*TALN 2009 – Session posters, Senlis, 24–26 juin 2009*

# Grammaires d'erreur – correction grammaticale avec analyse profonde et proposition de corrections minimales

Lionel Clément[1] Kim Gerdes[2] Renaud Marlet[3]
[1] Université Bordeaux 1, LaBRI [2] ILPGA, LPP, Sorbonne Nouvelle [3] INRIA, LaBRI

**Résumé.** Nous présentons un système de correction grammatical ouvert, basé sur des analyses syntaxiques profondes. La spécification grammaticale est une grammaire hors-contexte équipée de structures de traits plates. Après une analyse en forêt partagée où les contraintes d'accord de traits sont relâchées, la détection d'erreur minimise globalement les corrections à effectuer et des phrases alternatives correctes sont automatiquement proposées.

**Abstract.** We present an open system for grammar checking, based on deep parsing. The grammatical specification is a contex-free grammar with flat feature structures. After a shared-forest analysis where feature agreement constraints are relaxed, error detection globally minimizes the number of fixes and alternate correct sentences are automatically proposed.

**Mots-clés :** Correcteur grammatical, analyse syntaxique, forêt partagée.

**Keywords :** Grammar checker, parsing, shared forest.

## 1 Introduction

La correction grammaticale est une des technologies du TAL les plus utilisées du grand public. Pourtant, elle a suscité comparativement peu de recherches, au moins en nombre de publications. Il y a sans doute plusieurs raisons à cela.

Tout d'abord, d'un point de vue pratique, les systèmes semblent relativement dépendants de la langue à corriger. En outre, une grande partie du travail de définition d'un correcteur grammatical consiste en un recueil patient d'erreurs idiosyncrasiques, qui peuvent dépendre de la langue maternelle du locuteur et de son niveau de connaissance. Cette tâche est difficilement automatisable faute de larges corpus d'erreurs disponibles. Enfin, l'utilité d'un tel système repose beaucoup sur son intégration dans un traitement de texte, et ce n'est que depuis la montée d'OpenOffice qu'un tel travail peut facilement être mis à la disposition du public. Il est désormais imaginable qu'une communauté se crée autour de la recherche en correction grammaticale, qui partage ouvertement ses ressources et résultats, comme dans d'autres domaines du TAL.

Il existe peut-être aussi des raisons plus profondes. La linguistique a pris du temps pour se définir en tant que science de la Langue réellement parlée, et un retour du débat sur la normativité n'est pas à l'ordre du jour, même si certaines recherches (en sociolinguistique, en psycholinguistique, en recherche sur le FLE ou encore en lexicologie) sur les dérivations de la norme peuvent avoir un intérêt indirect pour le développement d'un correcteur grammatical.

Cet article plaide pour une analyse profonde et complète de la phrase, par opposition aux grammaires superficielles et locales fréquemment utilisées, et pour une minimisation globale des erreurs à corriger. Sur la base d'un analyseur de grammaire de réécriture aménagé, nous montrons comment déterminer les erreurs et faire des propositions concrètes de meilleures corrections.

# 2 Problème général de la correction grammaticale

Un correcteur grammatical est un outil qui a deux fonctionnalités principales :
– Il signale les phrases ou fragments incorrects (probablement erronés) dans un document.
– Il propose des corrections, avec éventuellement une explication « linguistique » de l'erreur.
En pratique, on demande des propriétés additionnelles à un correcteur grammatical :
– qu'il soit rapide : la vérification doit s'effectuer « en temps réel » au cours de la frappe ;
– qu'il minimise le bruit : s'il y a trop de fausses altertes, les utilisateurs le débrancheront ;
– qu'il minimise le silence : il doit laisser passer peu d'erreurs.
Ce ne sont bien sûr là que les fonctionnalités et exigences principales.

## 2.1 Grammaire positive vs grammaire négative

Il y a deux moyens pour décider si une phrase est correcte : utiliser une grammaire *positive*, qui décrit toutes les phrases correctes, ou une grammaire *négative*, qui décrit les phrases incorrectes. Du fait des régularités de la langue, il est beaucoup plus facile d'écrire une grammaire positive. Une grammaire négative n'est cependant pas inutile. Les deux sont en fait complémentaires et permettent d'approcher les phrases correctes par un double encadrement. La grammaire positive peut notamment vérifier la structure et la cohérence générale de la phrase, et la grammaire négative se concentrer sur les erreurs courantes. Cette séparation des concepts est aussi motivée sur le plan de l'ingénierie linguistique : la grammaire positive est propre à la langue, indépendante de l'énonciateur, alors que la grammaire négative peut être choisie en fonction de l'utilisateur, par exemple pour signaler des faux amis grammaticaux propres à une autre langue. Contrairement à d'autres, nous n'incluons pas dans la grammaire négative la vérification des accords (traitée par la grammaire positive), mais des erreurs plus stylistiques, comme les barbarismes (par ex. l'anglicisme « faire une décision ») ou pléonasmes (par ex. « danger potentiel »).

