Stage de recherche

Master II informatique

Université Paris-Est Marne-la-Vallée

De l'étiquetage morpho-syntaxique au super-chunking : Levée d'ambiguïtés à l'aide de méthodes hybrides et de ressources lexicales riches

Sigogne Anthony
laboratoire d'informatique de
l'institut Gaspard Monge, 2009
asigogne@etudiant.univ-mlv.fr

Maître de stage :
Matthieu Constant

Remerciements

Je tenais à remercier particulièrement Matthieu Constant, mon tuteur de stage, pour son aide précieuse pendant toute la durée du stage, ainsi que toute l'équipe d'informatique linguistique de l'Université Paris-Est Marne-la-Vallée.

Table des matières

Introduction :

L'étiquetage morpho-syntaxique et le chunking sont souvent considérés comme ayant atteints leurs limites de performance. Elles utilisent en général soit des approches statistiques soit des approches symboliques, mais peu utilisent des méthodes hybrides. Par ailleurs, ces processus ne tiennent pas ou très peu compte des mots composés, pourtant essentiels pour l'analyse de la langue. Dans ce stage, nous proposons d'étudier ces deux aspects encore trop peu considérés. La langue de travail sera le Français.

Nous souhaitons d'abord réaliser des expériences de levée d'ambiguïté morpho-syntaxique en combinant méthodes stochastiques supervisées et méthodes symboliques. Cette phase de levée d'ambiguïté se situe après une analyse lexicale à base de dictionnaires qui produit un automate représentant toutes les ambiguïtés. Le but est de trouver la meilleure analyse (soit le meilleur chemin) de l'automate. Cette première étude consistera à, tout d'abord, adapter et tester quelques méthodes stochastiques supervisées pour le français. En particulier les modèles de Markov cachés de premier et deuxième ordre (aussi écrit *HMM*). Ensuite on intégrera un désambiguiseur symbolique *Elag* [Laporte 1999] prenant en entrée un automate du texte et générant en sortie un autre automate élagué à l'aide d'un ensemble de règles. Un autre objectif est d'arriver à repérer et étiqueter les unités multi-mots telles que les mots composés plutôt que d'effectuer un étiquetage basique d'unités simples. Nous rappelons qu'un mot composé est un assemblage de mots dont le sens ne peut être déduit de l'addition des sens des mots qui le compose, par exemple les mots *cordon bleu* ou *coeur de pierre*. La dernière étape consistera à évaluer l'étiqueteur et en déduire si la prise en compte de ces mots composés améliore l'étiquetage classique.

Nous proposons ensuite d'étendre l'étude au super-chunking qui consiste à découper et étiqueter les phrases d'un texte en constituants non récursifs simples, ceci pouvant intégrer des unités multi-mots. Une première étape produit l'ensemble des analyses possibles sous la forme d'un automate acyclique à l'aide de lexiques et de grammaires. Une deuxième étape élague itérativement cet automate à l'aide de règles et d'heuristiques afin de trouver la bonne analyse. Il s'agit d'intégrer un module de levée d'ambiguïté stochatistique afin d'améliorer les résultats.

La première partie de ce rapport concerne l'analyse morpho-syntaxique d'un document textuel. Dans un premier lieu, nous détaillerons le principe général d'un étiqueteur morpho-syntaxique hybride qui combinera à la fois un module de désambiguisation symbolique et un module de désambiguisation statistique probabiliste utilisant comme base d'apprentissage un corpus annoté du Français. Nous verrons qu'il faudra améliorer lexicalement ce corpus qui souffre de certains défauts récurrents en particulier sur les unités multi-mots telles que les mots composés. Puis nous montrerons diverses expériences ménées sur la désambiguisation de l'automate ambigü du texte en s'aidant d'informations lexicales contenues dans les dictionnaires de mots composés. Ces ressources seront enrichies selon diverses méthodes statistiques et linguistiques, par exemple l'extraction automatique de noms composés d'un texte ou encore l'assignation automatique de traits morphologiques aux unités simples internes à ces mots composés. Pour finir, nous évaluerons notre étiqueteur d'après plusieurs critères comme la taille du jeu d'étiquettes, les types de mots composés pris en compte ou encore selon l'activation indépendante de chaque module de désambiguisation. Et enfin nous ferons une comparaison des résultats obtenus avec les performances d'autres étiqueteurs sur le même corpus et sur les mêmes critères.

La deuxième partie de ce rapport concerne le chunking, qui consiste à découper les phrases du texte en unités syntaxiques simples non récursives appelés chunks. Nous adapterons et évaluerons notre

algorithme de désambiguisation statistique probabiliste (basé sur les modèles de Markov cachés) en sortie d'un super-chunker[1] existant, Pom [Constant 2007].

Nous finirons ce rapport sur des perspectives futures et les innovations possibles à la fois sur notre étiqueteur morpho-syntaxique, sur le chunking ainsi que sur nos modules de désambiguisation d'automates.

1 Un super-chunker étend la notion de chunk en intégrant les unités multi-mots comme les noms composés ou les expressions figées.

I. Etat de l'art

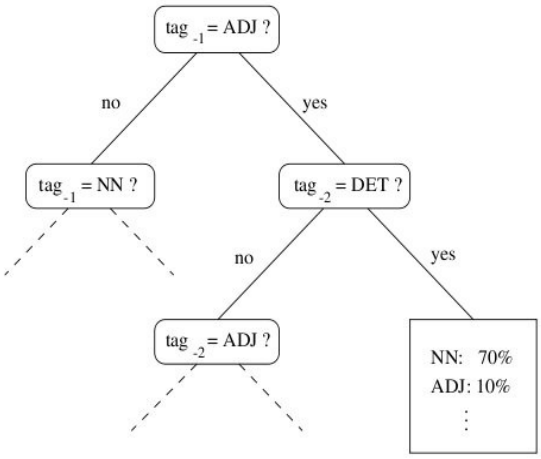
I.1. Analyse morpho-syntaxique

L'analyse morpho-syntaxique est une étape qui peut être considérée comme préliminaire à tout traitement linguistique plus poussé sur un texte, notamment l'analyse syntaxique. Elle consiste à affecter des étiquettes morpho-syntaxiques propres à chaque mot d'une phrase d'un texte (catégorie grammaticale, informations morphologiques comme le genre, le nombre...). L'étiquetage correct de la phrase *Max a mangé une pomme* est *Max.N a.V mangé.V une.DET pomme.N*. La principale difficulté de l'étiquetage morpho-syntaxique vient du fait que les mots de la langue sont ambigüs. C'est à dire que l'on peut affecter plusieurs étiquettes à un mot donné de la phrase. Le mot *grand* est un nom commun dans *Le grand mange* et est un adjectif dans *Le grand enfant*. Un étiqueteur morpho-syntaxique doit donc effectuer une phase de désambiguisation afin de sélectionner une séquence d'étiquettes possible pour la séquence de mots de la phrase, et si possible la séquence correcte. L'analyse morpho-syntaxique a été largement étudiée par le passé et est maintenant considérée comme un problème relativement résolu, les performances des étiqueteurs actuels étant très élevées (environ 97,50% de mots correctement étiquetés).
Plusieurs approches ont été proposées pour annoter automatiquement les mots d'un texte.

Approche statistique

Les étiqueteurs à base de statistiques subissent un regain d'intérêt depuis que leurs performances se sont révélées élevées (grâce notamment à l'arrivée de machines performantes). La plupart de ces étiqueteurs se basent sur les probabilités et plus particulièrement sur les modèles de Markov cachés de premier et deuxième degré (HMM). Plus une séquence d'étiquettes est probable, plus elle a de chance d'être choisie pour la suite de mots d'une phrase à étiqueter. La phrase *Max est grand* étiquetée *Max.N est.V grand.A* sera sûrement plus probable qu' étiquetée *Max.N est.N grand.N*. D'un point de vue linguistique, la phrase la plus probable n'est pas forcément celle qui est correcte. Les probabilités sont basées uniquement sur des observations et non sur des principes linguistiques de la langue. Les premiers étiqueteurs "performants" qui ont vu le jour dans les années 90 reprennent ce principe des probabilités et ont un schéma d'étiquetage classique des mots. C'est à dire dans le sens de lecture « left to right ». Pour étiqueter un mot donné d'une phrase, on va se servir des probabilités dites d'émission qui définissent la probabilité d'avoir un mot étant donné une étiquette ainsi que des probabilités dites de transitions qui prennent en compte les étiquettes des contextes gauches du mot. Par exemple, si l'on avait toujours la phrase *Max.N est.V grand*, *grand* pouvant être adjectif ou verbe, alors la probabilité d'avoir *grand.N* est calculée à partir de la probabilité d'avoir *grand.N* et d'avoir le contexte gauche *NV*. On fait de même pour *grand.A*. Au niveau de la phrase, on cherche donc à maximiser la probabilité d'avoir une séquence de mots étant donné une séquence d'étiquettes. L'apprentissage de ces probabilités est fait sur un corpus d'apprentissage annoté, la plupart du temps crées manuellement par des linguistes (pour le Français [Abeillé 2001]). Ce qui est un désavantage de cette approche car le coût de création de ce corpus est important étant donné sa taille.
Cependant, il faut bien distinguer le cas des mots connus et des mots inconnus dans un processus d'étiquetage utilisant les probabilités. Les mots inconnus sont par définition des mots qui apparaissent dans le texte à étiqueter mais qui sont absents du corpus d'apprentissage. La probabilité d'émission de ces mots est nulle quelque soit l'étiquette que l'on peut leur affecter. Nous devons donc appliquer un traitement particulier à ces mots. [Samuelsson 1993] a effectué une comparaison de diverses méthodes de traitement de ces mots et il semblerait que les meilleures méthodes soient

celles qui tiennent compte des suffixes des mots. Nous reprendrons dans notre étiqueteur une technique basée sur ces suffixes utilisée par [Thede 1998] qui consiste à choisir l'étiquette la plus probable pour le suffixe le plus long du mot inconnu.
Les étiqueteurs de [Thede 1998] et TnT [Banks 2000] suivent ce principe des HMM et se basent sur un modèle dit « trigramme », c'est à dire que les probabilités contextuelles sont calculées à partir d'un contexte gauche de deux étiquettes. Toujours dans les étiqueteurs basés sur les HMM, on peut citer TreeTagger [Schmid 1995] qui détermine les probabilités de transition à partir d'arbres de décisions. Les étiqueteurs classiques comme TnT utilisent des formules de probabilités particulières pour estimer les probabilités de transition et d'émission (basé sur le principe du maximum de vraisemblance). La formule de probabilité de transition peut donner des valeurs nulles ou proches de zéro. Ce qui veut dire deux choses, soit le trigramme (l'étiquette courante et le contexte gauche de deux étiquettes) n'existe pas, soit il est très rare. La distinction entre ces deux cas n'est pas évidente d'après les probabilités de transition obtenues. L'utilisation d'un arbre binaire de décision comme dans TreeTagger permet de régler partiellement ce problème. La probabilité d'un trigramme est déterminée en suivant le chemin à travers l'arbre de décision jusqu'à atteindre une feuille (où sont situées les probabilités de transition). D'après la figure ci-contre, si l'on recherche la probabilité d'avoir un nom précédé par le bigramme contextuel déterminant puis adjectif, *P(N/Det,Adj)*, alors il suffit tout d'abord de partir du noeud ADJ et de suivre le *yes-path* jusqu'au noeud DET. La réponse est encore oui car DET est l'ancêtre du mot, on continue donc jusqu'à la feuille qui contient les probabilités voulues.



Les performances de tous ces étiqueteurs sont généralement équivalentes, autour de 96,50% sur un corpus Anglais comme le Penn TreeBank.

A partir des années 2000, d'autres types d'étiqueteurs plus évolués, utilisant des processus plus lourds, font leur apparition. Parmi les plus connus et les plus performants on peut citer SVMTOOL [Gimenez 2003], Stanford Tagger [Toutanova 2003] et [Shen 2007]. La particularité de ces étiqueteurs est qu'ils intègrent une méthode dite « rich features », traits riches en Français. Lorsqu'ils étiquetent les mots d'une phrase, ils prennent en compte plus d'informations qu'un étiqueteur classique comme TnT. Ces derniers calculent les probabilités uniquement grâce au mot et à l'étiquette possible, ici on va se servir d'informations telles que les suffixes ou les préfixes du mot afin d'améliorer la finesse de l'étiquetage. Le plus grand changement vient du calcul des probabilités de transition. Le schéma classique d'étiquetage est « left to right », on se sert uniquement du contexte gauche, le passé du mot (en général deux étiquettes), pour étiqueter ce mot, or ce nouveau type d'étiqueteur va utiliser le futur du mot, le contexte droit, afin de calculer ces probabilités. Le schéma est dit « left to right to left ». Toutanova propose dans le Stanford Tagger une technique permettant d'étiqueter les mots en tenant compte du futur. L'idée est de constuire un graphe de dépendance cyclique, chaque noeud du graphe étant un mot avec ses étiquettes possibles. On peut donc connaître les contextes gauches et droits de chaque mot et calculer les probabilités locales à chaque mot. Quant à Shen, il propose une méthode différente d'étiquetage bi-directionnelle. L'idée est de mettre des hypothèses sur chaque mot de la phrase. Ces hypothèses, représentées par des probabilités, permettent de savoir quels sont les mots à étiqueter en priorité et quelles sont les actions possibles si ces mots sont étiquetés. Évidemment plus le mot permet de résoudre rapidement des ambiguités, plus il sera prioritaire. Par exemple, dans la phrase suivante *le livre de la bibliothèque*, le mot *de* pouvant être préposition ou déterminant, il serait utile de trouver les

étiquettes des mots *la* et *bibliothèque* avant de trouver celle de *de*. Pour finir, l'étiqueteur SVMTOOL a adapté la technologie SVM (« Support Vector Machine ») pour le domaine du traitement automatique des langues. C'est un classifieur linéaire qui permet de résoudre des problèmes de discriminations, et donc dans notre cas pouvoir discriminer les étiquettes assignables aux mots. Les performances de ces étiqueteurs sont largement meilleurs que les étiqueteurs classiques avec environ 97,30% en moyenne de mots correctement étiquetés.

Une autre classe d'étiqueteur statistique est basé sur les modèles de règles comme [Brill 1995]. Le principe de cet étiqueteur est de créer automatiquement des règles de transformation afin de corriger des erreurs d'étiquetage provoquées sur les étiquettes des mots d'un texte. Une règle de transformation est composée d'une partie « lexicale » qui indique la modification, la transformation, d'étiquette à effectuer et une partie « contextuelle », ou condition, qui indique dans quelles conditions effectuer cette transformation. On pourrait avoir une règle changeant un nom en déterminant si le mot concerné est *le* et le contexte droit du mot est un nom. Par exemple si l'on avait la phrase *le.N mot.N* alors d'après la règle de transformation précédente, on obtient l'étiquetage correct *le.Det mot.N*. On pourra remarquer que la partie contextuelle est bi-directionnelle, c'est à dire que l'on peut indiquer des contextes gauches et droits d'un mot à la manière des étiqueteurs précédents. Ces règles sont apprises de la façon suivante, on dispose d'un corpus de référence annoté et du même corpus brut. L'initialisation consiste à étiqueter basiquement ce corpus par une méthode aléatoire ou par un autre étiqueteur. Puis on compare les différences d'étiquetage entre le corpus étiqueté et le corpus de référence. Avec ces différences on crée des règles de transformation qui permettent de corriger les futures erreurs fréquentes ou non pour faire ressembler au maximum l'étiquetage à la vérité. Le processus est répété plusieurs fois en prenant en compte les règles créées aux étapes précédentes jusqu'à ce que le nombre d'erreurs d'étiquetage global soit inférieure à un certain palier fixé manuellement. L'ensemble des règles final est celui obtenu à la dernière itération de l'algorithme. Ces dernières sont classées selon leur nombre d'erreurs corrigées (la première étant celle qui corrige le plus d'erreurs et la dernière celle qui en corrige le moins) et sont donc appliquées dans cet ordre pour corriger un maximum d'étiquettes. Le cas des mots inconnus est sensiblement identique à ceux cités précédemment, c'est à dire qu'il utilise un algorithme à base de suffixes. Cet étiqueteur donne au final des résultats équivalents à la classe d'étiqueteur contenant TreeTagger et TnT avec environ 96,36% de mots correctement étiquetés sur le corpus Penn TreeBank pour l'Anglais. Nous reprendrons le principe général de cet étiqueteur dans le module de désambiguisation partiel *Learning Errors* utilisé dans notre étiqueteur morpho-syntaxique et détaillé au cours de ce rapport.

