maximum d'arbres atteint pour une phrase de notre corpus ne nécessitant pas de rattrapage d'erreur est de $8,8\,10^{45}$ pour une phrase de longueur 143 dont l'analyse non contextuelle prend seulement 0,05 s). De plus, les techniques de rattrapage d'erreur décrites dans la section 4.1 donnent de bons résultats dans la plupart des cas où le squelette CFG ne reconnaît pas la phrase d'entrée. Sur les 208 692 phrases, ceci ne concerne que 13 766 phrases ($6,60\,\%$).

5.2.2. *Évaluation du calcul des structures fonctionnelles*

Bien que le squelette non contextuel soit massivement ambigu, nos résultats montrent que nos techniques d'évaluation des f-structures sont efficaces. En effet, avec un délai maximum (*timeout*) de 1 seconde seulement, il suffit de 42 797 secondes, soit 11 heures et 53 minutes, pour analyser l'ensemble du corpus de 5 millions et demi de transitions, et seules $12,3\,\%$ des phrases atteignent ce délai sans avoir produit une

31. Ces expériences ont été réalisées sur une architecture AMD Athlon 2100+ (1,7 MHz) sous Linux Mandrake 10.2. Par ailleurs, la granularité de la mesure de temps d'exécution étant de 10 ms, nous avons ajouté 5 ms à tous les temps obtenus.

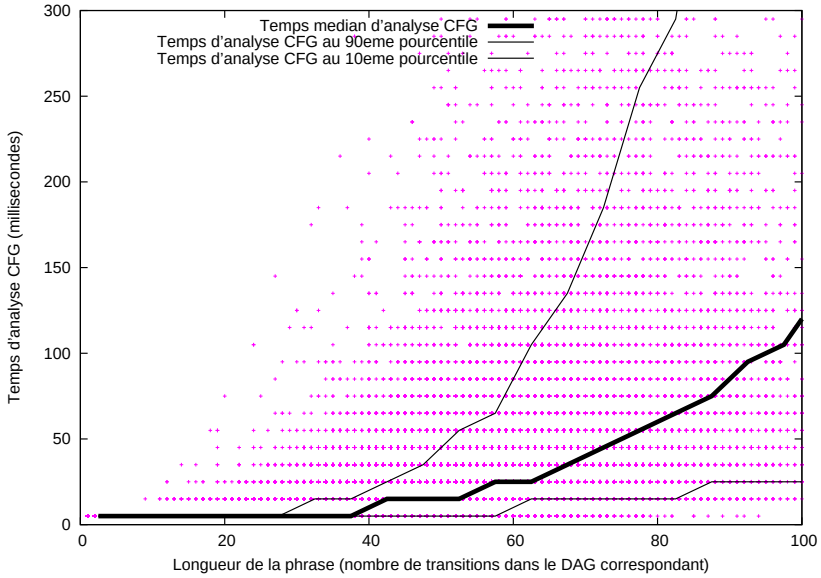


Figure 7. *Temps d'analyse par l'analyseur non contextuel (même remarque que pour la figure 6)*

analyse. Cela correspond à une vitesse moyenne de 128 transitions par seconde. On notera qu'un délai de 0,2 seconde fait descendre à un peu plus de 4 heures le temps total d'analyse, en ne faisant monter qu'à $24,3\ \%$ le pourcentage de phrases atteignant le délai. Enfin, un délai très agressif de 0,1 seconde permet d'analyser les 5 millions et demi de transitions en environ 2 heures et demie, « seulement » $31,5\ \%$ des phrases atteignant alors ce délai.

La couverture de la grammaire sur notre corpus avec désambiguïsation interne est de $58,5\ \%$, la couverture étant définie comme la proportion de phrases pour lesquelles une f-structure principale cohérente et complète est fournie par l'analyseur[32]. Ceci inclut les cas où la phrase était non-grammaticale pour l'analyseur non contextuel, mais où la forêt produite par le rattrapage d'erreur a permis de calculer une structure fonctionnelle principale cohérente et complète (cela concerne 1 782 phrases, c'est-à-dire $0,85\ \%$ de toutes les phrases et $12,9\ \%$ des phrases non-grammaticales pour le squelette CFG ; ceci montre que le rattrapage d'erreur dans l'analyseur non contextuel donne des résultats assez pertinents).