Un locuteur natif commet moins d'erreurs qu'un apprenant, et elles ont une répartition différente. On peut penser qu'employer une forme marquée (par ex., en français, une marque de féminin) est plus significatif pour un locuteur natif que pour un apprenant, que la corriger est donc moins plausible, et ce d'autant plus que la marque est particulièrement visible ou audible (par ex. « actrice / acteur », par opposition à « jolie / joli »). Des gradations propres à l'utilisateur sont donc envisageables. En outre, les possibilités d'édition locale de la modalité textuelle, y compris le « copier / coller », peuvent conduire du fait de fautes d'inattention à des incohérences globales. Aucun type d'erreur ne peut donc être exclu, quel que soit l'utilisateur.

## 2.2 Grammaire de surface vs grammaire profonde

Une panoplie d'outils sont disponibles pour analyser une phrase afin de préparer un jugement de correction grammaticale : étiqueteur de parties du discours, chunker, expressions régulières, grammaires d'unification, etc. Plus une analyse est superficielle, plus elle est propice au bruit et au silence. Les grammaires de surface restent de ce fait cantonnées aux grammaires négatives, où la localité des règles minimise cet effet, sans l'annuler. Des attributs pauvres (par ex. de simples étiquettes) augmentent aussi le nombre de règles à écrire (Souque, 2008). Par ailleurs, l'éclatement en règles indépendantes est source de signalements multiples pour un même mot.

La plupart des correcteurs grammaticaux (par exemple Antidote, Cordial, Microsoft Office et Prolexis pour le français) sont des produits commerciaux dont le fonctionnement est opaque. Le type de données manipulées est parfois connu, mais pas les calculs faits pour corriger. LanguageTool (Naber, 2007) est actuellement un des rares correcteurs grammaticaux libres et ouverts ;

il est notamment interfacé à OpenOffice. L'infrastructure de correction grammaticale est basée sur un étiqueteur de parties du discours et des règles locales construites à l'aide d'expressions régulières. Plus de 1600 règles sont disponibles pour le français. Considérons par exemple :

(1) Le chien de mes voisins mordent.

(2) Les voitures qu'il a font du bruit.

Avec des règles locales, LanguageTools, à tort, reste silencieux sur (1) et signale deux erreurs dans (2). Microsoft Office (Fontenelle, 2006), qui semble faire des analyses plus profondes, signale bien l'erreur dans (1), mais demande à tort dans (2) à remplacer « font » par « fait ».

### 2.3 Nos objectifs

Pour répondre aux questions qui précèdent, nous nous donnons les objectifs suivants.

– Nous voulons un système ouvert, indépendant de la langue, où l'on peut spécifier facilement lexiques et grammaires, et les compléter incrémentalement par simple ajout de règles.
– Nous voulons une analyse profonde, capable de modéliser des phénomènes complexes comme des dépendances à longue distance, inaccessibles à de simples expressions régulières.
– Nous voulons pouvoir modéliser une grammaire positive et une grammaire négative, cette dernière s'appuyant sur les analyses profondes de l'analyseur de grammaire positive.
– Nous voulons automatiser la localisation des erreurs, leur hiérarchisation, et pour certaines les propositions de correction : le travail principal du linguiste doit être de modéliser la langue.
– Nous voulons un correcteur rapide, capable d'analyser un texte au cours de la frappe.
– Nous voulons un système librement disponible, facilement intégrable dans un éditeur.

Un correcteur performant pour une langue donnée nécessite une étude de domaine poussée et la spécification d'une large quantité de règles (Fontenelle, 2006). Tel n'est pas notre propos ici.

Ce que nous présentons dans cet article sont les principes et le moteur de notre système d'analyse de grammaire positive, indépendamment de la langue. Le français sert d'illustration. En bref, le processus de correction est le suivant. Une phrase est d'abord segmentée en un graphe acyclique orienté qui représente toutes les séquences possibles des formes lemmatisées des mots simples et composés, auxquelles s'ajoutent des lemmes supplémentaires (par ex. homophones) représentant des substitutions plausibles[1]. L'analyseur syntaxique construit ensuite une forêt partagée d'analyses en ignorant les phénomènes d'accord. Puis un parcours montant de la forêt attribue aux analyses alternatives des coûts minimums de modification de traits et d'utilisation de lemmes substitués, afin de satisfaire les accords. Une phrase qui a une analyse de coût minimum nul est correcte ; sinon, selon le coût de correction établi au niveau global, un parcours descendant détermine les flexions et substitutions qui reconstruisent une phrase correcte.