Approche symbolique

Une deuxième approche pour l'étiquetage est dite « symbolique » comme ELAG [Laporte 1999]. L'idée générale est relativement équivalente à l'étiqueteur de Brill, c'est à dire que l'on va construire des règles lexicales et contextuelles de désambiguisation. La différence vient du fait que ces règles sont faîtes manuellement, la plupart du temps par des linguistes. L'avantage de cette approche est donc d'avoir des règles lisibles, modifiables et qui collent aux phénomènes linguistiques d'une langue particulière. Une règle pourrait nous dire que le mot *si* est un *adverbe* si et seulement si il a un contexte droit *adverbe, adjectif, participe passé* ou *ponctuation*. Si tel n'est pas le cas alors l'ambiguité *si.ADV* est supprimée. Le désavantage de cette approche est le même que pour les approches statistiques de par le fait qu'il faut construire l'ensemble des règles manuellement et sur un temps plus long que pour un corpus (qui lui est directement utilisable en fin de création).

Approche hybride

L'approche utilisée par notre étiqueteur est une approche hybride, c'est à dire que l'on va combiner les deux types d'approche statistique et symbolique. Dans le cadre de l'approche statistique, nous avons adapté un algorithme de désambiguisation statistique probabiliste basé sur les modèles de Markov cachés de deuxième degré basique à la manière de TnT. Pour l'approche symbolique nous avons utilisé un désambiguiseur symbolique ELAG. L'idée est de comparer cet étiqueteur hybride aux différents étiqueteurs précédents utilisant une unique approche. De plus ces derniers utilisent en général un lexique d'étiquettes qui provient du corpus d'apprentissage, or nous disposons, au laboratoire IGM de Paris-Est Marne-la-Vallée, de ressources lexicales riches écrites par des linguistes [Courtois 1990]. Ces lexiques sont des dictionnaires de mots simples et de mots composés. Nous observerons si l'utilisation de telles ressources à la plaçe du lexique du corpus permet d'améliorer les performances de l'étiquetage. Et enfin, nous essayerons d'intégrer les mots composés, issus de ces dictionnaires, au processus d'étiquetage plutôt que de traiter uniquement avec des mots simples comme le font la plupart des étiqueteurs cités précédemment. Nous favoriserons les ambiguités étiquetées par des mots composés. Par exemple le nom composé *journal télévisé* aura plus de poids que les mots simples *journal* et *télévisé*.

I.2. Extension au chunking

Le chunking est une étape plus complexe linguistiquement que l'analyse morpho-syntaxique. Elle consiste à découper la phrase en constituants simples non récursifs appelés chunks. L'analyse de la phrase *Max a mangé une pomme* nous donne les chunks suivants *<Max.XN> <a mangé.XV> <une pomme.XN>*. La principale particularité des chunks est son élément dit "tête" qui correspondant au mot central du chunk. La tête de *<une pomme.XN>* est le mot *pomme*. Le chunking se différencie d'une analyse syntaxique complète par le fait qu'on ne cherche pas à extraire tout l'arbre syntaxique de la phrase mais seulement celle de plus bas niveau qui consiste à trouver la fonction syntaxique de chaque mot. Le problème principal du chunking est équivalent à celui de l'analyse morpho-syntaxique, c'est à dire que l'analyse en chunks est ambigüe. On peut avoir plusieurs possibilités de découpage et d'assignation de catégories grammaticales aux mots. Plusieurs approches ont été étudiées par le passé.

Approche statistique

Le plus simple chunker de cette catégorie est celui défini par [Ross 1975] qui consiste à délimiter les chunks en s'aidant des "stop words", c'est à dire les mots fonctionnels comme les prépositions ou les déterminants. [Bourigault 1992] utilise cette technique pour délimiter les chunks nominaux pour le Français. Le principe est de définir les "chinks" comme des mots qui ne peuvent apparaître dans un chunk nominal, par exemple les catégories grammaticales verbe, déterminant, conjonctions, ... avec certaines exceptions comme les mots *de*, *de la* et *a*. Les chunks sont définis comme les bouts de texte contenus entre les chinks. Par exemple la phrase *un [traitement de texte] est installé sur le [disque dur de la station de travail]*. Les approches statistiques à base de probabilités suivantes sont une

extension de l'analyse morpho-syntaxique définie précédemment. Une technique simple a été créée par [Church 1988]. Il utilise une technique stochastique, c'est à dire qu'il a construit un chunker délimiteur de chunks nominaux qui prend en entrée la sortie d'un étiqueteur morpho-syntaxique basé sur les modèles de Markov cachés. Son chunker insère des parenthèses ouvrantes et fermantes pour délimiter les chunks nominaux grâce aux étiquettes morpho-syntaxique assignées aux mots. Elles sont placées selon les probabilités, lexicales et contextuelles, d'avoir les séquences de mots et leur étiquette associée. Ces probabilités sont apprises sur un corpus arboré annoté manuellement. Des étiqueteurs comme [Skut 1997] et [Molina 2002] se basent sur cette technique mais en le généralisant pour tous types de chunks. D'autres étiqueteurs se basent sur des grammaires de dépendances. Les chunkers [Koskenniemi 1992] et ENGCG de [Voutilainen 1995] appliquent une méthode dite "reductionniste". On démarre avec un automate acyclique contenant toutes les analyses en chunks possibles et on applique une grammaire de règles contraignantes qui va permettre de réduire le nombre d'analyses. [Joshi 1996] utilise des grammaires lexicales d'arbre adjoints. Chaque arbre élémentaire de dérivation est associé à un mot et donc ils représentent les ambiguités syntaxiques possibles de ces mots. L'idée est d'effectuer des adjonctions et des substitutions de ces arbres pour créer un arbre de dépendances entre les mots, et donc la segmentation en chunks. Ces deux chunkers finalisent le processus de chunking par une linéarisation statistique faite par les HMM à la manière de Church. Le chunker de [Aït-Mokhtar 1997] est hybride dans le sens où il intègre à la fois une approche réductionniste et une approche par arbres de dérivation.

Approche symbolique

L'approche symbolique est quelque peu différente de l'approche statistique. La plupart utilisent le principe de cascade de transducteurs défini dans [Abney 1996], [Koskenniemi 1990] et [Koskenniemi 1992]. Il consiste à appliquer successivement des grammaires, des transducteurs sur l'automate lexicale ambigü qui vont reconnaître, pour chacun d'eux, un type de chunk particulier. L'entrée de chaque transducteur est constituée par la sortie du transducteur précédent. L'idée est donc d'arriver au niveau zéro des transducteurs, c'est à dire les axiomes. Cette technique est réutilisée dans les chunkers Pom [Constant 2007] et Macaon [Nasr 2007]. Contrairement à des chunkers statistiques s'aidant des grammaires de dépendances apprises sur des corpus, on va ici créer ces grammaires manuellement, la plupart du temps par des linguistes.
Par exemple, la grammaire suivante permet de reconnaître les chunks nominaux, verbaux et prépositionnels dans une phrase.

1: NP → D? A* N+ | Pron
   VP → Md Vb | Vz | Hz Vbn | Bz Vbn | Bz Vbg
2: PP → P NP
3: SV → NP VP
4: S → (Adv|PP)? SV NP? (Adv|PP)*

Approche hybride

De même que pour l'analyse morpho-syntaxique et pour reprendre les idées de celle-ci, nous avons décidé d'utiliser une approche hybride dans le cadre du chunking. Pour l'approche

symbolique, nous avons pris un chunker existant Pom [Constant 2007] qui applique une technique classique se basant sur une cascade de transducteurs pour la segmentation en chunks. Il applique différentes heuristiques et règles contextuelles permettant d'élaguer l'automate ambigü du texte (les analyses en chunks), et il termine sur une linéarisation basique faite par le hasard. L'idée est de greffer en sortie de ce chunker, l'algorithme de linéarisation statistique probabiliste vu lors de l'étiquetage morpho-syntaxique qui se base sur les modèles de Markov cachés (HMM). Les probabilités seront apprises sur un corpus annoté manuellement. Nous observerons si les performances du chunker avec cette linéarisation sont meilleures qu'avec une utilisation seule de Pom.
De plus Pom est un super-chunker, ce qui veut dire qu'il prend en compte les unités multi-mots (mots composés, expressions figées, ...) lors de la segmentation en chunks et donc durant la phase d'analyse lexicale. Ces unités multi-mots sont contenus dans des dictionnaires écrits par des linguistes. Peu d'étiqueteurs par le passé font l'usage de ces unités multi-mots [Nivre 2004], ou se restreignent à l'utilisation des mots composés qui sont plus stables et moins nombreux. Nous prendrons donc en compte également ces unités multi-mots lors des évaluations.

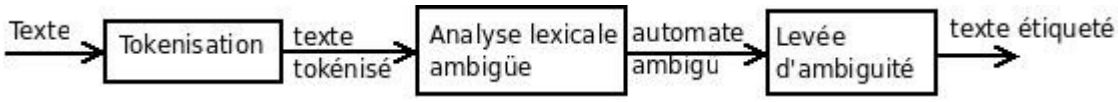
Méthodes d'évaluation

Nous n'avons pas parlé jusqu'à maintenant du mode de fonctionnement des évaluations d'un chunker. Contrairement à l'analyse morpho-syntaxique où l'évaluation est aisée car il suffit de valider le mot et son étiquette, le chunking présente deux types de difficulté. Tout d'abord au niveau de la segmentation en chunks, il peut être différent selon le système utilisé, de même pour les étiquettes grammaticales qui peuvent ne pas avoir d'équivalents. Ensuite le deuxième problème vient de la spécialisation du texte à analyser, plus celui-ci est spécifique à un domaine particulier, plus le chunker éprouvera des difficultés à analyser ce texte (par exemple le domaine juridique). Ceci est causé souvent par les mots simples ou les unités multi-mots qui ne seraient pas présents dans les dictionnaires utilisés lors de l'analyse lexicale ambigüe. Nous avons donc pris pour corpus d'évaluation un corpus basé sur le même schéma de découpage et d'étiquetage que Pom et ainsi avoir des résultats proches de la réalité.

II. Linéarisation statistique par les modèles de Markov

II.1. Principe général

L'étiquetage morpho-syntaxique consiste à assigner à chaque mot d'une phrase d'un texte des informations morpho-syntaxiques qui décrivent quel est le rôle du mot dans la phrase. Ces informations sont la catégorie grammaticale (Nom,Adjectif...) et les traits morphologiques (masculin,pluriel,...). Par exemple, si on prend la phrase suivante, *je bois du rouge.* On voudrait obtenir automatiquement en sortie la phrase étiquetée suivante, *je.PRO bois.V du.DET rouge.N.* Le problème principal vient du fait que les mots de la langue traitée sont ambigus. On peut parfois leur affecter plusieurs étiquettes morpho-syntaxiques différentes. Dans la phrase précédente, le mot *rouge* peut à la fois faire référence au *vin rouge*, donc un nom, mais aussi à un adjectif pour la *couleur*. Un étiqueteur va tenter de désambiguiser l'automate ambigü et choisir ensuite la meilleure séquence d'étiquettes pour la séquence de mots de la phrase.

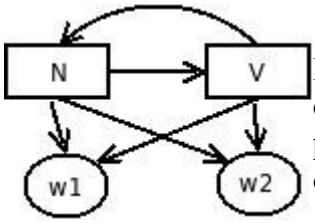


Le schéma d'un étiqueteur standard se divise en trois parties distinctes, tout d'abord il faut effectuer la tokenisation du texte. On découpe le texte en unités lexicales simples. Ensuite on effectue une analyse lexicale ambigüe qui consiste à assigner à chaque token-mot l'ensemble des étiquettes possibles (à partir d'un lexique comme les dictionnaires *Delaf* ou *Delacf* d'Unitex). Les mots inconnus subissent un traitement spécial car on ne dispose d'aucunes informations lexicales sur eux. Toutes les ambiguïtés des mots d'une phrase seront représentées par un automate acyclique.
Et enfin la dernière phase est la phase de levée d'ambiguïté qui recherche le meilleur chemin dans l'automate de la phrase (séquence reconnue par l'automate). Cette étape sera résolue selon différentes approches. Les plus courantes sont les approches statistiques basées la plupart du temps sur des probabilités (modèles de Markov cachés,...). La particularité de ces approches est qu'elles utilisent un corpus d'apprentissage annoté manuellement. Ensuite il y a une approche symbolique dont les règles contextuelles et lexicales sont écrites à la main. Et enfin il existe des approches hybrides qui combinent les deux types d'approche précédents.

L'architecture de notre étiqueteur reprend le principe des approches hybrides, c'est à dire que nous allons utiliser tout plusieurs modules de d'élagage d'automate (appelés *DelPaths*, *AddPaths*, *LearningErrors*,...) associés à un désambiguiseur symbolique (ELAG) pour finir avec un désambiguiseur stochastique probabiliste basé sur les modèles de Markov cachés de deuxième degré. Ce dernier va linéariser l'automate du texte en entrée et donc attribuer une étiquette par token-mot d'une phrase. Il reprend les principes de l'étiquetage morpho-syntaxique introduit par [Kupiec 1992]. Nous allons combiner l'utilisation des HMM avec un algorithme dynamique de recherche de meilleur chemin en temps réduit, [Viterbi 1967].

II.2. Modèle probabiliste de Markov

Nous allons maintenant expliquer en quoi consiste les modèles de Markov cachés de deuxième degré et le modèle trigramme qui en découle.



Les états du HMM correspondent aux étiquettes morpho-syntaxiques et les observables aux mots du lexique. Les transitions sont étiquetées par un poids qui est la probabilité de passer d'un état à un autre ou d'un état vers un observable.

Le processus d'étiquetage dans ce modèle consiste à retrouver la suite d'états cachés *w* la plus probable étant donné une suite d'observables *t*. $T = argmax_t(P(w/t))$

Les paramètres d'un HMM se divisent en deux probabilités :

- probabilité d'émission, qui est la probabilité d'avoir un mot ***w*** étant donné son étiquette grammaticale *t*, *P(w/t)*.
- probabilité de transition, qui est la probabilité qu'une étiquette morpho-syntaxique t_i suive une étiquette t_{i-1}, $P(t_i/t_{i-1})$.

Ces deux types de probabilité nous définissent un modèle appelé modèle *bigramme* qui repose sur l'hypothèse markovienne qu'une étiquette ne dépend que de l'étiquette précédente. Selon la littérature, on peut étendre cette notion en disant qu'une étiquette dépend des n-1 étiquettes précédentes. Cependant plus on augmente le nombre de contextes utilisés pour calculer les probabilités, plus les probabilités sont faibles et donc il faut un très grand corpus. Or on ne dispose que de ressources linguistiques limitées. On se limitera donc à un contexte de deux étiquettes.

Ce modèle est appelé modèle *trigramme* et donne des probabilités plus fines qu'un simple modèle bigramme.

Les formules de base de calcul des deux types de probabilité dans le cadre d'un modèle trigramme sont :

- probabilité d'émission : $P(w_i/t_i) = \frac{freq(w_i, t_i)}{freq(w_i)}$
- probabilité de transition : $P(t_i/t_{i-2}t_{i-1}) = \frac{freq(t_{i-2}t_{i-1}t_i)}{freq(t_{i-2}t_{i-1})}$

Sachant que :

- w_i est le mot à l'index i du corpus d'apprentissage.
- t_i est l'étiquette grammaticale de w_i.
- $freq(w_i, t_i)$ est la fréquence de la paire (mot,étiquette grammaticale).
- $freq(w_i)$ est la fréquence du mot w_i.
- $freq(t_{i-2}, t_{i-1}, t_i)$ est la fréquence de la séquence des trois étiquettes grammaticales.
- $freq(t_i)$ est la fréquence de l'étiquette t_i.

Les fréquences des token-mots et des étiquettes seront apprises à partir du corpus d'apprentissage annoté du francais (voir *II.5. corpus d'apprentissage annoté du Français*).