Puisque nous ne voulons pas que l'ambiguïté du squelette non contextuel influe sur notre évaluation de l'efficacité du calcul des f-structures par les analyseurs produits

32. Si on exclut les phrases qui atteignent le *timeout*, on obtient parmi les phrases analysées jusqu'au bout un taux de couverture de $66,8\ \%$.

Nombre total de phrases	208 692	
Reconnues par le squelette CFG	194 926	93,4 %
Analyse CFG avec rattrapage	13 766	6,6 %
f-structure cohérente et complète	122 188	58,5 %
f-structure incohérente ou incomplète	37 979	18,2 %
f-structures partielles	21 483	10,3 %
Aucune f-structure (rattrapage CFG trivial)	1 245	0,6 %
Erreur de l'analyseur (!)	31	0,01 %
Délai expiré (1 s)	25 759	12,3 %

Tableau 2. *Résultats de couverture sur un corpus journalistique brut de 5 millions et demi de mots environ (plus de 5,5 millions de transitions dans les DAG de mots correpondant)*

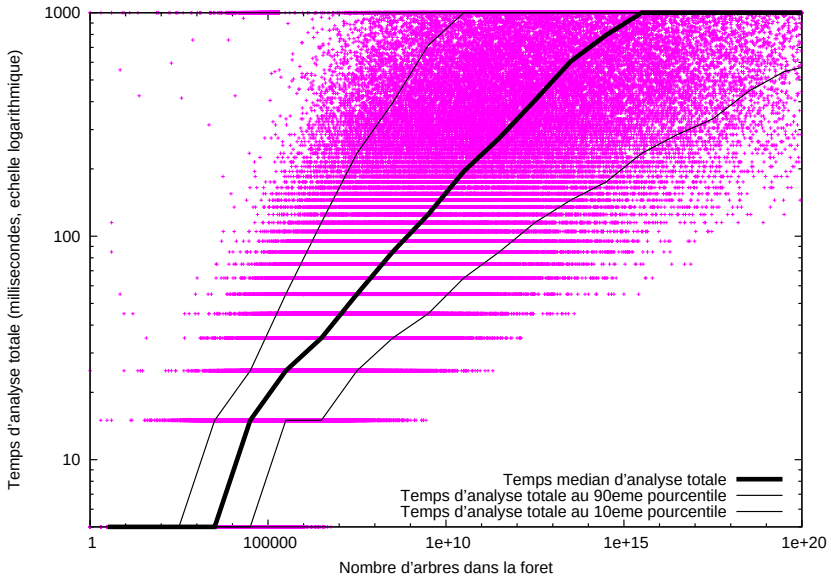


Figure 8. *Temps total d'analyse en fonction du nombre d'arbres dans la forêt produite par l'analyseur non contextuel (les médianes sont calculées sur des classes de longueur allant de* 10^i *à* $10^{i+1} - 1$ *et se voient attribuer une abscisse centrée (*$10^{i+1/2}$*)) sur une échelle logarithmique*

par SXLFG, la figure 8 représente le temps total d'analyse, y compris l'évaluation des structures fonctionnelles, en fonction du nombre d'arbres produits par l'analyseur non contextuel.

6. Conclusions

Cet article montre que des grammaires à large couverture reposant sur des mécanismes d'unification peuvent être utilisées pour définir des langues naturelles et qu'il est possible de les utiliser dans des analyseurs capables d'analyser du texte brut grâce à des techniques permettant de calculer des structures fonctionnelles sur une forêt partagée massivement ambiguë. En particulier, ces analyseurs peuvent être suffisamment efficaces, à l'image de notre analyseur SXLFG, pour permettre l'analyse de corpus volumineux (plusieurs millions ou dizaines de millions de mots) en quelques heures, malgré l'absence de filtrage statistique.

Nous montrons également qu'il est pertinent d'utiliser des techniques de rattrapage d'erreur à la fois au niveau des constituants et au niveau des structures fonctionnelles. De plus, les différentes techniques de robustesse que nous appliquons au niveau fonctionnel nous permettent de construire de l'information pertinente, même si elle n'est parfois que partielle. On notera que ces techniques de robustesse, qui n'altèrent pas significativement les performances globales de SXLFG, s'appliquent à la fois dans le cas où la grammaire est incomplète (couverture insuffisante) et où la phrase est agrammaticale, bien qu'il semble que les résultats soient meilleurs dans le second cas.

7. Bibliographie

Andrews A., Functional closure in LFG, Technical report, The Australian National University, 1990.