## 3 Correction grammaticale

Un correcteur grammatical localise des erreurs et propose des corrections vraisemblables, suggérant d'abord les plus plausibles. Le problème est donc de trouver des phrases correctes au *voisinage* d'une phrase incorrecte et de les classer par ordre de *proximité*.

### 3.1 Le principe du relâchement de contraintes

Les notions de voisinage et de proximité sont subjectives. Les circonscrire et les formaliser pour une langue demande un travail considérable d'expérimentation sur corpus. Nous faisons

[1] Le séquencement en « mots » pour le français recherche des caractères séparateurs. Nous travaillons sur une méthode pour traiter de façon plus générale et robuste la composition, l'agglutination, etc. pour des langues variées.

l'hypothèse que l'on peut modéliser un voisinage avec une grammaire positive dont on relâche certaines contraintes afin de construire davantage d'analyses modulo erreur, et que l'attribution d'un poids aux contraintes insatisfaites renseigne sur la proximité. La spécification du voisinage est en réalité inversée : on décrit alors les phrases incorrectes à proximité des phrases correctes.

Les grammaires intrinsèquement basées sur des contraintes, comme les grammaires de propriétés, sont particulièrement bien adaptées à cette vision du problème (Prost, 2008), mais elles posent encore des problèmes de performance et sont moins répandues. Le relâchement de contraintes a toutefois fait ses preuves pour la robustesse des analyseurs syntaxiques qui utilisent des systèmes de traits. Par exemple, (Vogel & Cooper, 1995) appliquent cette idée à HPSG, et (Fouvry, 2003) ajoute des pondérations sur l'importance des traits. Sur ce principe, nous avons opté pour un formalisme connu et éprouvé, qui permet une implémentation efficace. Notre spécification grammaticale est une grammaire non contextuelle dont les termes sont équipés d'une structure de traits. Pour un bon compromis efficacité-expressivité, nos structures de trait sont plates et construites sur des ensembles de valeurs finis. En voici un exemple simplifié.

```
gn[nb=N;gen=G;pers=3] -> det[nb=N;gen=G] sadj[nb=N;gen=G;type=anté]*
    nc[nb=N;gen=G] sadj[nb=N;gen=G;type=post]* gp[]? rel[nb=N;gen=G]? ;
```

Nous relâchons les contraintes d'accord liées aux traits. L'analyse de la structure hors-contexte reste en revanche rigide. La grammaire peut néanmoins être écrite pour accepter des phrases incorrectes, par exemple en rendant certains termes optionnels (évaluables par la grammaire négative). Nous conservons la totalité des ambiguïtés structurelles d'analyse : nous construisons une forêt partagée sur le squelette hors-contexte de la grammaire (Billot & Lang, 1989). Nous reposons pour cela sur une variante de l'algorithme d'Earley (Earley, 1970) capable de traiter directement l'étoile de Kleene et les termes optionnels, de complexité cubique en la longueur de la phrase dans le pire cas. Qui plus est, nous considérons qu'un axiome de la grammaire peut débuter à chaque mot d'une phrase. Nous pouvons ainsi construire toutes les analyses partielles d'une phrase non couverte par la grammaire et signaler les désaccords de traits locaux.

On peut diviser les erreurs grammaticales en *erreurs structurelles* (pas de structure de constituants) et *erreurs non structurelles* (l'unification échoue) (Bustamante & León, 1996). Pour les premières, on ne peut pas proposer de correction ; on ne peut que signaler la présence d'une erreur. Pour les secondes en revanche, on peut lister des propositions alternatives en jouant sur les valeurs de traits. Nous élargissons cette notion d'erreur non structurelle en autorisant des substitutions lemmatiques plausibles (par ex. des homonophones), qui permettent de proposer des corrections même si la phrase analysée n'a pas de structure de constitutants. Certaines erreurs structurelles, comme l'antéposition adjectivale, peuvent aussi être modélisées avec des traits.

## 3.2 Correction minimale

Uszkoreit, cité dans (Sågvall Hein, 1998), propose de distinguer 4 niveaux dans la correction grammaticale : (1) identification de segments possiblement erronés, (2) identification de contraintes possiblement violées, (3) identification de sources d'erreur possibles, et (4) construction et hiérarchisation d'alternatives de correction. C'est sur ce dernier point que nous mettons l'accent. Nous modélisons la plausibilité d'une correction en terme de *coût minimum*. Plusieurs notions de minimalité sont concevables. L'exemple suivant illustre la question de la localité.

(3) Les cheval blanc sont salissants.

(4) Le cheval blanc est salissant.

(5) Les chevaux blancs sont salissants.

Avec une décision locale, la meilleure correction de « les cheval blanc » dans (3) est un déterminant singulier ; puis pour la phrase, 3 mots au singulier et 2 au pluriel font préférer (4). Mais en fait, à un niveau global, on recense 5 termes à accorder en nombre ; 3 sont au pluriel, 2 au singulier. La correction globalement minimale en nombre de modifications est donc (5). Une décision locale est ainsi inexacte, et peut aussi conduire à de mauvais signalements en cascade.