Nous verrons que nous aurons besoin d'effectuer un traitement spécial pour les mots inconnus car la formule d'émission n'est pas adaptée (les probabilités nous donnent un chiffre nul).

Nous avons donc tous les paramètres pour calculer la meilleur séquence d'étiquettes grammaticales. En appliquant la formule de Bayes et diverses hypothèses d'indépendances des mots ente eux, nous obtenons la formule suivante :

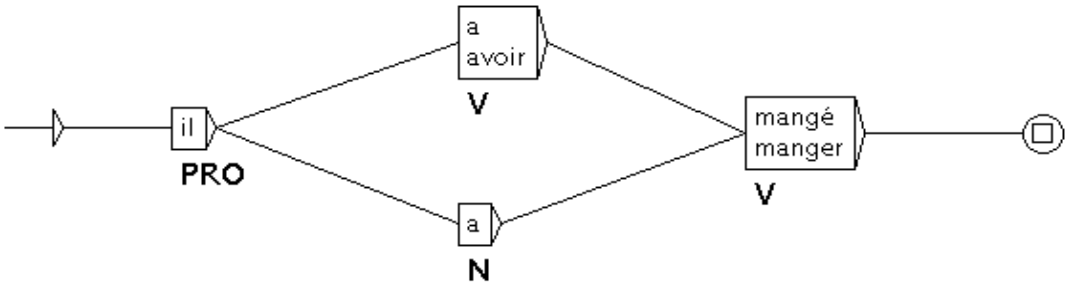
$$argmax\, P_t(w/t) = argmax_t \prod P(w_i/t_i) P(t_i/t_{i-2}t_{i-1})$$

Pour éviter de perdre en précision du fait de l'utilisation de la multiplication de valeurs inférieures à 1, une solution est de prendre le logarithme de la formule précédente. Ainsi on effectue uniquement

des additions de logarithmes de probabilités :

$$argmin\, P_t(w/t) = argmin_t \log(\prod P(w_i/t_i) P(t_i/t_{i-2}t_{i-1})) = argmin_t - \sum \log(P(w_i/t_i)) - \sum \log(P(t_i/t_{i-2}t_{i-1}))$$

Prenons un exemple :
La phrase du texte à étiqueter est *il a mangé* et nous avons le lexique *{il.PRO}, {a.N}, {a.V}, {mangé.V}*. On peut représenter cette phrase par l'automate du texte acyclique suivant :



On suppose que les probabilités d'émissions et de transitions ont déjà été apprises sur le corpus d'apprentissage. Les tableaux suivant montre les résultats. Pour simplifier les calculs, l'exemple est basé sur un modèle de type *bigramme* (tout en sachant que le principe de calcul est équivalent pour un modèle *trigramme*).

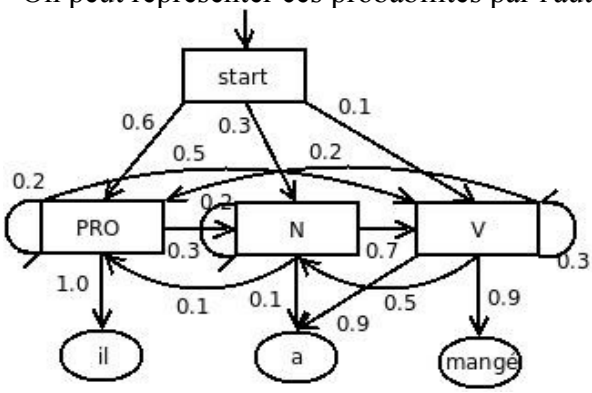
Probabilité de transitions :

étiquettes t-1 \ t	start	PRO	N	V
start	-	0.6	0.3	0.1
PRO	-	0.2	0.3	0.5
N	-	0.1	0.2	0.7
V	-	0.2	0.5	0.3

Probabilités d'émission :

mot \ étiquette	PRO	N	V
il	1.0	-	-
a	-	0.1	0.9
mangé	-	-	0.9

On peut représenter ces probabilités par l'automate suivant :



L'étiquette *start* indique le début d'une nouvelle phrase et donc sert à calculer quelle est l'étiquette suivante la plus probable en début de phrase.

Nous obtenons les chemins suivants possibles dans l'automate du texte pour la séquence de mots de la phrase :

- $P(il\,a\,mangé/PRO\,N\,V) = -\log(P(il/PRO)) - \log(P(PRO/START)) - \log(P(a/N))...$
- $P(il\,a\,mangé/PRO\,V\,V) = -\log(P(il/PRO)) - \log(P(PRO/START)) - \log(P(a/V))...$

Ce qui nous donne :

- P(il a mangé/PRO N V) = 4,48
- P(il a mangé/PRO V V) = 2,61

C'est donc le chemin étiqueté par *PRO N V* qui est optimal pour la séquence de mots "*il a mangé*".

II.3. Etiquetage des mots inconnus

Les formules de transitions et d'émissions sont à priori faites pour les mots du texte qui sont dans le lexique du corpus d'apprentissage, or il peut arriver qu'un mot dit "inconnu" fasse son apparition dans le texte traité. Le problème étant que si ce mot est inconnu alors la probabilité d'émission de ce mot est de 0 quelque soit l'étiquette grammaticale qui lui est affectée.

Nous allons donc faire un traitement particulier pour ce type de mot par l'intermédiaire d'une autre méthode de calcul de la probabilité d'émission. Dans la littérature, plusieurs articles [Brill 1995] [Thede 1998] font état de l'utilisation des suffixes pour déterminer quelle est la meilleure catégorie pour le mot. [Samuelsson 1993] a prouvé lors d'une série d'évaluation de plusieurs techniques d'étiquetage des mots inconnus que ces méthodes à base de suffixes sont celles qui donnent les meilleurs résultats. Nous avons donc adapter un algorithme basique d'étiquetage qui consiste à assigner l'étiquette la plus fréquente du plus long suffixe du mot inconnu.

Ainsi nous allons modifier la probabilité d'émission d'un mot w_i étant donné son étiquette t_i :

$P(w_i/t_i) = P(suffix(w_i)/t_i)$ avec $suffix(w_i)$, suffixe du mot w_i.

On notera que le suffixe choisi pour chaque mot w_i dépend de la longueur *n* de ce dernier :

- si $n > 6$: le suffixe choisi est de longueur 4.
- si $n < 6$ et $n > 3$: suffixe de longueur n-2.
- sinon on retourne 0 pour la probabilité d'émission, le mot est trop court pour déterminer quelle étiquette est la plus probable.

Cette modification de la probabilité d'émission nécessite une phase d'apprentissage légèrement différente par l'extraction des fréquences des paires *(suffixe,étiquette grammaticale)* pour chaque mot du corpus d'apprentissage de longueur supérieure à 3.

II.4. Algorithme dynamique de Viterbi

Pour trouver la séquence d'étiquette la plus probable à partir de l'automate, on doit calculer la probabilité d'avoir une séquence d'étiquette étant donné une séquence de mots pour chaque séquence d'étiquettes possible de l'automate. On voit bien que l'algorithme de recherche de la meilleure séquence devient exponentiel en temps. Un algorithme de programmation dynamique efficace pour résoudre ce problème est l'algorithme de Viterbi. Sa complexité réduite est de $0(n^2)$.
L'algorithme peut être décomposé en deux étapes distinctes, tout d'abord un algorithme dit *forward*. Pour une transition d'étiquette *t* (mot associé à sont étiquette morpho-syntaxique), on va se servir des étapes précédentes de calculs pour calculer le meilleur chemin possible pour passer du noeud *t-1* à *t*. A chaque fois, on note par quel chemin on passe (par quels noeuds). Puis l'étape finale est de reconstuire le chemin par lequel on est passé en appliquant un algorithme *backward*.
Prenons un exemple :

Nous choisirons comme pour l'exemple sur les HMM, un modèle bigramme pour simplifier l'explication, sachant que les calculs sont identiques tout en ayant un contexte en moins.

On commence par l'étape d'initialisation, à *t = 1 :*

On calcule quelle transition de début de phrase donne la meilleure probabilité et on sauvegarde cette transition et le noeud d'arrivé de celle-ci. Sachant que pour la calculer, on ajoute un noeud fictif *Start* dans l'automate (voir exemple HMM).

Puis pour toutes les étapes *t > 1* :

A une étape *t* (un noeud de l'automate), et pour chaque noeud *t-1* ayant une transition (de mot w et d'étiquette *t* dans la formule) arrivant en *t,* on va se servir des calculs effectués pour arriver au noeud *t-1*. Il s'agit de savoir par quel chemin le plus probable, on est arrivé en *t-1*. Ce qui nous amène à maximiser le score suivant :

$$argmax(\delta(P(t-1))xP(t/t-1)xP(w/t))$$

Avec :

δ(P(t-1)) , la probabilité partielle à l'étape *t-1*, c'est à dire le chemin le plus probable pour arriver à *t-1*.

Et enfin l'étape finale consiste à effectuer l'algorithme *backward* :

Lorsque l'on est arrivé à la dernière observation, il suffit de refaire le chemin à l'envers et récupérer les noeuds par lesquels on est passé pour reconstituer le chemin d'étiquettes la plus probable pour la séquence de mots.

```
Algorithme Viterbi  :
#entrée/sortie
entrée : automate du texte ambigü A
sortie : automate linéarisé B
#paramètres
score = { } #probabilités partielles de chaque transition de l'automate
#initialisation de l'algorithme Forward
best_score = 0
#itération, algorithme Forward
Pour chaque noeud N de A (non initial) :
    Pour chaque transition K allant du noeud O au noeud N :
        best_score = 0
        Pour chaque prédécesseur Z de O, de transition X (et d'ancêtre probable d'étiquette Y) :
            P(N) = P_trans(Y,X,K) + P_emit(K)
            L = score(O) + P(N)
            Si L > best_score :
                score(N) = (L,O,X)
#itération, algorithme backward, reconstitution du meilleur chemin
node = BestScore_Terminal(A) #noeud terminal de A dont le score est le meilleur
Tant que node != initial(A) :
    (L,N,T) = score(node) #récupération du noeud précédent de node
    B(N,T) -> node #rajout de la transition et du noeud dans B
    node = N
retourner automate linéarisé B
```

II.5. Corpus d'apprentissage annoté du Francais

Dans le cadre de l'apprentissage du modèle probabiliste utilisé par les HMM, nous avons besoin d'un corpus annoté du Français. Nous avons fais le choix de prendre le corpus arboré du Français [Anne Abeillé 2001].
Il est constitué de 43 documents XML issus du journal Le Monde, les articles étant extraits aléatoirement de plusieurs années. C'est donc un corpus généraliste.

Le choix s'est porté sur ce corpus pour plusieurs raisons :

- ➔ découpage du texte en tokens-mots (unités simples)
- ➔ informations morpho-syntaxiques pour chaque token-mot
- ➔ repérage des mots composés
- ➔ analyse syntaxique du texte (à plusieurs niveaux)

Cependant un certain nombre de défauts nuisent à l'utilisation immédiate de ce corpus.

Tout d'abord, nous pouvons remarquer que les fichiers xml ne sont pas normés par une quelconque DTD car il n'y a pas la même structure selon que l'on se trouve dans tel ou tel autre fichier.
Par exemple, les mots composés, et en particulier le début d'un mot composé est repéré par l'attribut *"compound=yes"*, or sur quelques fichiers cette balise n'est pas présente.
Dans un autre registre, on peut souligner certains mots qui n'existent pas en français et qui n'ont pas été corrigés à la main.
Un défaut plus gênant concerne les unités multi-mots, et donc les mots composés. Les étiquettes morpho-syntaxiques sont très incomplètes la plupart du temps pour les tokens-mots internes à ces mots. Les traits morphologiques ne sont jamais renseignés et parfois la catégorie grammaticale non plus.

```
<w compound="yes" lemma="la plupart de" cat="D">
  <w>la</w> // la.DET:fs
  <w>plupart</w> // plupart.N:fs
  <w>de</w> // de.PREP
</w>
```

Ce défaut est assez gênant pour notre étiqueteur comme nous le verrons par la suite car nous allons nous baser uniquement sur les mots simples à la fois pour l'apprentissage du modèle trigramme et ensuite pour l'étiquetage du texte. Les mots composés nous serviront uniquement à désambiguiser l'automate du texte à étiqueter comme nous le verrons dans les expériences *AddPaths* et *DelPaths*.
Nous allons donc résoudre ce problème par un enrichissement du corpus et donc des informations morpho-syntaxiques pour chaque token-mot interne à chaque mot composé.

II.5.1. Enrichissement du corpus

La première étape est d'extraire la liste des formes composées du corpus (les formes non lemmatisées).

```
<w lemma="journal télévisé" cat="N">
   <w>journal</w>
   <w>télévisé</w>
</w>
```

La forme composée est *journal télévisé.N.*

La deuxième étape consiste à appliquer des ressources linguistiques, en l'occurrence des dictionnaires de mots composés (*Delacf*), dans le but de trouver, pour chaque token-mot d'un mot

composé, les informations morpho-syntaxiques qui lui sont propres.
Cette étape est effectuée par Unitex avec l'utilisation de graphes reconnaissant des formes de mots composées (voir *Annexe VIII.2 formes reconnues*).

Ensuite on réecrit le corpus enrichi avec les informations morpho-syntaxiques pour chaque token-mot interne à un mot composé repéré et traité.
journal télévisé, codé N+NA dans le *Delacf* nous donne :

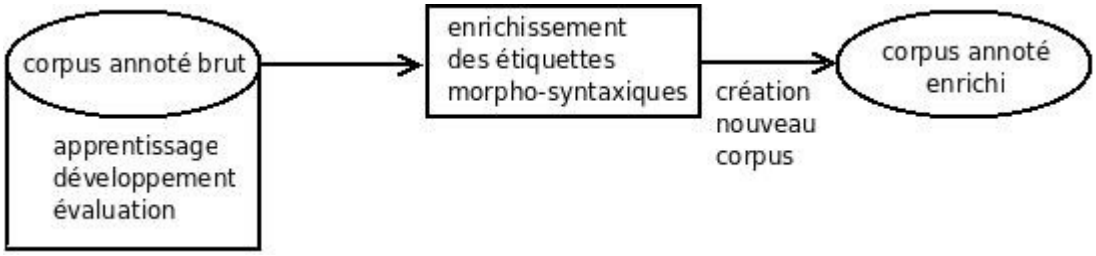
- *journal*,.N:ms
- *télévisé*,.A:ms

Nous aurons donc dans le corpusL modifié :

```
<w lemma="journal télévisé" cat="N">
   <w lemma="journal" subcat="N" ft="ms">journal</w>
   <w lemma="télévisé" subcat="A" ft="ms">télévisé</w>
</w>
```

L'évaluation de l'enrichissement lexical des mots composés du corpus a porté sur deux critères, tout d'abord selon le nombre de mots composés reconnus par les graphes Unitex et donc enrichis lexicalement (chaque mot interne obtient une étiquette morphologique complexe, c'est à dire une catégorie grammaticale et pour certaines d'entre elles, des traits morphologiques).
Nous obtenons 80,2% de formes composés extraites du corpus qui sont dans ce cas de figure.
Seuls environ 5% des mots composés non reconnus sont des noms composés. Le reste est essentiellement des verbes composés et à moindre mesure, des adverbes et prépositions composées.
Le deuxième critère porte sur le nombre de mots composés qui ont en interne des mots étiquetés au minimum par une étiquette simple, c'est à dire juste composée de la catégorie grammaticale. Nous obtenons un résultat de 95,3%.

Après la modification du corpus, nous avons été amenés à découper ce dernier en trois sous corpus, le premier est un corpus d'apprentissage utilisé pour l'apprentissage du modèle statistique probabiliste utilisé par les HMM. Il est composé de 33 documents xml et d'environ 500000 mots.
Le deuxième correspond à un corpus de développement pour le paramètrage des diverses variables de l'étiqueteur. Et enfin le dernier est un corpus d'évaluation, nécessaire dans le cadre de l'évaluation finale de notre étiqueteur morpho-syntaxique. Ces deux derniers corpus sont composés de 5 documents XML et d'environ 60000 mots.



II.5.2. Différences de jeux d'étiquettes

Le jeu d'étiquettes du corpus annoté est différent sur certains points de celui d'Unitex (voir *Annexe VIII.1 jeu d'étiquettes*).