Boullier P., « Guided Earley Parsing », *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT'03)*, Nancy, France, p. 43-54, 2003.

Briffault X., Chibout K., Sabah G., Vapillon J., « An object-oriented linguistic engineering environment using LFG (Lexical-Functional Grammar) and CG (Conceptual Graphs) », *Proceedings of Computational Environments for Grammar Development and Linguistic Engineering, ACL'97 Workshop*, 1997.

Clément L., Kinyon A., « XLFG – an LFG parsing scheme for French », *Proceedings of LFG'01*, Hong Kong, 2001.

Kaplan R., « The formal Architecture of Lexical Functionnal Grammar », *Journal of Informations Science and Engineering*, 1989.

Kaplan R., Bresnan J., « Lexical-Functional grammar : a formal system for grammatical representation », *in* , J. Bresnan (ed.), *The Mental Representation of Grammatical Relations*, MIT Press, Cambridge, Massachusetts, États-Unis, p. 173-281, 1982.

Kaplan R. M., Maxwell J. T., Grammar Writer's Workbench, Version 2.0, Technical report, Xerox Corporation, 1994.

Kaplan R., Riezler S., King T., Maxwell J., Vasserman A., Crouch R., « Speed and Accuracy in Shallow and Deep Stochastic Parsing », *Proceedings of HLT/NAACL*, Boston, Massachusetts, États-Unis, 2004.

Kinyon A., « Are structural principles useful for automatic disambiguation ? », *Proceedings of in COGSCI'00*, Philadelphia, Pennsylvania, United States, 2000.

Miyao Y., Tsujii J., « Maximum entropy estimation for feature forests », *Proceedings of HLT*, San Diego, Californie, États-Unis, 2002.

Riezler S., King T., Kaplan R., Crouch R., Maxwell J., Johnson M., « Parsing the Wall Street Journal using a Lexical-Functional Grammar and Discriminative Estimation Techniques », *Proceedings of the Annual Meeting of the ACL*, University of Pennsylvania, 2002.

Sagot B., Boullier P., « From raw corpus to word lattices : robust pre-parsing processing », *Proceedings of L&TC 2005*, Poznań, Pologne, 2005a.

Sagot B., Clément L., Éric Villemonte de La Clergerie, Boullier P., « Vers un méta-lexique pour le français : architecture, acquisition, utilisation », *Journée ATALA sur l'interface lexique-grammaire*, March, 2005b. http://www.atala.org/article.php3?id_article=240.

ANNEXE POUR LE SERVICE FABRICATION

A FOURNIR PAR LES AUTEURS AVEC UN EXEMPLAIRE PAPIER
DE LEUR ARTICLE ET LE COPYRIGHT SIGNE PAR COURRIER
LE FICHIER PDF CORRESPONDANT SERA ENVOYE PAR E-MAIL

1. ARTICLE POUR LA REVUE :
   TAL. Volume 46 – n° 2/2005

2. AUTEURS :
   Pierre Boullier — Benoît Sagot

3. TITRE DE L'ARTICLE :
   Analyse syntaxique profonde à grande échelle : SXLFG

4. TITRE ABRÉGÉ POUR LE HAUT DE PAGE MOINS DE 40 SIGNES :
   Analyse profonde à grande échelle

5. DATE DE CETTE VERSION :
   19 juin 2006

6. COORDONNÉES DES AUTEURS :

   – adresse postale :

   INRIA Rocquencourt - Projet ATOLL
   Domaine de Voluceau
   Rocquencourt, B.P. 105
   F-78153 Le Chesnay cedex

   {pierre.boullier,benoit.sagot}@inria.fr

   – téléphone : 01 39 63 57 01

   – télécopie : 01 39 63 54 69

   – e-mail : {pierre.boullier,benoit.sagot}@inria.fr

7. LOGICIEL UTILISÉ POUR LA PRÉPARATION DE CET ARTICLE :
   LaTeX, avec le fichier de style `article-hermes.cls`,
   version 1.2 du 2004/07/24.

8. FORMULAIRE DE COPYRIGHT :
   Retourner le formulaire de copyright signé par les auteurs, téléchargé sur :
   `http://www.revuesonline.com`

SERVICE ÉDITORIAL – HERMES-LAVOISIER
14 rue de Provigny, F-94236 Cachan cedex
Tél : 01-47-40-67-67
E-mail : revues@lavoisier.fr
Serveur web : http://www.revuesonline.com