Deux modèles de proximité permettent d'ordonner les corrections plausibles : l'un est basé sur le nombre de mots à corriger, l'autre sur le nombre de traits. Bien que souvent en accord, ces deux modèles ne sont pas comparables :

(6) C'est encore une histoire de cliente arrivée mécontent mais repartis satisfaits.
(7) C'est encore une histoire de client arrivé mécontent mais reparti satisfait.
(8) C'est encore une histoire de cliente arrivée mécontente mais repartie satisfaite.
(9) C'est encore une histoire de clients arrivés mécontents mais repartis satisfaits.

Dans (6), le syntagme nominal de « client » est plutôt au masculin, 3 votes contre 2 et plutôt au singulier, 3 votes contre 2 également. Cette minimisation du nombre de traits modifiés corrige 4 occurrences de traits et 4 mots (7). Mais deux alternatives sont plus économiques en nombre de mots corrigés : (8) et (9). Elles corrigent 5 occurrences de traits et mais seulement de 3 mots. Nous optons ici pour la minimisation du nombre de traits à corriger, qui nous semble « cognitivement » plus motivée, mais l'autre choix peut néanmoins être encodé dans notre proposition.

# 4 Correction lexicale

Voisinage et proximité jouent aussi au niveau du mot. Le lexique doit permettre de définir quelles formes peuvent en corriger d'autres, et à quel coût. Un premier type de correction correspond à des flexions alternatives d'un même lemme, par ex. « jolie » pour « joli ». Mais tout trait ne varie pas librement, par ex. rendre « crayon » féminin a un coût infini. Un deuxième type de correction correspond à une substitution plausible de lemmes, notamment des homophones : on / ont, est / ait / ai, à / a, etc. Comme les catégories peuvent ici varier, à la différence du cas précédent, ces corrections potentielles construisent des analyses alternatives supplémentaires. Cela augmente donc la combinatoire de l'analyse et il convient d'en bien mesurer l'usage.

La frontière entre grammaire positive et grammaire négative est ici ténue car on souhaite inclure au voisinage des phrases correctes (grammaire positive) un maximum de phrases incorrectes plausibles. Sans faire appel à une grammaire négative, on peut par exemple vouloir corriger un « que » erroné en « dont », soit en créant une classe lemmatique pour certains pronoms relatifs, que l'on fléchit avec un trait de cas, soit en les considérant explicitement comme substituables les uns aux autres. On peut faire de même pour les prépositions afin de corriger la rection verbale. Par ailleurs, le coût d'une correction peut dépendre de l'utilisateur, du type de lemme, et de ses formes (cf. §2.1). Le coût de substitution n'est donc pas symétrique, ni figé dans le lexique ; une partie doit pouvoir être calculée dynamiquement. Pour cela, nous définissons :
– un *lexique de flexions*, qui spécifie les lemmes, leur formes et les variations de leurs traits,
– un *lexique de substitutions*, qui spécifie des substitutions entre formes de lemmes différents,
– un *modèle de proximité* qui associe des coûts aux substitutions de traits ou de lemmes.

Notre lexique de flexions actuel a un format extensionnel :

```
# Forme   Lemme    Catégorie et traits
heureux   heureux  adj[gen=masc; nb=sing|plur]
portions  portion  nc[gen=fem!; nb=plur]
portions  porter   v[pers=1; nb=plur; mode=ind|subj; tps=imp]
```

Si plusieurs lemmes correspondent à une forme, on crée autant d'entrées. Pour chaque forme, nous indiquons le lemme correspondant, sa catégorie et ses valeurs de traits actuelles. L'opérateur « | » permet de spécifier un ensemble de valeurs. Par défaut, toutes les valeurs possibles d'un trait sont autorisées ; la marque « ! » interdit les valeurs autres que celle indiquées. Les traits et valeurs sont pour cela déclarés au préalable. Peu coûteuse et structurante pour le linguiste, cette déclaration permet aussi de signaler des incohérences dans le lexique ou la grammaire. Construire un tel lexique (pour une langue peu dotée) est en grande partie automatisable (Clément *et al.*, 2004). (De fait, notre lexique pour le français est déduit du Lefff.) Quant au lexique de substitutions, il liste les formes substituables les unes aux autres, par ex., « `on|ont` », « `a|à` ». Le lexeur introduit *systématiquement* ces formes commes alternatives avant l'analyse. (Notre lexique de substitutions du français provient des homophones de Lexique 3 (New, 2006).)