La différence principale vient des étiquettes *ET* (pour dire *mot étranger* selon Anne Abeillé) sont utilisées dès qu'une forme composée est dite nom propre, ainsi tous les tokens-mots internes à ce mot composé seront étiquetés ET. Le mot *le conseil général* est considéré comme nom propre

composé donc il sera décomposé *le.ET conseil.ET général.ET.* De même le nom composé *Jean-Pierre* donnera *Jean.ET – Pierre.ET.* Les deux noms propres ne sont pas de la même nature, le premier est un nom composé commun alors que le deuxième est un nom propre composé. Et pourtant ils sont traités de la même manière en interne.
Dans le premier cas, *Jean-Pierre*, s'il est dans le dictionnaire des mots composés sera étiqueté *Jean-Pierre.N+Pr*, dans le cas contraire nous aurons *Jean.N+Pr – Pierre.N+Pr.* Ce nom propre composé n'est pas ambigu.
En revanche, dans le cas de *le conseil général,* il y a deux cas possibles

- *le conseil général* est étiqueté N+Pr dans le dictionnaire des mots composés ou alors *le, conseil* et *général* ont une étiquette N+Pr dans le dictionnaire des mots simples. Dans ce cas il n'y a pas de problème car on peut les étiqueter de la même façon que dans le corpus.
- Par contre, si tel n'est pas le cas, on ne pourra jamais avoir N+Pr sur chaque composant interne de ce mot composé.

Nous pouvons donc avoir une évaluation de notre étiqueteur légèrement biaisé par ce problème sur certains noms composés.

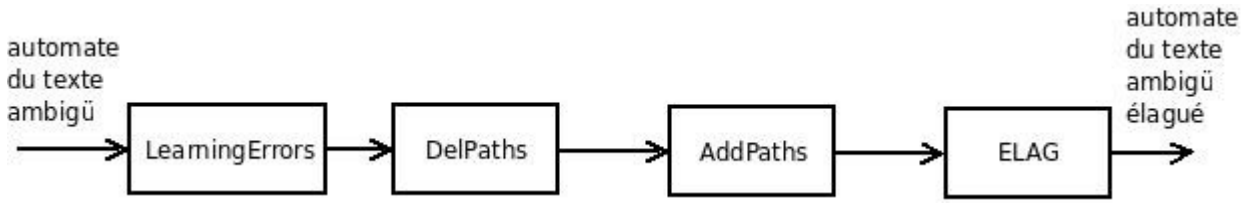
III. Algorithmes d'élagage partiel de l'automate ambigü

Nous allons maintenant décrire différentes expériences algorithmiques permettant de d'élaguer au maximum l'automate du texte avant l'étiquetage effectué par l'algorithme de Viterbi.

Cette phase en amont va faire intervenir plusieurs modules distincts :

- DelPaths, supprime des chemins en s'aidant des mots composés.
- AddPaths, ajoute des chemins étiquetés par les noms composés.
- Learning Errors, applique des règles lexicales et contextuelles apprises automatiquement à partir des erreurs d'étiquetage.
- ELAG, désambiguiseur symbolique appliquant des règles lexicales et contextuelles écrites à la main.

Ces modules se succèdent selon un schéma bien particulier qui est optimal, en terme de performance dans l'étiquetage, dans la configuration suivante (voir *V. Evaluation de l'étiqueteur morpho-syntaxique*) :



A chaque sortie de module, on obtient un nouvel automate ambigü élagué.

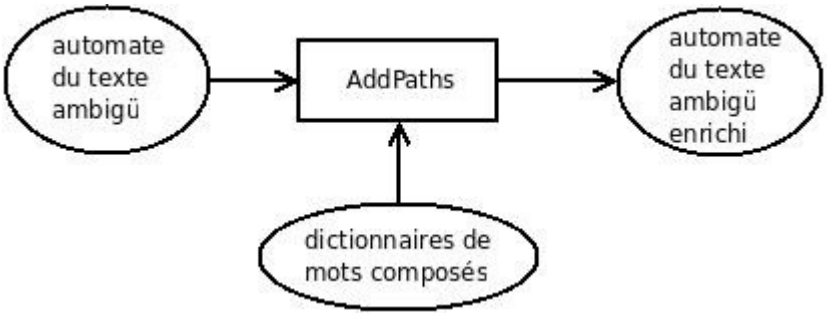
III.1. Elagage à l'aide des mots composés

Nous allons utiliser des ressources linguistiques, comme les dictionnaires de mots composés, afin d'élaguer au maximum l'automate du texte ambigü. L'idée est d'intégrer ces mots composés lors de l'analyse lexicale ambigüe. Nous aurons donc en parallèle des chemins de mots simples, des chemins étiquetés par des mots composés. Le but est d'utiliser ces derniers lors de la phase d'élagage et non lors de l'étiquetage car nous voulons comparer notre étiqueteur avec des méthodes classiques d'étiquetage des étiqueteurs actuels. C'est à dire en traitant uniquement avec des mots simples. Dans cette optique d'utilisation des ressources, nous avons décidé d'augmenter la couverture des noms composés en effectuant une extraction automatique des noms composés du texte à étiqueter (voir *IV. Extraction de collocations*). Les deux modules concernés par l'élagage à l'aide des mots composés sont AddPaths et DelPaths que nous allons voir maintenant.

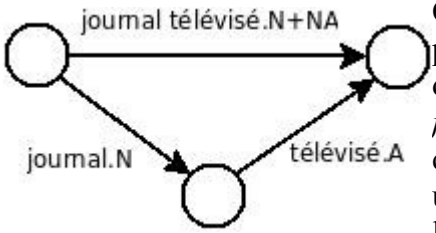
III.1.1 *AddPaths* : Ajout de chemins étiquetés par les noms composés

L'expérience AddPaths va favoriser les chemins étiquetés par des mots composés, pour ce faire, on va rajouter des chemins de mots simples dans l'automate étiquetés grâce aux structures internes des mots composés. Ces structures internes sont codées dans les dictionnaires pour chaque mot composé qui en fait partie. Chaque étiquette morpho-syntaxique des mots simples crées sera composée de la catégorie grammaticale du mot composé ainsi que de la catégorie grammaticale issue de la structure interne du mot composé associée à ce mot. Ce principe consiste donc à

effectuer un balisage des mots composés dans l'automate en utilisant uniquement des mots simples.

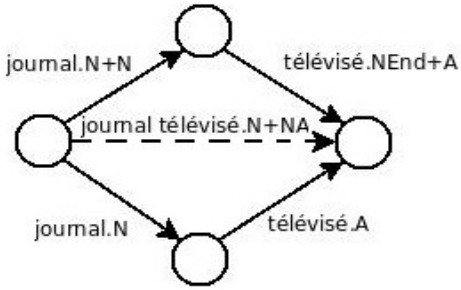


Prenons en exemple l'automate du texte suivant :



On voit qu'il y a un chemin possible dans l'automate qui passe par un mot composé *journal télévisé.N+NA*. On va donc découper ce mot composé en unités simples, *journal.N+N* et *télévisé.NEnd+A*. La balise *NEnd* indique que le nom composé se termine à ce mot. Ensuite on ajoute un nouveau chemin avec les mots simples étiquetés dans l'automate du texte.

On obtient donc l'automate suivant :



A la fin du processus de création d'un nouveau chemin, on supprime la transition étiquetée par le mot composé (en pointillée sur la figure ci-dessus).

```
Algorithme AddPaths  :
#entrée/sortie
entrée :  automate du texte ambigüe A
sortie  :  automate du texte ambigüe enrichi A
#itération, ajout des chemins de mots simples étiquetés par les mots composés
Pour chaque noeud N de A :
    Pour chaque transition sortante K de N (allant du noeud N au noeud O) :
        Si K étiquetée par un mot composé M de catégorie grammaticale C :
            #ajout du nouveau chemin de mots simples étiquetés par M entre N et O
            node = N #noeud de départ
            Pour chaque token-mot m simple de M d'étiquette grammaticale c :
                Si m, dernier mot de M :
                    A(node,(m,C+c)) -> O
                Sinon :
                    A(node,(m,C+c)) -> newnode
                    node = newnode
            Suppression de l'étiquette K
Retourner A
```

Apprentissage

Un pré-processing du corpus d'apprentissage est nécessaire pour obtenir des bons résultats avec ce module de désambiguisation. On doit modifier le jeu d'étiquettes des mots simples qui sont internes aux noms composés.
Ainsi si l'on avait dans le corpus :

```
<w compound="yes" pos="N" lemma="journal télévisé">
        <w pos="N">journal</w>
        <w pos="A">télévisé</w>
</w>
```

Dans l'expérience précédente, on extrayait simplement le mot avec sa catégorie grammaticale définie dans la balise *pos*. Ce qui nous donnait *journal.N* et *télévisé.A*. On va maintenant inscrire la catégorie grammaticale du mot composé dans ces étiquettes. Étant donné que *journal télévisé* est *N+NA*, et donc un nom composé, cela nous donne *journal.N+N* et *télévisé.NEnd+A*. On a donc une uniformisation des codes grammaticaux dans le corpus d'apprentissage et dans les étiquettes de l'automate ambigü du texte.

Modification des probabilités

Dans la première expérience, on supprimait des chemins pour en obtenir le moins possible dans l'automate. Ici on possède tous les chemins de l'automate du texte de départ plus d'autres chemins issus de mots composés.
On peut donc se poser les questions suivantes :

1. Quel chemin choisir ?
2. Comment passer par le maximum de noms composés?
3. Comment discriminer deux ou plusieurs chemins issus de noms composés?

On reprend l'intuition que les noms composés ont plus de poids que les mots simples.
Ainsi si l'on rencontre un ou plusieurs noms composés, on voudrait passer par un chemin de mots simples étiquetés par un de ces noms composés. Pour arriver à capter le maximum de noms composés, il faut modifier les probabilités d'émissions et de transitions d'un mot simple issu d'un nom composé dans le désambiguiseur statistique HMM.
Dans l'exemple précédent, *journal.N+N* doit avoir une probabilité au moins aussi grande que *journal.N*, et de même pour *télévisé.NEnd+A/télévisé.A.* On va donc définir la probabilité d'émission d'un mot simple *w* issu d'un nom composé d'étiquette grammaticale *C* comme l'addition de sa probabilité d'émission originelle pour *C* plus la probabilité d'émission de l'étiquette grammatical *c* du mot dans le nom composé :

$$P(w/C) = P(w/C) + P(w/c)$$

Ainsi sur l'exemple de *journal télévisé,* cela nous donne :

P(journal/N+N) = P(journal/N+N)+P(journal/N)
P(télévisé/N+A) = P(télévisé/N+A)+P(télévisé/A)

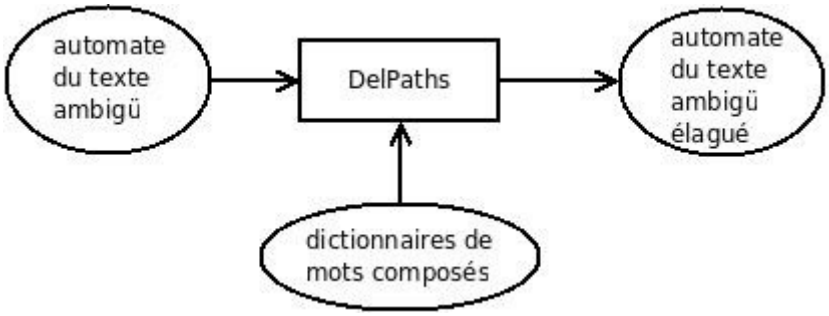
On procède de même pour les probabilités de transitions. La probabilité de transition originelle est additionnée à la probabilité d'avoir l'étiquette grammaticale *c* issue du nom composé de la transition courante. Si les contextes ont pour lettres *z* et *y* :

$$P(C/zy) = P(C/zy) + P(c/zy)$$

Si on avait la phrase *dans.PREP le.DET journal télévisé.N+NA,* la probabilité d'avoir *journal.N+N* étant donné son contexte est de *P(N+N/PREP,DET) = P(N+N/PREP,DET) + P(N/PREP,DET)* et la probabilité d'avoir *télévisé.N+A* est de *P(NEnd+A/DET,N) = P(NEnd+A/DET,N) + P(A/DET,N)*.

III.1.2 *DelPaths* : Suppression de chemins à l'aide des noms composés

L'expérience DelPaths est une autre expérience faisant intervenir les mots composés dans le processus d'élagage de l'automate. Elle a été créee dans le but de palier à un défaut de AddPaths. Le problème de ce module provient principalement des probabilités apprises sur le corpus d'apprentissage. On peut parfois passer par un chemin de mots simples initiaux plutôt que par un chemin de mots simples réétiquetés avec les mots composés. Pour régler ce problème et toujours favoriser les mots composés, nous allons donc élaguer, grâce à DelPaths, les chemins de mots simples parallèles aux mots composés. Ceci dans le but d'avoir moins de chemins par lesquels passer lors de l'étiquetage fait par la linéarisation probabiliste. De même que pour AddPaths, on s'aidera des structures internes inscrites dans ces mots composés.



Prenons en exemple le nom composé *grand journal télévisé* de structure interne *ANA* (adjectif, nom, adjectif). Les mots *grand* et *télévisé* sont ambigüs avec respectivement comme catégories grammaticales N,A et A,V. Si on suppose que *grand journal télévisé* a plus de poids dans l'automate que les chemins de mots simples alors, d'après la structure interne, on veut que *grand* soit étiqueté A, *journal* N et *télévisé* A. On supprime donc les autres analyses de lautomate qui ne correspondent pas à celles-ci, en l'occurrence *grand N* et *télévisé V*. D'après ce principe, nous nous basons donc uniquement sur les structures internes des mots composés pour supprimer certaines ambiguïtés qui seraient susceptibles d'être séléctionnées lors de la linéarisation.

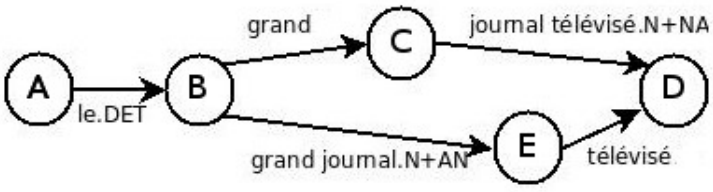
Nous pouvons poser un premier problème évident :

Quel(s) chemin(s) prendre en compte pour filtrer un maximum de mots simples?

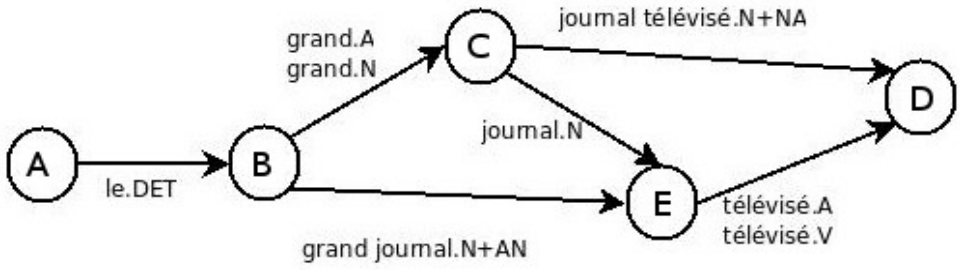
Une méthode évidente consisterait à calculer les plus courts chemins de l'automate, c'est à dire une suite successive de mots simples et de mots composés.

Algorithme des plus courts chemins

Nous allons donc définir une heuristique qui consiste à prendre les plus courts chemins de l'automate et ainsi déterminer les mots composés qui seront utilisés dans le cadre de la désambiguisation de l'automate. Nous appliquons l'algorithme de [Dijkstra 1959] pour obtenir le graphe des plus courts chemins (modifié pour qu'il retourne plusieurs plus courts chemins si tel est le cas). Ainsi si l'on possède l'automate des plus courts chemins suivants :



Et que l'automate du texte est :



Il y a trois chemins possibles dans l'automate entre le noeud A et le noeud B :

- (A,B) (B,C) (C,D) : plus court chemin
- (A,B) (B,C) (C,E) (E,D)
- (A,B) (B,E) (E,D) : plus court chemin

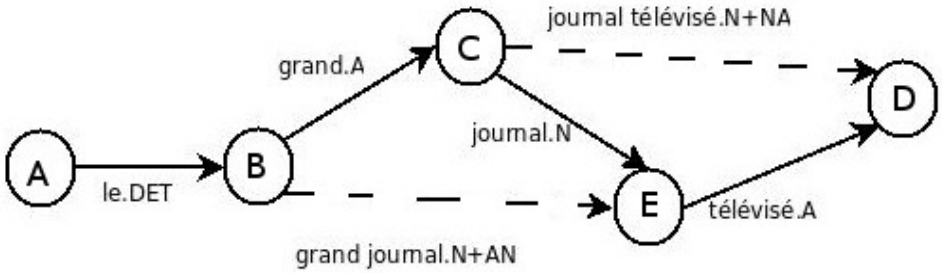
Le premier plus court chemin est composé des tokens *grand* puis *journal télévisé.N+NA*, ce qui nous donne un chemin d'étiquettes grammaticales *XNA*. Quant au deuxième plus court chemin, il nous donne *grand journal.N+AN* puis *télévisé* et donc *ANX*.
L'étiquette X indique que l'on passe par un mot simple, elle n'intervient aucunement dans la phase de filtrage des chemins (élément neutre). Les motifs grammaticaux sont donc [ANX,XNA].
Ensuite on parcourt tous les chemins possibles de l'automate à partir du noeud B et on filtre les tokens-mots selon les motifs composés de catégories grammaticales.
Le premier mot sortant de B doit être de catégorie grammaticale inclue dans [A,X].
Les transitions sortantes de B sont :

- *grand.N*, transition filtrée car N n'est pas dans [A,X]
- *grand.A,* transition non filtrée car A est dans [A,X]
- *grand journal.N+AN* (non traité car dans graphe des plus courts chemins)

Ensuite on s'interesse aux noeuds C et E (les noeuds ayant les transitions entrantes étiquetées par *journal* ne sont pas montrées car *journal* non ambigu).
Les transitions sortantes de E sont :

- *télévisé.A,* transition non filtrée car A est dans [X,A]
- *télévisé.V*, transition filtrée car V n'est pas dans [X,A]

L'automate du texte en sortie de l'algorithme est :

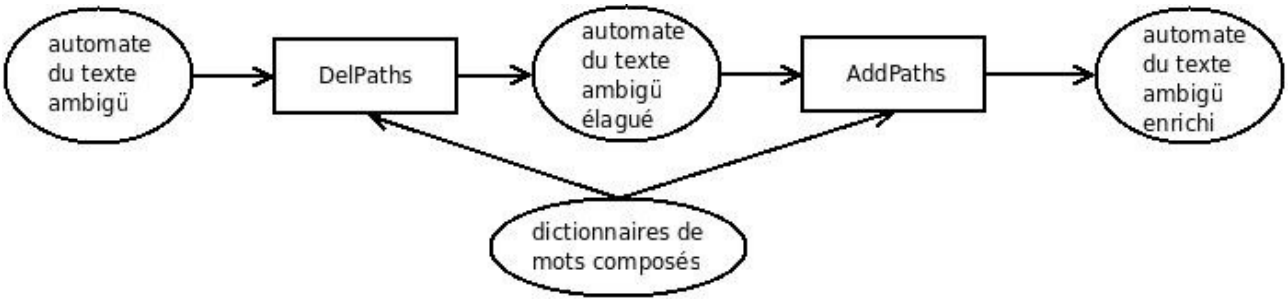


Comme pour l'expérience AddPaths, on supprime les transitions étiquetées par les mots composés à la fin du processus (en pointillées sur la figure ci-dessus).
Au final, il y a donc beaucoup moins d'ambiguïtés et donc de transitions à analyser pour chaque mot. Ceci dans le but d'avoir un algorithme final de linéarisation de Viterbi optimal tant au niveau de la rapidité que du taux de succès de l'étiquetage. Tout en sachant qu'on a fait l'hypothèse que les mots composés sont plus importants que les chemins de mots simples et que l'heuristique des plus courts chemins est correcte.

```
Algorithme DelPaths  :
#entrée/sortie
entrée : automate du texte ambigü A
sortie : automate du texte élagué ambigü A
#paramètres
Pcc = Dijkstra(A) #graphe des plus courts chemins (algorithme non détaillé dans ce rapport)
#itération, suppression des chemins de mots simples dans l'automate
Pour chaque noeud N de A :
    motifs M = Pcc(N) #extraction des motifs grammaticaux des transitions de N dans Pcc
    Pour chaque transition sortante K de N,étiquetée par le mot m de catégorie grammaticale C :
        Si C∉M :
            A(N,K) = Ø #suppression de la transition K à partir du noeud N
Retourner A
```

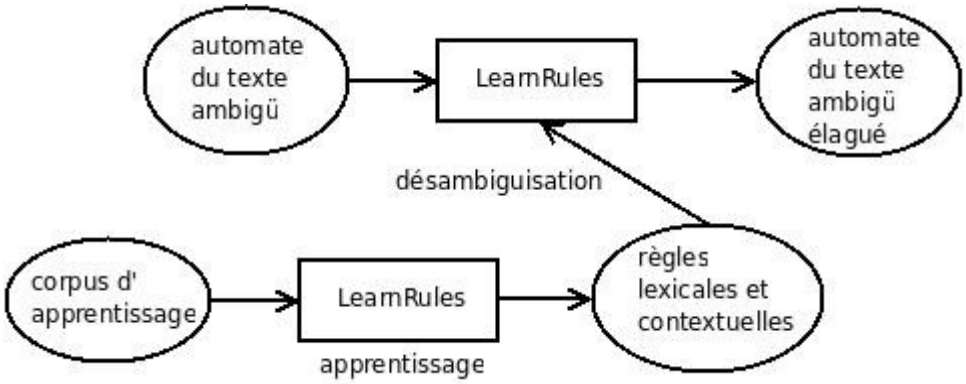
III.1.3 Couplage de AddPaths et DelPaths

Dans le cadre de l'élagage de l'automate ambigü, nous avons décidé de coupler les modules AddPaths et DelPaths, car l'un peut palier les défauts de l'autre. Les résultats que l'on obtient sont concluants car les deux modules réunis donnent de meilleurs résultats que chaque module utilisé indépendamment (voir *V. Evaluation de l'étiqueteur morpho-syntaxique*).



III.3. Learning Errors

Comme énoncé durant l'état de l'art, ce module reprend une partie du principe de l'étiqueteur de [Brill 1995] pour désambiguiser l'automate ambigü d'un texte à étiqueter. La phase d'apprentissage consiste à étiqueter initialement et basiquement un grand corpus, dans notre cas le corpus d'apprentissage, puis d'en extraire les mots, avec leurs étiquettes et leurs contextes, provoquant les erreurs les plus fréquentes. Avec ces dernières, nous allons créer des règles lexicales et contextuelles automatiquement. La deuxième phase de l'algorithme élague l'automate ambigü du texte à étiqueter, en appliquant les règles obtenues.



III.3.1 Règles de transformation de Brill

Le principe de l'étiqueteur de [Brill 1995] est de créer automatiquement des règles de transformation afin de corriger des erreurs d'étiquetage provoquées sur les étiquettes des mots d'un texte. Une règle de transformation est composée d'une partie « lexicale » qui indique la modification, la transformation, d'étiquette à effectuer et une partie « contextuelle », ou condition, qui indique dans quelles conditions effectuer cette transformation. On pourrait avoir une règle changeant un nom en déterminant si le mot concerné est *le* et le contexte droit du mot est un nom. Par exemple si l'on avait la phrase *le.N mot.N* alors d'après la règle de transformation précédente, on obtient l'étiquetage correct *le.Det mot.N*. On pourra remarquer que la partie contextuelle est bi-directionnelle, c'est à dire que l'on peut indiquer des contextes gauches et droits d'un mot à la manière des étiqueteurs précédents. Ces règles sont apprises de la façon suivante, on dispose d'un corpus de référence annoté et du même corpus brut. L'initialisation consiste à étiqueter basiquement ce corpus par une méthode aléatoire ou par un autre étiqueteur. Puis on compare les différences d'étiquetage entre le corpus étiqueté et le corpus de référence. Avec ces différences on crée des règles de transformation qui permettent de corriger les futures erreurs fréquentes ou non pour faire ressembler au maximum l'étiquetage à la vérité. Le processus est répété plusieurs fois en prenant en compte les règles créées aux étapes précédentes jusqu'à ce que le nombre d'erreurs d'étiquetage global soit inférieure à un certain palier fixé manuellement. L'ensemble des règles final est celui obtenu à la dernière itération de l'algorithme. Ces dernières sont classées selon leur nombre d'erreurs corrigées (la première étant celle qui corrige le plus d'erreurs et la dernière celle qui en corrige le moins) et sont donc appliquées dans cet ordre pour corriger un maximum d'étiquettes. L'adaptation de ce principe nous a conduit à adapter les règles de transformation d'étiquetage en règles lexicales et contextuelles applicables sur l'automate du texte. Il s'agit donc maintenant de règles d'élagage et non de règles de transformation. De plus notre

algorithme effectuera une seule itération de l'étiquetage basique afin de récupérer la liste des erreurs fréquentes grâce à l'ajout de contraintes fortes lors de la création des règles.

III.3.1 Apprentissage des règles lexicales et contextuelles

Nous appliquons notre algorithme d'étiquetage basique sur le corpus d'apprentissage, c'est à dire sans aucuns modules de d'élagage présentés précédemment (*AddPath, DelPath,* ...). A partir du rapport d'erreurs, nous récupèrons les mots provoquant le plus fréquemment des erreurs d'étiquetage. Pour chacun de ces mots, nous prélevons du corpus d'apprentissage plusieurs informations essentielles pour créer les règles. Tout d'abord, on extrait les catégories grammaticales associées (*nom N, verbe V,* ...), et pour chacune d'entre elles, sa fréquence dans le corpus pour ce mot (*N:100, V:200,* ...), ceci afin de créer des règles lexicales. Et on extrait également les contextes gauches et droits avec leurs fréquences dans le corpus pour cette catégorie grammaticale, pour créer des règles contextuelles. Et enfin on extrait également les traits morphologiques associés au mot.
Par exemple, on obtient la table suivante pour le mot *ce* :

Catégorie grammaticale	Fréquence dans le corpus	Contextes gauches	Contextes droits	Traits morphologiques
PRO	577	N: 25, CONJ: 40	V: 46, PRO: 300	PpvIL:3ms:3mp
DET	859	N: 20, CONJ: 30	A: 40, N: 530	ms
N	2	N: 2	V: 2	ms

Et on obtient la table suivante pour le mot *aussi* :

Catégorie grammaticale	Fréquence dans le corpus	Contextes gauches	Contextes droits	Traits morphologiques
PRO	577	N: 25, CONJ: 40	V: 46, PRO: 300	PpvIL:3ms:3mp
DET	859	N: 20, CONJ: 30	A: 40, N: 530	ms
N	2	N: 2	V: 2	ms

Une règle lexicale issue de cette table pourrait être que le mot *aussi*, s'il est rencontré dans l'automate du texte, sera toujours associé à *ADV*. Quant à une règle contextuelle, cela pourrait être que *ce* est toujours associé à *PRO*, si le contexte droit est un *PRO*.

L'étape suivante consiste à filtrer les mots qui pourraient provoquer beaucoup d'erreurs d'étiquetage dûe à leur ambiguïtés tant au niveau des étiquettes grammaticales que des contextes associés (après application des règles sur l'automate ambigu). Plusieurs observations/cas peuvent nous aider dans cette démarche (nous affichons les fréquences dans le corpus d'apprentissage pour chaque catégorie grammaticale citée) :

a) Le mot a une catégorie grammaticale prédominante et une ou deux autres beaucoup plus rares, en général de 1 à 9 occurrences. C'est le cas de *aussi, ADV* : 539, *CONJ* : *7*.
b) Le mot a une seule catégorie grammaticale associée. C'est le cas de *dont, PRO* : 535.
c) Certains mots ont plus de 3 catégories associées, le mot est considéré comme trop ambigüe pour être traité correctement, il est donc filtré. C'est le cas de *est* qui possède les catégories *N,A,PFX,V,...*
d) En ce qui concerne les contextes, certains mots associés à leur catégorie grammaticale ont un schéma contextuel assez constant. C'est le cas de *ce.PRO* qui est très souvent suivi d'un autre pronom *que,qui*... En revanche, il n'est pratiquement jamais suivi d'un nom, contrairement à *ce.DET*.

```
Algorithme LearnRules :
#entrée/sortie
entrée : ensemble T de mots à l'origine d'erreurs d'étiquetage fréquentes
sortie  : ensemble R de règles lexicales et contextuelles
#variables
lex_rules = {}    #règles lexicales, chaque mot est associé à sa cat gram la plus fréquente
cont_rules = {}  #règles contextuelles, chaque tuple (mot,cat) est associé à ses contextes fréquents
#itération de l'algorithme et création des règles
Pour chaque tuple M de T :
        #les catégories dans cats sont classés dans l'ordre décroissant de leur fréquence
        cats = [ensemble des cat gram de M présentes dans le corpus d'apprentissage]
        #cas n°1 : aucune ambiguïté trouvée dans le corpus, règle lexicale
        Si taille(cats) == 1 :
                lex_rules = lex_rules + {M,cats[0]}
        #cas n°2 : si une cat gram est prédominante par rapport à la deuxième, règle lexicale
        Si taille(cats) in [2, 3] && cats[0] >> cats[1] :
                lex_rules = lex_rules + {M,cats[0]}
        #cas n°3 : plusieurs cats grams prédominantes, règle contextuelle
        Si taille(cats) in [2, 3] :
                Pour chaque catégorie C dans cats :
                        cont_g = [contextes exclusifs les plus fréquents à gauche pour (M,C)]
                        cont_d = [contextes exclusifs les plus fréquents à droite pour (M,C)]
                        cont_rules = cont_rules + {M,C,cont_g,cont_d}
        #cas n°4 : le mot est trop ambigü au niveau des cats grams, il ne faut rien faire
        Si taille(cats) > 3 :
                do nothing
Retourner lex_rules, cont_rules
```

Appliquons cet algorithme sur les mots *aussi* et *ce* :
Le mot *aussi* a deux catégories grammaticales possibles, *ADV*: 539, *CONJ*: *7*. Nous sommes dans le cas n°2 de l'algorithme car la catégorie *ADV* prédomine en terme de fréquence, nous pouvons donc déduire que nous attriburons toujours la catégorie grammaticale ADV au mot *aussi*.
Le mot *ce* a trois catégories grammaticales, *DET*: 859, *PRO*: 57*7, N: 2* d'après la table précédente. Nous sommes dans le cas n°3 de l'algorithme car les catégories *DET* et *PRO* prédominent. Il faut donc extraire les contextes les plus fréquents pour chaque catégorie grammaticale et qu'ils soient exclusifs à cette catégorie. La catégorie N n'est pas assez fréquente et n'est donc pas traitée.
Nous obtenons pour :

- la catégorie *DET*, les contextes droits fréquents *N* et *A*.
- la catégorie *PRO*, les contextes droits fréquents *PRO* et *V*.

Les contextes gauches des deux catégories, *N* et *CONJ*, ont environs la même fréquence donc on ne peut les discriminer sur ce point.

III.3.2 Application des règles sur l'automate

Nous allons maintenant élaguer l'automate ambigü du texte à étiqueter en appliquant les règles lexicales et contextuelles sur ce dernier.
Pour chaque transition sortante de l'automate, si le mot de l'étiquette est concerné par une des règles, alors il y a deux cas possibles :

- Si le mot est concerné par une règle lexicale alors on supprime toutes les transitions sortantes du noeud courant, c'est à dire toutes les étiquettes grammaticales du mot, et on ajoute la nouvelle transition étiqueté par la catégorie grammaticale la plus fréquente pour le mot (voir ci-dessous le cas n°1 avec le mot *aussi*).