Notre intention est d'offrir des outils pour paramétrer facilement un modèle de coût, par exemple une distance de Levenshtein pour pondérer un coût par le degré de marquage des formes. Actuellement, c'est un programme ad hoc qui attribue un coût, selon un schéma fixe très simple. À toute flexion (c.-à-d. assignation d'une valeur de trait), est associé le coût de cette correction :
– Le coût de flexion est 1, quel que soit le type de trait, pour toute assignation à une valeur autre que celles indiquées (par ex. 1 pour attribuer fem à « heureux »).
– L'assignation d'un trait à une valeur interdite a un coût infini (par ex. fem pour « crayon »).
– Une substitution a un coût (arbitraire) de 1. Elle est modélisée par un trait implicite additionnel, noté « $ », et qui ne peut avoir qu'une valeur : son assignation est donc obligatoire.

On peut aussi coupler ici le correcteur orthographique au correcteur grammatical en incluant des substitutions proposées pour des mots hors lexique. Cela permet de traiter à un même niveau les corrections grammaticales et les flexions erronées comme « chevals » ou « faisez ». (Une simplification consiste à laisser l'utilisateur corriger d'abord l'orthographe, puis la grammaire.)

# 5 Signalement d'erreur

Supposons le texte segmenté en phrases[2]. Toute phrase est d'abord elle-même segmentée par un lexeur qui construit un graphe orienté acyclique (DAG) de lemmes alternatifs (dus aux ambiguïtés lexicales et aux substitutions plausibles), auxquels sont ajoutées des informations morpho-syntaxiques (notamment les catégories flexionnelles).

L'analyse syntaxique structurelle de ce DAG construit ensuite un *graphe et/ou* qui représente une forêt partagée d'arbres d'analyse (cf. fig. 1). Les traits sont totalement ignorés à ce stade ; seule est prise en compte et conservée la structure syntagmatique de la phrase (arbres de dérivation du squelette non contextuel de la grammaire). Le nombre d'analyses différentes dans le pire cas est exponentiel en la taille de la phrase. Les différents arbres d'analyse ne sont toutefois pas énumérés ici. C'est la forêt partagée qui les représente qui est construite, en un temps au plus cubique en la taille de la phrase. Le parseur s'accomode aussi d'analyses infinies, comme il peut s'en produire (généralement involontairement) avec des règles comme `np -> np pp?`. Il génère dans ce cas des cycles dans la forêt, qui peuvent aisément être éliminés.

Faute de place, nous ne détaillons pas ici l'algorithme complet de détermination des corrections de coût global minimum (soumis pour publication) ; nous en présentons juste l'idée intuitive, informellement. Nous n'explicitons pas non plus l'ensemble des constructions grammaticales disponibles ; nous nous concentrons sur la construction principale.

[2]Des heuristiques de découpage basées sur la ponctuation fournissent une segmentation raisonnable. Nous travaillons néanmoins sur une technique plus robuste qui exploite la capacité de notre analyseur à travailler en flux.

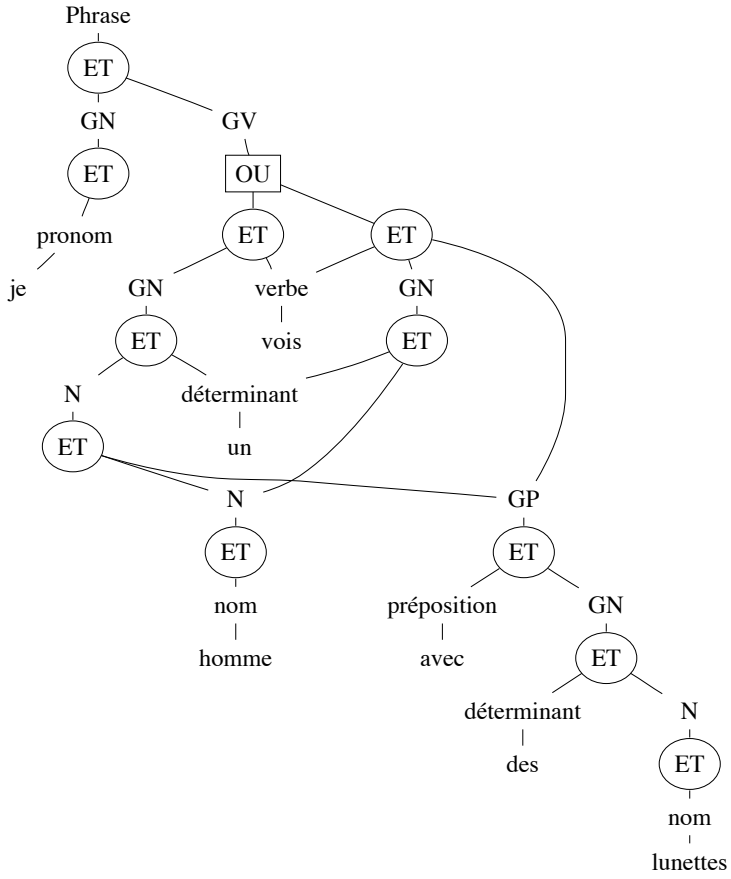


FIG. 1 – graphe *et-ou* de la forêt partagée pour « je vois un homme avec des lunettes »