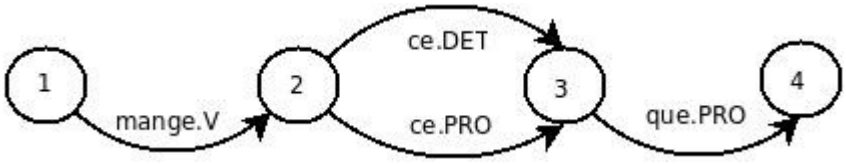
- Sinon le mot est concerné par une règle contextuelle, alors on extrait les contextes gauches et droits du mot et on regarde quelles sont les catégories grammaticales du mot qui possèdent ces contextes fréquents dans la règle(voir ci-dessous cas n°2 avec le mot *ce*). On dégage plusieurs cas d'application de la règle :
    - Si le contexte gauche et le contexte droit sont fréquents avec la même catégorie et une seule, il n'y a aucun problème.
    - Une règle contextuelle tenant compte d'un contexte gauche est prioritaire sur une autre avec un contexte droit uniquement.
    - Si plusieurs contextes gauches sont fréquents avec différentes catégories grammaticales pour le mot alors on ne peut décider quelle règle appliquer, on passe au noeud suivant.
    - Si aucuns des contextes gauches ou droits ne sont fréquents avec une catégorie grammaticale possible pour le mot alors on passe au noeud suivant.

  Si on trouve une catégorie grammaticale candidate, alors de la même façon qu'avec une règle lexicale, on supprime toutes les transitions sortantes du noeud courant et on insère la nouvelle transition étiqueté par la catégorie grammaticale adéquate pour le mot.

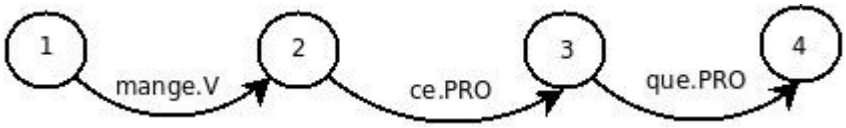
Exemples :
Cas n°1 : *L'inverse a aussi été verifié.*

Le mot *aussi*, situé en sortie du noeud 2 est concerné par une règle lexicale qui dit qu'il est toujours lié à *ADV*. On supprime donc toutes les transitions sortantes du noeud 2 arrivant au noeud 3., *aussi.CONJ* et *aussi.ADV*. Puis on ajoute une transition étiquetée par le *aussi.ADV*. Ce qui nous donne l'automate élagué suivant :

Cas n°2 : *Je mange ce que le serveur me donne.*



Le mot *ce* situé en sortie du noeud 2 est concerné par une règle contextuelle qui dit qu'il est toujours associé à *PRO* s'il est suivi par un *PRO*. C'est le cas ici donc on supprime toutes les transitions sortantes du noeud 2 arrivant au noeud 3, *ce.DET* et *ce.PRO*. Puis on ajoute une transition étiquetée par *ce.PRO*. Ce qui nous donne l'automate élagué suivant :



```
Algorithme ApplyRules :
#entrée/sortie
entrée : automate du texte ambigüe A, ensemble de règles lexicales L et contextuelles T
sortie : automate ambigü A élagué
#itération de l'algorithme et application des règles
Pour chaque noeud N de l'automate A :
    Pour chaque mot M des transitions sortantes (noeud N à noeud O) de N :
        Si M dans L alors :
            #cas d'application d'une règle
            A(N,O) = Ø #suppression de toutes les transitions entre N et O
            pos = L(M)
            A(N,(M,pos)) -> O #insertion d'une transition entre N et M d'étiquette (M,pos)
        Si M dans T alors :
            lcat = extract_left_context(A,N,T(M))
            rcat = extract_right_context(A,N,T(M))
            Si size(lcat) > 1 :
                # plusieurs cat gram sont reconnues par la règle avec les contextes gauches
                do nothing
            Si  lcat == rcat :
                # les deux contextes réfèrent à la même catégorie grammaticale
                goto cas d'application d'une règle M dans L avec lcat
            Si lcat != rcat :
                # la catégorie grammaticale associée au contexte gauche est prioritaire
                goto cas d'application d'une règle M dans L avec lcat
            Si lcat == None ou rcat == None :
                # un des contextes est reconnu par la règle contextuelle
                goto cas d'application d'une règle M dans L avec la cat gram non nulle
            Si lcat == rcat == None :
                # les contextes ne correspondent pas à la règle contextuelle
                do nothing
```

III.3.3 Evaluation

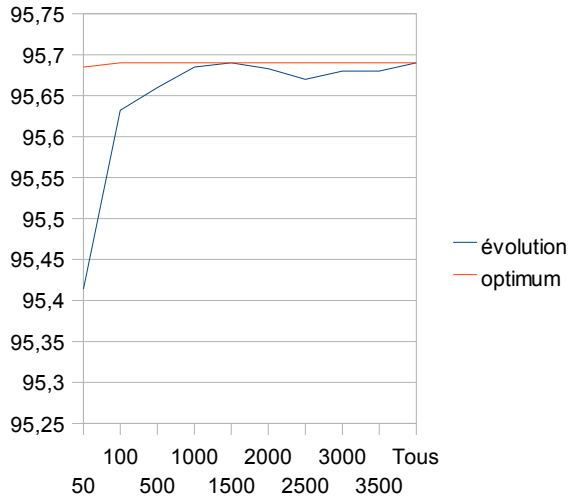
Cette première évaluation de l'algorithme consiste à observer l'évolution de l'étiquetage morpho-syntaxique du corpus d'évaluation en prenant en compte de plus en plus de mots provoquant des erreurs d'étiquetage, en ordre décroissant en terme de nombre d'erreurs d'étiquetage provoquée sur le corpus d'apprentissage.

L'étiquetage du corpus d'évaluation utilisant l'algorithme Basic est utilisé à titre de comparaison (voir *V. Evaluation de l'étiqueteur morpho-syntaxique* pour une description de cet algorithme).

Nombre d'erreurs traitées	Règles lexicales	Règles contextuelles	Mots connus	Mots inconnus	Total	Evolution 0 à t/(t-1 à t)
Basic (0)	-	-	95,561	92,162	95,414	-
50	12	1	95,793	92,162	95,632	+0,22/+0,22
100	29	3	95,793	92,162	95,632	+0,22/+0,00
500	187	15	95,812	92,162	95,660	+0,25/+0,03
1000	410	17	95,840	92,243	95,685	+0,27/+0,02
1500	679	18	95,840	92,243	95,685	+0,27/+0,00
2000	985	18	95,836	92,285	95,683	+0,25/-0,02

2500	1351	18	95,830	92,285	95,678	+0,20/-0,05
3000	1708	18	95,832	92,285	95,680	+0,22/+0,02
3500	2051	19	95,832	92,285	95,680	+0,22/+0,00
Tous (3679)	2171	19	95,832	92,327	95,682	+0,24/+0,02

Ce qui nous donne cette courbe statistique :

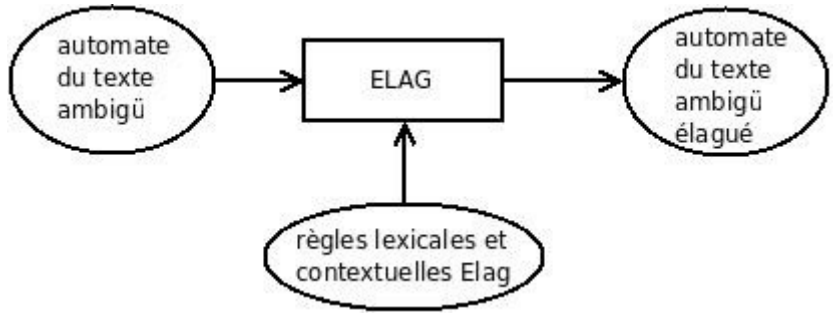


On observe une courbe classique qui nous montre que les mots provoquant le plus d'erreurs d'étiquetage, lorsqu'ils sont corrigés, nous rapprochent de l'optimum très rapidement.

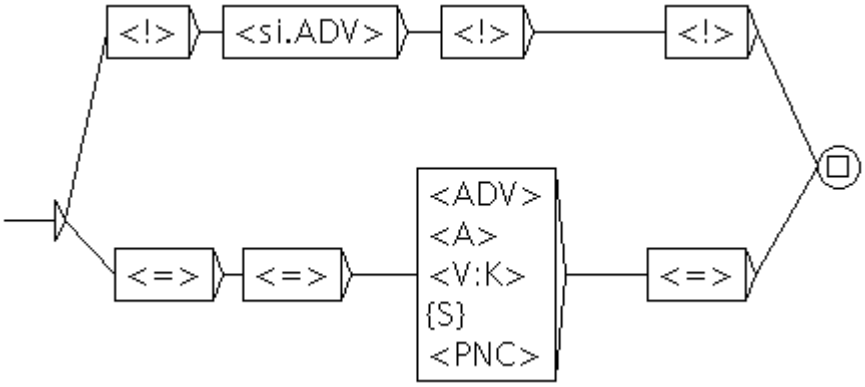
Ensuite la courbe est relativement stable jusqu'à la prise en compte de la totalité des mots.

III.4. ELAG, désambiguiseur symbolique

Les modules de désambiguisation précédents étaient des désambiguiseurs statistiques, or si nous voulons être très précis sur certains points de grammaire de la langue et donc sur certaines étiquettes morpho-syntaxiques, un coût humain est nécessaire. Ce coût se retrouve dans la désambiguisation statistique puisque nous devons utiliser un corpus annoté de grande taille crée manuellement.
Nous allons introduire ELAG [Laporte 1999], un système qui décrit un ensemble de grammaires écrites à la main définissant des règles lexicales et contextuelles qui fonctionnent sensiblement de la même façon que LearningErrors sur l'automate du texte ambigüe. Ces règles suppriment les chemins ambigües qui ne les respectent pas.



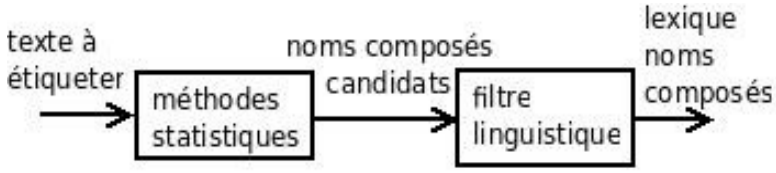
Notre étiqueteur applique l'ensemble des règles du Français d'Elag, c'est à dire 45 règles.
Pour décrire le formalisme d'Elag, voyons un exemple de règle définit pour les adverbes.



Cette règle concerne le mot *si* qui est *adverbe*. Il ne peut exister que si le contexte droit est composé des catégories suivantes, *adverbe*, *adjectif*, *participe passé*, *une ponctuation* ou *une balise de fin de phrase*. Par exemple dans les phrases suivantes, *elle est si.ADV belle.A*, *si.ADV intimement.ADV liée*...Si tel n'est pas le cas alors l'ambiguïté *si.ADV* est supprimé de l'automate du texte.

IV. Extraction de collocations "*Cecf*"

Pour obtenir un lexique de mots composés relativement conséquent à utiliser dans les expériences *AddPaths* et *DelPaths*, nous avons eu l'idée d'extraire automatiquement les collocations du texte à étiqueter. Nous pouvons ainsi mieux coller au texte et avoir un lexique spécifique au texte, plutôt que d'utiliser des ressources généralistes, même si elles sont très complètes. Les collocations, par définition, sont des mots qui apparaissent souvent ensemble, or ce que nous voulons ce sont des mots composés et non des collocations (les mots composés sont une partie de l'ensemble des collocations). Cependant nous parlerons bien d'extraction de mots composés dans la suite de ce rapport, car nous allons mettre des contraintes tellement fortes sur les algorithmes d'extraction de collocations classiques qu'en sortie nous obtiendrons des mots composés au sens propre du terme. Seuls les noms composés sont concernés par cette extraction car leurs structures internes sont relativement constantes contrairement aux autres catégories de mots composés comme les adverbes ou les verbes, où l'on peut insérer de multiples modifieurs.



Il existe plusieurs approches concernant l'extraction automatique de collocations. Certains chercheurs adoptent des méthodes entièrement statistiques [Smadja 1993]. D'autres se basent uniquement sur une analyse syntaxique complète à base de filtres linguistiques [Seretan 2004]. Nous avons opté pour une approche hybride [Todirascu 2008] qui combine nos propres méthodes statistiques et filtres linguistiques. Le principe de l'algorithme consiste à appliquer une serie de méthodes statistiques permettant d'extraire les noms composés candidats. Puis nous appliquons des filtres linguistiques, spécifiques aux noms composés, pour filtrer le maximum de mauvais candidats en fonction de leurs structures internes. Nous appelerons le dictionnaire de noms composés en sortie de cet algorithme le *Cecf* (nom réutilisé par la suite dans le rapport).

IV.1 Pré-processing et apprentissage

La phase d'apprentissage de l'algorithme concerne uniquement les méthodes statistiques qui ont besoins d'avoir des informations sur les fréquences de co-occurrences des mots dans le texte traité. Un co-occurrent d'un mot est un mot qui apparaît dans les contextes du mot dans la phrase. Pour que ces fréquences soient exploitables, il faut au préalable apprendre sur un corpus de grande taille, que nous agrémenterons avec celles de chaque texte à traiter. Nous avons choisi une partie du corpus "mf94.utf8" (le Monde 94) composé de 300000 phrases.
L'apprentissage consiste à extraire tous les bigrammes lexicaux du texte traité dans une fenêtre de taille 8 autour de chaque mot (fenêtre que nous appelerons fenêtre de co-occurrence). Les mots composés qui nous interessent sont les noms composés donc formés de mots simples ayant pour catégorie grammaticale essentiellement *N* ou *A,* ainsi que des connecteurs possibles entre ces mots (conjonctions, déterminants, prépositions...). Nous ne traitons pas les verbes car ils sont trop ambigües au niveau du rôle dans la phrase courante, sachant qu'ils peuvent être le verbe principal ou simple "adjectif" dans le nom composé. Nous avons donc besoin d'attribuer des étiquettes grammaticales à chaque mot du corpus pour ne prendre que celles qui nous intéressent. Pour ce faire, nous avons utiliser un étiqueteur externe Tree-Tagger, car il est rapide et simple d'utilisation. De plus, celui-ci est capable de lemmatiser les mots, ce qui est très utile puisque nous n'allons pas travailler avec les mots tels quels mais lemmatisés. Les fréquences seront beaucoup plus

exploitables. La première phase de l'apprentissage consiste à extraire pour chaque bigramme de mots la distance relative entre le premier et le deuxième mot de chaque bigramme (en terme de nombe de mots qui les sépare).
Prenons les mots *monnaie* et *nation* avec des bigrammes rencontrés dans le texte :

monnaie et nation : *nation* est à une distance de 1 de *monnaie*
monnaie de la nation : *nation* est à une distance de 3 de *monnaie*
nation de la monnaie : *nation* est à une distance de -3 de *monnaie*

Ainsi nous obtenons un vecteur de distances relatives pour les bigrammes de mots *nation* et *monnaie,* [*monnaie*,*nation*] = [1,3,-3]. De plus, la taille du vecteur nous indique la fréquence d'apparition des bigrammes qui est de 3 pour ces deux mots. Le vecteur de distances et la fréquence de chaque bigramme sont nécessaires pour nos méthodes statistiques.

```
Algorithme trainingStats :
#entrée/sortie
entrée : texte T étiqueté et lémmatisé avec Tree-Tagger
sortie  :  ensemble V de bigrammes associés à leurs vecteurs de distances
#paramètres
V = { }
#itération, création des vecteurs de distances pour chaque bigramme de mots de T
Pour chaque tuple (M,C) de T : #paire (mot,catégorie grammaticale)
    Si C∉[N, A]  :
        do nothing
    Sinon :
        Pour chaque co-occurrent (O,Co) de M (fenêtre de taille 8) : #paire (mot, cat gram)
            Si  Co∉[N, A] :
                do nothing
            Sinon :
                #ajout de la distance relative dans le vecteur du bigramme (M,O)
                V[M,O] = V[M,O]+{dist(M,O)}
retourner V
```

IV.2 Extraction des noms composés

Après avoir effectuer un apprentissage de l'algorithme sur le corpus d'apprentissage puis sur le texte à traiter, nous devons maintenant extraire les noms composés candidats grâçe à diverses méthodes statistiques. Ces méthodes sont issues principalement de (Manning 1999) et reprises dans (Todirascu 2008). La première méthode est *Mean and Variance* et la deuxième est le *ttest*. Ces deux méthodes vont nous donner des bigrammes candidats en tant que noms composés. Or un nom composé peut être plus grand qu'un simple bigramme de mot, nous allons donc appliquer une phase d'assemblage des bigrammes qui nous donneront les véritables noms composés candidats.
La méthode *Mean and Variance* calcule la deviation moyenne de la distance relative des deux mots d'un bigramme par rapport à son centre moyen. Plus la déviation d'un bigramme est grande, plus les distances relatives entre les mots du bigramme varient dans le texte et donc moins ce bigramme est intéressant en tant que nom composé.
Le *mean* d'un bigramme est le centre du vecteur de distances de ce bigramme :

$$M(d(w_{1,}w_2))=\frac{1}{n}\times\sum d_i$$

Avec :

- $(w_{1,}w_2)$ le bigramme de mots
- *d* le vecteur de distances du bigramme de $(w_{1,}w_2)$
- *n* la taille de *d*

- d_i la distance à l'indice *i* de *d*

La *variance* calcule de combien les distances du vecteur dévient du *mean :*

$$V(d(w_{1,}w_2),M)=\frac{1}{n-1}\times\sum(d_i-M)^2$$

Avec :

- $(w_{1,}w_2)$ le bigramme de mots
- *d* le vecteur de distances du bigramme de $(w_{1,}w_2)$
- *M* le *mean* de *d*
- *n* la taille de *d*
- d_i la distance à l'indice *i* de *d*

Et enfin la déviation est simplement la racine carré de la variance $D=\sqrt{V}$. Il s'agit d'une normalisation de la variance.

La méthode *ttest* effectue un test sur l'hypothèse d'indépendance des mots d'un bigramme, c'est à dire si deux mots apparaissent ensemble souvent ou s'il s'agit juste du hasard. Plus la valeur du *ttest* est grande, plus l'hypothèse d'indépendance est fausse, plus le bigramme est considéré comme formant un mot composé. Cette valeur est similaire au très utilisé Chi-square.

$$t=\frac{x-u}{\sqrt{\frac{s^2}{N}}}$$

Avec :

- *x* la taille du vecteur de distances sur le nombre total de bigrammes du corpus
- *u* la probabilité d'avoir le premier mot multiplié par la probabilité d'avoir le deuxième mot
- *N* le nombre total de bigrammes du corpus
- *s* la variance du vecteur de distances du bigramme

Pour qu'un bigramme soit accepté, nous fixons un palier minimum de 1,7 pour le ttest et maximum de 1,8 pour la deviation. Ces paliers ont été testés et ce sont ceux qui donnent les meilleurs résultats.

La sortie de l'application de ces méthodes statistiques sur le texte à traiter est un ensemble de bigrammes candidats en tant que noms composés et faisant peut-être partie d'un nom composé plus grand qu'un bigramme de mots.

La dernière étape, comme énoncé auparavant, consiste à effectuer l'assemblage des différents bigrammes successifs pour obtenir des mots composés plus ou moins grands.

Prenons des bigrammes successifs extraits du texte (entrecoupés de mots de liaisons) :

union économique : deviation = 0.5, ttest = 5.3

économique et monétaire : deviation = 0.8, ttest = 5.1

Pour savoir si un bigramme peut être relier avec un bigramme précédent ou un mot composé en construction, nous calculons le rapport entre le ttest du bigramme et le ttest moyen du mot composé en construction. Ainsi si deux bigrammes sont proches, ce rapport est environ de 1, sinon on peut dire que les deux bigrammes ensembles ne conduisent pas à la création d'un mot composé.

Dans l'exemple précédent, le rapport des ttest des deux bigrammes est de : 5.3/5.1 = 1,03.

On peut donc les assembler : *union économique et monétaire*.

En revanche si on avait les bigrammes suivant :

références pour le petit : deviation = 1.4, ttest = 3.3

petit écran : deviation = 0.1, ttest = 30.5

Le résultat du rapport des ttest est de : 3.3/30.5 = 0.14, donc les deux bigrammes ne forment pas un mot composé. Cette méthode évite ainsi de créer des mots composés trop grands et de créer des mots composés discutables. Car si deux bigrammes successifs du texte ne forment pas un mot composé et que les deux sont candidats en temps que mot composé alors celui qui possède le plus grand ttest est celui qui est le probable mot composé.

élections communales : deviation = 1.4, ttest = 23.3

communales de mars : deviation = 0.3, ttest = 5.5

Le mot composé probable est bien *élections communales* et non *communales de mars*. De même dans l'exemple précédent, le nom composé candidat est bien *petit écran* et non *références pour le petit*.

```
Algorithme extractCollocations :
#entrée/sortie
entrée : texte T, ensemble V de bigrammes associés à leurs vecteurs de distances
(apprentissage)
sortie  :  lexique L des noms composés candidats de T
#paramètres
thT = palier minimum du ttest pour qu'un bigramme ne soit pas filtré
thD = palier maximum de la déviation pour qu'un bigramme ne soit pas filtré
curr_comp = Ø #mot composé candidat
#itération, extraction des noms composés candidats
Pour chaque bigramme (w,w') de T : #pris successivement dans l'ordre du texte
    Si bigramme (w,w')∉V  :
        Si curr_comp != Ø :
            L = L + {curr_comp}
            curr_comp = Ø
    d = V(w,w') #vecteur de distances du bigramme
    M = calculate_mean(d)
    D = calculate_deviation(d,M)
    K = calculate_ttest(d,V)
    Si K < thT ou D > thD :
        Si curr_comp ! = Ø :
            L = L + {curr_comp}
            curr_comp = Ø
    Sinon :
        #calcul du rapport entre le ttest moyen et le ttest du bigramme
        v = average_ttest(curr_comp) / K
        Si v >> 1 ou v << 1 :
            L = L + {curr_comp}
            curr_comp = Ø
        Sinon :
            curr_comp = curr_comp + {w'}
retourner L
```

IV.3 Application des filtres linguistiques

Les filtres linguistiques utilisés par (Todirascu 2008) sont définis par l'analyse linguistique des propriétés morpho-syntaxiques des constructions qui l'intéressent, en l'occurrence les constructions verbo-nominales. Nous avons crée des filtres beaucoup moins évolués linguistiquement car elles ne concernent que les catégories grammaticales des mots internes aux noms composés candidats. L'algorithme de filtrage consiste simplement à tester si la structure des mots composés est valide linguistiquement et donc si elle réfère à un motif de nom composé. Ces motifs ont été extraits automatiquement à partir de l'ensemble des mots composés issus du corpus d'apprentissage.
Prenons les motifs autorisés M suivants [NN,AN,NA].
Ainsi si l'on a les mots composés candidats suivants :

élections communales de structure NA qui est dans M, ce mot composé est accepté.
quarante quatre de structure AA qui n'est pas dans M, ce mot composé est rejeté.
cours des dernières de structure NPA qui n'est pas dans M, ce mot composé est rejeté.

Les mots composés en sortie de ce filtrage sont considérés comme valides et ajoutés au lexique des

noms composés du texte traité.

```
Algorithme linguisticFilter :
#entrée/sortie
entrée : ensemble de mots composés candidats C, motifs grammaticaux G
sortie  :  lexique L des noms composés de T
#paramètres
L = { }
#itération, filtrage des mots composés candidats
Pour chaque mot composé M de C :
      Si internal_struct(M) dans G :
           L = L  + {M}
retourner L
```

IV.4 Evaluation

Les évaluations ont été faites sur le texte du corpus de développement. Deux ensembles ont été évalués, le premier sur la totalité des noms composés extraits par l'algorithme et le deuxième sur une portion des 50 premiers mots en terme de valeur *ttest*.

Evaluation sur l'ensemble des mots :

Plusieurs critères sont évalués :

- rappel/précision par rapport aux noms composés extraits du corpus.
- rappel/précision par rapport aux noms composés dans le dictionnaire du texte (à partir du *delaf/delacf*).
- rappel/précision par rapport aux noms composés communs au corpus et au dico du texte.
- rappel/précision par rapport aux noms composés totaux (dico+corpus)

Le rappel *r* se calcule par le rapport du nombre de noms composés pertinents trouvés sur le nombre de noms composés pertinents *B*. Quant à la précision *p* se calcule par le rapport du nombre de noms composés pertinents trouvés sur le nombre de candidats trouvés *A*. Ces deux valeurs sont définies par les formules suivantes :

$$r = \frac{A \cap B}{B} \qquad p = \frac{A \cap B}{A}$$

Les résultats sont donnés par la table suivante :

Noms composés	Rappel	Précision
Corpus	519/733 70,80%	519/2623 19,79%
Dictionnaire du texte	1041/1492 70,1%	1041/2623 40,68%
Mots communs Dico/Corpus	457/566 80,74%	457/2623 17,4%
Tous les mots (distincts)	1103/1659 66,5%	1103/2623 42%

La précision est assez faible dû au grand nombres de noms composés extraits automatiquement, beaucoup ne sont pas dans le dictionnaire du texte et dans la liste du corpus. Nous pouvons faire plusieurs remarques, la première étant que sur les 2623 noms composés extraits, 524 sont des noms

propres composés. Et que beaucoup de noms composés pertinents comme *semaine dernière* ou *élections européennes* ne sont ni dans le corpus ni dans les dictionnaires d'Unitex. De plus nous avons été obligés d'utiliser un étiqueteur externe comme Tree-Tagger pour étiqueter à la fois le corpus d'apprentissage et le texte traité, or cet outil n'est pas excempt de défaut d'étiquetage. Et enfin certains mots composés sont assez rare en terme de fréquence, et nous n'avons donc pas assez d'informations pour les repérer.

Evaluation sur une partie des mots (50 mots) :

Nous allons maintenant évaluer la précision des 50 premiers noms composés (de ceux extraits automatiquement) en terme de valeur *ttest* et donc d'importance.

Noms composés	Précision
Corpus	32/50 64%
Unitex	38/50 76%

Ce tableau est à l'opposé de l'évaluation précédente car la précision des 50 premiers noms composés a un taux moyen d'environ 70%.

V. Evaluation de l'étiqueteur morpho-syntaxique

L'évaluation a porté sur les divers algorithmes et expériences montrées dans les chapitres précédents.

Le corpus utilisé est le corpus de test composé de 2595 lignes et de 60127 mots.

54090 mots sont connus et 2370 mots sont inconnus (le reste étant des ponctuations).

La première série d'évaluations portera sur les algorithmes basiques qui vont servir de base de comparaison et d'observation de l'évolution de l'étiquetage morpho-syntaxique lorsque nous incluerons des algorithmes plus poussés de désambiguisation de l'automate du texte.

La deuxième série d'évaluations concerne ces modules de désambiguisation développés au cours de ce rapport. Ces évaluations serviront de base de comparaison avec celles faites avec d'autres étiqueteurs sur le même corpus d'évaluation.

Toutes les évaluations seront effectuées selon deux jeux d'étiquettes différents, un simple dont les étiquettes contiennent uniquement une catégorie grammaticale et un complexe dont certaines étiquettes sont agrémentées de traits morphologiques. Sachant que ce dernier jeu nous donnera aussi une évaluation sur un jeu d'étiquettes simples (qu'on notera *simple issu de complexe*).

Les critères d'évaluations seront les mêmes pour presque toutes les tables de résultats :

- *mots connus* est le pourcentage de mots connus ayant une étiquette morpho-syntaxique correcte équivalente à celle du corpus d'évaluation pour le même mot.
- *mots inconnus* est identique à *mots connus* exceptés que les mots ne se trouvent pas dans les lexiques de mots simples.
- *global* est le pourcentage global de mots (mots connus et mots inconnus) correctement étiquetés
- *évolution* représente le pourcentage positif ou négatif par rapport à des résultats de base.

V.1. Evaluation basique de l'étiqueteur

L'algorithme *baseline* consiste à assigner l'étiquette la plus fréquente à chaque token-mot du corpus d'évaluation. Les étiquettes et les fréquences correspondantes sont calculées à partir du corpus d'apprentissage.

L'algorithme *basic* consiste à effectuer uniquement une désambiguisation statistique sur l'automate du texte crée à partir des dictionnaires lexicaux de mots simples et composés d'Unitex.

Le cas des mots inconnus posera problème dans le cadre de l'algorithme *baseline*. On attribuera l'étiquette la plus fréquente du suffixe le plus long de chaque mot inconnu.

Algorithme utilisé	Jeu d'étiquettes	Mot connus	Mots inconnus	Global
Baseline	simple	95,36	78,38	95,10
Basic	simple	95,56	92,16	95,41

Baseline	complexe	90,96	65,17	89,85
Basic	complexe	92,45	90,39	92,40
Basic	simple issue de complexe	94,08	91,67	93,98

On observe tout d'abord que l'algorithme *baseline* nous donne étonnamment de très bons résultats avec un taux d'environ 95% d'étiquetage correct avec utilisation d'un jeu d'étiquettes simple. A titre de comparaison, l'algorithme *baseline* sur le Penn TreeBank, et donc pour le cas de l'Anglais, donne environ 93% de mots correctement étiquetés. On peut se dire que le lexique du corpus d'apprentissage est trop proche de celui du corpus d'évaluation et donc certaines ambiguïtés n'apparaissent pas sur les mots de ce dernier.
L'algorithme *basic* améliore un peu l'étiquetage des mots connus et à plus large mesure les mots inconnus.

L'utilisation d'un jeu d'étiquettes complexe fait baisser grandement les résultats. On peut observer que *basic* se stabilise à un niveau relativement haut avec environ 92,50% alors que *baseline* chute en dessous de 90%. Ce qui ici est plutôt logique de part la taille du jeu d'étiquettes complexe (environ 110 étiquettes). De plus la dégradation des résultats se répercute sur l'étiquetage des catégories grammaticales seules avec une baisse de 1,40% de mots correctement étiquetés.

Nous allons maintenant effectuer une évaluation sur un algorithme basé sur le même principe que *basic* à l'exception près que les étiquettes possibles, assignables à chaque mot du corpus d'évaluation, seront extraites du corpus d'apprentissage (et non plus issues de dictionnaires d'Unitex).
Le cas des mots inconnus est résolu pareillement que pour *baseline.*

Algorithme utilisé	Jeu d'étiquette	Mots connus	Mots inconnus	Global
basic	simple	95,92	79,95	95,21
basic	complexe	91,79	65,17	90,67

Les résultats pour le jeu d'étiquettes simples est meilleur sur les mots connus que pour l'algorithme *basic* précédent. En revanche pour le jeu d'étiquettes complexes, c'est le contraire car sur les mots connus il y a une différence 0,66%. Ce qui veut dire que l'utilisation de lexiques crées par des linguistes améliorent les résultats de l'étiquetage dans le cadre d'un jeu d'étiquettes complexes.

Ces deux évaluations nous permettent de déduire plusieurs choses. Tout d'abord l'utilisation de ressources lexicales riches donnent des meilleurs résultats dans l'étiquetage morpho-syntaxique. Ensuite un algorithme de désambiguisation statistique classique basé sur les HMM couplé à ces ressources est déjà très robuste quelque soit le jeu d'étiquettes utilisé.

V.2. Evaluation des modules de désambiguisation :

Nous allons évaluer chaque module de désambiguisation indépendamment des autres en les exécutant avant l'algorithme *basic* dans le processus d'étiquetage morpho-syntaxique.

Algorithme utilisé	Jeu d'étiquette	Mots connus	Mots inconnus	Global	évolution

LearningErrors	simple	95,83	93,33	95,68	+0,24
LearningErrors	complexe	92,87	90,06	92,71	+0,31
LearningErrors	simple issue de complexe	94,68	92,01	94,50	+0,52

Nous voyons donc que *LearningErrors* améliore grandement les résultats au niveau des mots connus avec une évolution positive d'environ 0,30%. De plus les catégories grammaticales sont mieux étiquetées malgré l'utilisation d'un jeu d'étiquettes complexe avec un bond de 0,52%.