## 5.1 Recherche d'erreur

La recherche d'erreur est un parcours de la forêt d'analyse pour déterminer les coûts de correction minimums des différentes analyses alternatives. À la racine, une forêt qui contient un arbre de coût de correction nul est considérée sans erreur — au risque d'être parfois silencieux du fait de possibilités de rattachement erronées, et en l'absence, pour le moment dans notre système, d'un jugement de plausibilité syntaxico-sémantique des analyses. Sinon, les coûts de correction des alternatives permettent d'ordonner les propositions de correction par ordre de plausibilité.

Une correction est représentée par une *assignation de traits*, c'est-à-dire un choix de valeur pour un ensemble de traits. Par exemple, pour un syntagme nominal : $(\mathsf{genre} \mapsto \mathsf{masc}, \mathsf{nombre} \mapsto \mathsf{pl})$.

Par ailleurs, un *coût d'assignation de traits* associe à toute valeur possible d'un trait un coût qui représente la plausibilité de modifier ce trait pour qu'il prenne la valeur en question dans une correction éventuelle (plus le coût est faible, plus la plausibilité est grande). Par exemple, pour un adjectif masculin singulier : $(\mathsf{genre} \mapsto (\mathsf{masc} \mapsto 0, \mathsf{fem} \mapsto 1), \mathsf{nombre} \mapsto (\mathsf{sg} \mapsto 0, \mathsf{pl} \mapsto 1))$. Un coût infini correspond à une affectation de trait impossible, comme d'imposer le trait féminin à nom commun exclusivement masculin (par ex. « crayon »).

Enfin, effectuer une assignation de traits, sur la base d'un certain coût d'assignation de traits, a un coût global qui est la somme des coûts des valeurs de traits assignées. Par exemple, effectuer l'assignation de traits $(\mathsf{genre} \mapsto \mathsf{masc}, \mathsf{nombre} \mapsto \mathsf{pl})$ sur la base du coût d'assignation de traits $(\mathsf{genre} \mapsto (\mathsf{masc} \mapsto 0, \mathsf{fem} \mapsto 1), \mathsf{nombre} \mapsto (\mathsf{sg} \mapsto 0, \mathsf{pl} \mapsto 1))$ a un coût global de $0 + 1 = 1$ : le coût de la modification du genre (ici nul) ajouté au coût de modification du nombre (ici 1).

Pour rechercher et ordonner les corrections, nous examinons les coûts d'assignation de traits des analyses alternatives. L'absence de monotonie des coûts minimums de correction et la recherche d'un optimum global obligent a priori à examiner toutes les alternatives. Dans le pire cas, elles peuvent être en nombre exponentiel en la longueur de la phrase. Dans cette sous-section, nous considérons que nous les examinons effectivement toutes ; nous verrons à la §5.2 comment réduire et contrôler cette combinatoire. Dans tous les cas, nous exploitons néanmoins la structure de partage syntaxique pour factoriser le calcul des coûts d'assignation de traits alternatifs.

Pour tout nœud du graphe *et/ou* de la forêt d'analyse, on calcule un ensemble de *coûts alternatifs d'assignation de traits* qui représentent l'ensemble des coûts minimums d'utilisation de ce noeud dans une alternative d'analyse, et pour les différentes affectations de traits. Le calcul opère « de bas en haut » (des feuilles vers les racines), de la manière suivante[3].

(a) Une feuille de la forêt est un item lexical, qui ne représente qu'une alternative. Son coût d'assignation de traits est défini par le modèle de coût pour les corrections lexicales (cf. §4).

(b) Un nœud-*ou* représente un ensemble de sous-arbres alternatifs (dont la tête est associée à un même non-terminal de la grammaire). L'ensemble des coûts d'assignation de traits pour un nœud-*ou* est la réunion de tous les coûts alternatifs de ses nœuds fils.

(c) Un noeud-*et* représente la tête d'un sous-arbre. Il correspond à l'application d'une règle de grammaire, règle de la forme $A \rightarrow B_1 B_2 \ldots B_n$, où chaque élément $B_i = b_i[\ldots]$ (et de même pour $A = a[\ldots]$) est un non-terminal $b_i$ (resp. $a$) associé à un ensemble d'équations de traits de la forme « trait = variable ». La tête du sous-arbre que représente un tel noeud-*et* est associée au non-terminal $a$ ; ses fils sont des nœuds-*ou* associés aux non-terminaux $b_i$.

L'idée intuitive est que le coût de correction d'un syntagme, c'est-à-dire le coût d'assignation de certains traits à des valeurs choisies, est la somme des coûts de correction de chaque élément de ce syntagme. En l'occurrence, le coût de correction du syntagme associé à $A$ est la somme des coûts de correction des $B_i$. En pratique, il faut tenir compte des points suivants :

1. À chaque $B_i$ est associé non pas un unique coût d'assignation de traits, mais un ensemble $K_i$ de tel coûts, correspondant à des analyses alternatives du nœud-*ou* associé. L'ensemble des coûts d'assignation de traits du noeud-*et* est basé sur l'ensemble des combinaisons possibles de choix d'un $n$-uplet de coûts dans $K_1 \times \cdots \times K_n$. (C'est la source de l'explosion combinatoire, que nous montrons comment contrôler en §5.2.)
2. Les coûts d'assignation de traits à additionner sont ceux pour lesquels un trait apparaît plusieurs fois avec une même variable dans la partie droite de la règle, comme par exemple le trait `nb` dans la règle de la section §3.1, associé à la variable `N`. Ce traitement remplace en quelque sorte l'opération d'unification d'une analyse syntaxique ordinaire : au lieu de vérifier que les valeurs associées aux différentes occurrences de `N` sont égales, on calcule pour chaque valeur possible (ici `sg` ou `pl`) le coût de l'affecter à `N` dans la partie droite de la règle. Et ce coût, qui peut être non nul pour une valeur de trait donnée, est la somme des coûts correspondants déjà calculés pour les éléments de cette partie droite.
3. On peut noter que les équations de traits associées à un $B_i$, de la forme « trait = variable », ne couvrent pas nécessairement tous les traits qui figurent dans l'ensemble des coûts d'assignation calculés pour $B_i$. Certains traits, en quelque sorte, ne sont donc pas propagés au niveau de la règle. Nous pouvons statuer sur eux localement, dès ce ni-

[3]Nous ne décrivons pas ici le traitement d'un certain nombre de constructions : filtrage (équivalent du $=_c$ de LFG), contrainte par des constantes (« trait = valeurs » en partie gauche ou droite de règle), idem avec variable (noté « trait = variable & valeurs) ». Le cas « trait = variable », que nous décrivons ici, est le cœur de l'algorithme.

veau. En revanche, on ne peut pas encore statuer sur les traits qui sont propagés, car des contraintes additionnelles peuvent ultérieurement en modifier les coûts d'assignation.

Pour garder la trace des traits non propagés sur lesquels on a statué, nous nous reposons sur le trait spécial $ (aussi utilisé dans le lexique de substitution, cf. §4), que l'on propage systématiquement jusqu'en haut de la forêt. Pour effectuer cette propagation systématique, le trait $ est *implicitement* considéré comme présent dans chaque non-terminal, associé à une même variable de trait $x_\$$. Autrement dit, toute règle définie comme $a[...] \rightarrow b_1[...] \dots b_n[...]$ représente en fait la règle $a[...; \$ = x_\$] \rightarrow b_1[...; \$ = x_\$] \dots b_n[...; \$ = x_\$]$. Statuer sur les traits non propagés consiste à intégrer (additionner) leur coût d'assignation minimum dans le trait spécial $. Ils pourront ainsi contribuer au coût global lorsque l'on statuera sur l'ensemble des traits, à la racine de la forêt.

4. Il faut également prendre en compte le fait que certaines variables peuvent n'apparaître qu'en partie droite de la règle, et non en partie gauche. En ce cas, on peut également statuer localement sur les traits auxquelles elles sont associées. Les coûts minimums d'assignation de traits correspondant sont alors aussi cumulés dans le trait spécial $.

Arrivé à la racine, on dispose d'un ensemble de coûts d'assignation de traits alternatifs, dont on calcule le minimum. Une fois choisie une assignation de valeur de traits de coût minimum, un parcours inverse de haut en bas (de la racine vers les feuilles) énumère les alternatives de correction effectives qui ont ce coût minimum, afin de les présenter à l'utilisateur. S'il y a de trop nombreuses alternatives de correction de coût minimum, on peut aussi en seuiller le nombre.

## 5.2 Réduction et contrôle de la combinatoire

Grâce à la forêt partagée, l'algorithme en §5.1 partage des calculs et structures de données. Il n'en énumère pas moins toutes les alternatives d'analyse, potentiellement en nombre exponentiel. Il est capital de réduire cette combinatoire, sans altérer l'optimum ou via des heuristiques.

On peut tout d'abord éliminer les coûts d'assignation de traits dont on sait qu'ils seront de toute façon plus mauvais que tout autre coût d'assignation de traits dans d'un même ensemble d'alternatives : soit parce que leurs coûts individuels (pour toute valeur de trait) sont uniformément plus élévés, soit parce que leur coût minimum (sur l'ensemble des traits) est supérieur au coût maximum de chaque autre alternative. La combinatoire est réduite sans modifier l'optimum.

Bien qu'expérimentalement efficace sur le type de phrases et de grammaire utilisées, cette réduction du nombre d'alternatives à considérer ne garantit toutefois pas un temps de correction polynomial. Pour se prémunir d'une explosion combinatoire en toute circonstance, on peut en outre (ou alternativement) appliquer une ou plusieurs des heuristiques suivantes :
– éliminer toute alternative dont le coût minimum est au dessus d'un certain seuil,
– éliminer toute alternative dont le coût maximum est au dessus d'un certain seuil,
– borner la taille des ensembles d'alternatives. (Dans le cas particulier où cette borne est 1, on retrouve comme cas particulier le calcul d'un optimum purement local.)

L'optimum n'est alors plus nécessairement global, mais la complexité devient polynomiale en la taille de la phrase. Seule la dernière heuristique garantit qu'une solution est toujours trouvée.

# 6 Conclusion

Nous avons posé le problème central de la correction grammaticale et argumenté en faveur d'une correction globale, basée sur l'accord des traits syntaxiques. Nous avons présenté pour cela un formalisme grammatical simple mais expressif, et appliqué un principe d'analyse en

structure rigide (mais ambiguë) avec relâchement des contraintes d'accord. Nous avons détaillé l'algorithme qui calcule une correction optimale, ainsi que des propriétés et heuristiques qui permettent de réduire et de contrôler sa combinatoire. Au final, des alternatives de correction de coûts minimal sont automatiquement proposées, en une seule passe sur la structure d'analyse. Le système et la grammaire actuellement implémentés corrigent par exemple (10) en (11) :

(10) Les enfants ont mangé ces cerise rouge qui étaient juteuses et sucrées et qu'ils ont vu que j'avais cueillis.

(11) Les enfants ont mangé ces cerises rouges qui étaient juteuses et sucrées et qu'ils ont vu que j'avais cueillies.

(12) Les enfants ont mangé cette cerise rouge qui était juteuse et sucrée et qu'ils ont vu que j'avais cueilli.

Cette correction minimale (3 traits) nécessite une analyse à la fois globale et profonde. La forêt est ici assez peu ambiguë, grâce à l'usage massif de l'étoile de Kleene et des termes optionnels. La correction, analyses syntaxiques partielles comprises, s'effectue en moins de 300 ms. Par comparaison, Microsoft Office, qui pourtant semble avoir une certaine vision globale, fait des erreurs en cascade et corrige (10) en (12), ce qui n'est pas minimal (5 traits modifiés) et même faux (mauvais accord du participe passé de la relative). LanguageTool reste silencieux.

Des expériences sont en cours pour mettre le système à l'épreuve, à la fois en termes de performance et de facilité d'écriture de grammaire. Nous voulons aussi expérimenter avec d'autres langues, et implémenter une intégration dans OpenOffice.

# Références


BILLOT S. & LANG B. (1989). The structure of shared forests in ambiguous parsing. In *27th annual meeting on Association for Computational Linguistics (ACL)*, p. 143–151 : ACL.

BUSTAMANTE F. R. & LEÓN F. S. (1996). Gramcheck : A grammar and style checker. In *COLING*, p. 175–181.

CLÉMENT L., SAGOT B. & LANG B. (2004). Morphology based automatic acquisition of large-coverage lexica. In *LREC '04*. Voir `http://alpage.inria.fr/~sagot/lefff.html`.

EARLEY J. (1970). An efficient context-free parsing algorithm. *Comm. of the ACM*, **13**(2).

FONTENELLE T. (2006). Les nouveaux outils de correction linguistique de Microsoft. In *Conférence sur le Traitement Automatique des Langues Naturelles (TALN)*, p. 3–19, Louvain.

FOUVRY F. (2003). Constraint relaxation with weighted feature structures. In *8th International Workshop on Parsing Technologies*.

NABER D. (2007). Integrated tools for spelling, style, and grammar checking. OpenOffice.org Conference, Barcelona. Outil disponible à l'URL `http://www.languagetool.org`.

NEW B. (2006). Lexique 3 : une nouvelle base de données lexicales. In *TALN '06*, p. 892–900.

PROST J.-P. (2008). *Modélisation de la gradience syntaxique par analyse relâchée à base de contraintes*. Thèse de doctorat, Université de Provence et Macquarie University.

SOUQUE A. (2008). Vers une nouvelle approche de la correction grammaticale automatique. In *Rencontre des Étudiants Chercheurs en Informatique pour le TAL*, p. 121–130, Avignon.

SÅGVALL HEIN A. (1998). A chart-based framework for grammar checking – initial studies. In *11th Nordic Conference in Computational Linguistic*, p. 68–80.

VOGEL C. & COOPER R. (1995). Robust chart parsing with mildly inconsistent feature structures. *Edinburh Working Papers in Cognitive Science : Nonclassical Feature Systems*, **10**.