Algorithme utilisé	Jeu d'étiquette	Mots connus	Mots inconnus	Global	évolution
DelPaths	simple	95,62	93,70	95,54	+0,13
DelPaths	complexe	92,78	90,50	92,68	+0,28
DelPaths	simple issue de complexe	94,41	91,76	94,30	0,32

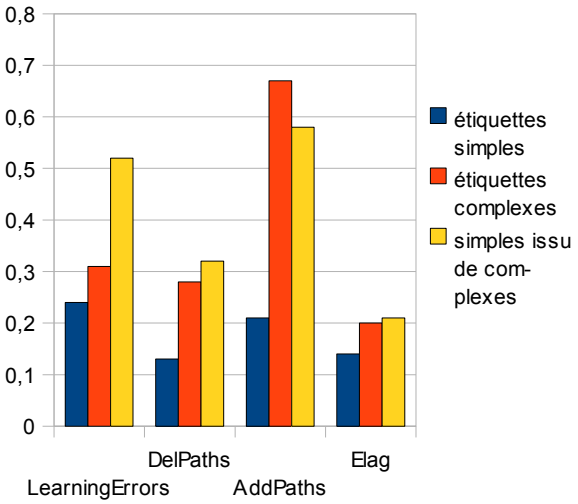
Pour *DelPaths*, on observe également une évolution positive, moindre avec un taux d'environ 0,13% seulement avec un jeu d'étiquettes simple, en revanche le taux est équivalent à *LearningErrors* pour un jeu d'étiquettes complexe.

Algorithme utilisé	Jeu d'étiquette	Mots connus	Mots inconnus	Global	évolution
AddPaths	simple	95,73	93,75	95,65	+0,21
AddPaths	complexe	93,17	90,60	93,07	+0,67
AddPaths	simple issue de complexe	94,68	91,88	94,56	+0,58

De même que pour les deux algorithmes précédents, on voit une amélioration positive, très marquée dans le cadre du jeu d'étiquettes complexe avec un bond d'environ 0,70% de mots correctement étiquettés et surtout dans la catégorie des mots connus.

Algorithme utilisé	Jeu d'étiquette	Mots connus	Mots inconnus	Global	évolution
Elag	simple	95,67	93,54	95,58	+0,14
Elag	complexe	92,72	90,55	92,60	+0,20
Elag	simple issue de complexe	94,31	91,90	94,19	+0,21

La désambiguisation symbolique faite par *Elag* est très déçevante au niveau des résultats. Ils sont relativement équivalents que ce soit pour un jeu d'étiquettes simple ou pour un jeu complexe avec environ 0,2% de mots correctement étiquetés en plus.



Globalement, on peut dégager plusieurs points de ces résultats. Tout d'abord les modules utilisant massivement les ressources lexicales riches comme *AddPaths* et *DelPaths* améliorent grandement le nombre de mots correctement étiquetés allant jusqu'à 0,67% pour *AddPaths*. Ensuite l'algorithme *LearningErrors*, comme prévu, améliore lui aussi les résultats . En revanche la désambiguisation symbolique, combinée avec la désambiguisation statistique nous donnent des résultats décevants compte tenu des attentes. Ceci à cause de *Elag* qui augmente peu les performances de l'étiqueteur.
On peut remarquer qu'en général les algorithmes sont moins performants lorsqu'il s'agit d'étiqueter un texte avec un jeu d'étiquettes simples, ceci étant dû au score déjà élevé de mots correctement étiquetés par le désambiguiseur statistique (algorithme *basic*).

V.3. Evaluation selon la prise en compte des mots composés

Nous utilisons beaucoup de ressources lexicales riches comme les dictionnaires de mots composés dans le processus d'étiquetage morpho-syntaxique, principalement dans les modules *AddPaths* et *DelPaths*.
Il serait interessant de voir quelle est l'évolution de l'étiquetage selon les catégories grammaticales des mots composés traitées dans ces algorithmes ainsi que les structures internes correspondantes.
Nous prendrons comme base de comparaison l'algorithme *basic* associé à *LearningErrors*.
On dispose de plusieurs critères d'évaluations en plus de ceux habituels :

- La *précision* nous donne le pourcentage des mots composés étiquetés qui sont dans les lexiques de mots composés.
- Le *rappel* nous donne le pourcentage des mots composés étiquetés par rapport au nombre total de mots composés du même type dans les lexiques de mots composés.
- La *fmeasure* est une mesure mélangeant à la fois le rappel et la précision, c'est une normalisation de ces valeurs.
- Le *nombre de mots composés étiquetés* nous indique quel est le nombre de mots composés qui ont été étiquetés dans le corpus d'évaluation.

Nous montrons la première table suivant un jeu d'étiquettes simple :

Catégorie traitée	Mots connus	Mots inconnus	Global	évolution	Précision	Rappel	FMeasure	Nb mots composés étiquetés
Basic	95,83	93,33	95,68	-	-	-	-	-
N	95,92	93,49	95,78	+0,10	0,31	0,65	0,43	3254
NA	95,93	93,50	95,78	+0,10	0,34	-	-	1467
AN	95,79	93,32	95,64	-0,04	0,17	-	-	365
NN	95,83	93,11	95,67	-0,01	0,50	-	-	252
NPN	95,83	93,45	95,68	+0,00	0,32	-	-	1043

NDN	95,83	93,36	95,68	+0,00	0,55	-	-	464
NPDN	95,84	93,32	95,69	+0,01	0,08	-	-	220
ADV	95,93	93,41	95,78	+0,10	0,61	0,71	0,60	692
PN	95,87	93,41	95,72	+0,04	0,70	-	-	464
PDN	95,86	93,33	95,71	+0,03	0,60	-	-	84
ADVADV	95,83	93,33	95,68	+0,00	0,75	-	-	71
ADVP	95,84	93,32	95,69	+0,01	0,65	-	-	54
PREP	95,93	93,36	95,78	+0,10	0,62	0,57	0,59	638
PDNP	95,83	93,33	95,68	+0,00	0,60	-	-	107
ADVP	95,83	93,37	95,69	+0,01	0,72	-	-	100
PNP	95,86	93,32	95,70	+0,02	0,61	-	-	282
PP	95,90	93,36	95,75	+0,07	0,80	-	-	94
ADV+N	96,04	93,58	95,89	+0,21	0,35	0,63	0,45	4094
ADV+ PREP	96,02	93,45	95,87	+0,19	0,66	0,54	0,60	1292
N+PREP	96,03	93,54	95,88	+0,20	0,35	0,63	0,45	3914
Tous	96,13	93,83	95,98	+0,30	0,38	0,61	0,47	4503

On peut dégager plusieurs points de cette table. Nous commencerons par étudier les colonnes spécifiques à *AddPaths* et *DelPaths* (colonnes de base). Tout d'abord on observe que chaque type de mots composés (suivant la catégorie grammaticale du mot, noms, adverbes et prépositions) augmente le nombre de mots correctement étiquetés à taux équivalents (+0,10%), aussi bien les mots connus, que les mots inconnus. Les différents assemblages (*N+ADV,N+PREP...Tous*) vont dans ce sens. Quant aux structures internes des mots composés, les résultats sont de plusieurs types. Chaque catégorie de mot composé possède une ou deux structures internes prédominantes en terme d'amélioration de l'étiquetage, et donc de la désambiguisation de l'automate du texte. Par exemple, on peut noter *NA* pour les noms dont l'amélioration est équivalente à celle de tous les noms composés. Pour les adverbes, il y a *PN* et *PDN* dans le même cas. En revanche d'autres structures font baisser les résultats comme *AN* pour les noms avec -0,04%.

Ensuite nous allons voir les résultats spécifiques à l'algorithme *DelPaths*, c'est à dire concernant l'étiquetage des mots composés. Globalement, si l'on prend en compte tous les mots composés, on obtient 4509 mots composés étiquetés, ce qui donne environ 2 mots étiquetés par phrase. Les noms composés sont ceux qui sont le plus étiquetés. La précision est assez faible avec 30%, ceci s'explique par le fait que l'on utilise beaucoup de dictionnaires de mots composés externes au corpus. Et donc que beaucoup ne se trouvent pas dans le corpus d'évaluation. Le rappel est quant à lui d'environ 62%, ce qui est assez faible. On possède donc une couverture assez faible des noms composés malgré le nombre important, environ 3000, de ces noms étiquetés. La situation des adverbes et composés est relativement équivalente exceptée une précision plus grande du fait de l'utilisation d'un dictionnaire de mots composés issu du corpus, et donc la possibilité d'avoir le même ensemble de mot composé.

Nous montrons la deuxième table suivant un jeu d'étiquettes complexe :

Catégorie traitée	Mots connus	Mots inconnus	Global	évolution	Précision	Rappel	FMeasure	Nb mots composés étiquetés
Basic	92,87	90,06	92,71	-	-	-	-	-
N	93,19	90,06	93,01	+0,30	0,31	0,63	0,41	3201
NA	93,13	90,19	92,96	+0,25	0,35	-	-	1471
AN	92,87	90,11	92,71	+0,00	0,14	-	-	359
NN	92,89	89,90	92,72	+0,01	0,52	-	-	222
NPN	92,88	90,11	92,72	+0,01	0,32	-	-	1025
NDN	92,87	90,06	92,71	+0,00	0,57	-	-	442
NPDN	92,87	90,06	92,71	+0,00	0,08	-	-	172
ADV	93,06	90,10	92,89	+0,18	0,68	0,50	0,57	702
PN	92,98	90,06	92,81	+0,10	0,51	-	-	558
PDN	92,90	90,07	92,74	+0,03	0,74	-	-	45
ADVADV	92,87	90,07	92,71	+0,00	0,75	-	-	71
ADVP	92,89	90,07	92,72	+0,01	0,80	-	-	60
PREP	92,02	90,15	92,85	+0,14	0,65	0,50	0,56	527
PDNP	92,88	90,07	92,71	+0,00	0,60	-	-	40
ADVP	92,90	90,07	92,73	+0,02	0,72	-	-	100
PNP	92,92	90,11	92,76	+0,05	0,62	-	-	282
PP	92,94	90,11	92,77	+0,06	0,80	-	-	95
ADV+N	93,38	90,10	93,20	+0,49	0,34	0,60	0,43	3965
ADV+ PREP	93,17	90,15	93,00	+0,29	0,65	0,48	0,55	1212
N+PREP	93,33	90,11	93,15	+0,44	0,34	0,60	0,44	3712
Tous	93,53	90,11	93,31	+0,60	0,37	0,57	0,45	4558

Cette table montre des résultats équivalents à la table précédente exceptée qu'ici les évolutions par rapport à l'algorithme *basic* sont plus marquées pour chaque type de mot composé. Les noms composés ont beaucoup plus de poids que les adverbes et prépositions composées. De plus, aucunes structures internes ne fait baisser les résultats. Ce qui veut dire que certaines ambiguïtés sont tout de même résolues par ces types de mots composés et contrebalancent les erreurs faîtes ailleurs dans le texte.

Catégorie traitée	Mots connus	Mots inconnus	Total	évolution
Basic	94,68	92,01	94,50	-
N	94,86	91,95	94,67	+0,17
ADV	94,87	92,01	94,68	+0,18
PREP	94,83	92,11	94,64	+0,14

ADV+N	95,06	91,95	94,86	+0,36
ADV+ PREP	95,00	92,10	94,80	+0,30
N+PREP	95,01	92,10	94,81	+0,31
Tous	95,14	92,01	95,01	+0,51

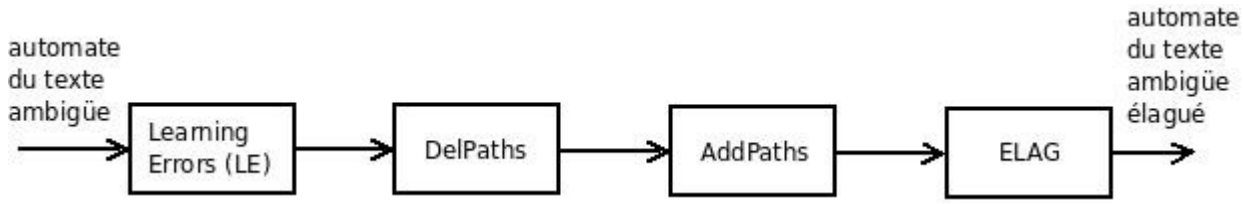
Pour les même raisons que précédemment, l'évolution est plus marquée que pour un jeu d'étiquettes simples. Ici les adverbes composés et les noms composés font jeu égal dans l'amélioration de l'étiquetage, ce qui n'était pas le cas dans la table précédente.

V.4. Evaluation des combinaisons des modules

Nous allons maintenant combiner les modules pour savoir si cela améliore les résultats et auquel cas trouver la combinaison optimale de succession des modules de désambiguisation pour notre algorithme d'étiquetage morpho-syntaxique du texte.

(LE pour *LearningErrors*, DP pour *DelPaths* et AP pour *AddPaths*)

Modules appliqués successivement	Mots connus	Mots inconnus	Global	évolution
Basic	92,45	90,39	92,40	-
LE,DP,AP,ELAG	93,59	90,50	93,46	1,06
LE,AP,DP,ELAG	93,25	89,93	93,11	+0,71
DP,LE,AP,ELAG	93,53	90,16	93,39	+0,99
DP,AP,LE,ELAG	93,21	89,97	93,07	+0,67
AP,DP,LE,ELAG	93,23	89,92	93,09	+0,69
AP,LE,DP,ELAG	93,21	89,97	93,08	+0,68
ELAG,LE,DP,AP	91,62	89,90	91,55	-0,85



D'après cette table on peut observer que la combinaison optimale successive des modules est celle que nous avons énoncé au cours de ce rapport, c'est à dire :

Modules appliqués successivement	Jeu d'étiquettes	Mots connus	Mots inconnus	Global	évolution
LE,DP,AP,ELAG	complexe	93,59	90,50	93,46	+1,06
LE,DP,AP,ELAG	simple issu	95,25	91,92	95,11	+1,12

	de complexe				
LE,DP,AP	complexe	93,51	90,60	93,38	+0,98
LE,DP,AP	simple issu de complexe	95,14	92,01	95,01	+1,02

Elle nous donne 93,40% de mots bien étiquetés avec l'utilisation d'un jeu d'étiquettes complexe, ce qui améliore de 1,06% l'étiquetage effectué par l'algorithme *basic*. L'étiquetage des catégories grammaticales est lui aussi amélioré avec un bond de 1,12%.
Cette table nous montre également que si l'on enlève *Elag* lors du processus de désambiguisation alors on observe une diminution des résultats par rapport à l'optimale. Nous avons légèrement moins de mots correctement étiquetés (0,08%).

Nous ne montrerons pas toute la table des résultats dans le cas du jeu d'étiquettes simple mais nous obtenons la même combinaison de modules optimale.

Modules appliqués successivement	Jeu d'étiquettes	Mots connus	Mots inconnus	Total	évolution
Basic	simple	95,56	92,16	95,41	-
LE,DP,AP,ELAG	simple	96,23	93,76	96,13	+0,72
LE,DP,AP	simple	96,13	93,83	95,98	+0,57

L'amélioration de l'étiquetage est du même ordre que dans la table précédente ce qui confirme que les modules de désambiguisation ont les mêmes performances quelque soit le jeu d'étiquettes utilisé. Là aussi, si l'on enlève le module de désambiguisation symbolique Elag, les résultats baissent de 0,15% ce qui est plus conséquent que pour le jeu d'étiquettes complexe.

V.5. Conclusion partielle des évaluations

Nous pouvons donc faire plusieurs remarques sur notre étiqueteur. Tout d'abord, un simple algorithme basique nous donne déjà 95,41% de mots correctement étiquetés, avec un jeu d'étiquettes simple. Ce qui est plutôt étonnant, car cela tourne autour de 90% pour l'anglais. Notre étiqueteur n'améliore donc que de 0,72% l'étiquetage par rapport à cet algorithme. De même pour un jeu d'étiquettes complexe où l'on a au maximum 1% de plus. Avec ce jeu d'étiquettes, on voit que l'étiqueteur donne 93,40%, ce qui est correct étant donné le nombre important d'étiquettes assignables possibles (116 étiquettes). La plus grande déception vient de la désambiguisation symbolique effectuée par Elag qui améliore peu les résultats de l'étiquetage. Il faudrait peut-être augmenter le volume de règles lexicales et contexuelles ou encore modifier celles existantes. Quant aux modules de désambiguisation statistique, on peut observer qu'ils ont tous environ le même impact sur l'étiquetage, en moyenne 0,25% de mots en plus. La prise en compte des mots composés, dans les algorithmes *AddPaths* et *DelPaths*, n'apporte finalement qu'une faible amélioration dans l'étiquetage, par rapport aux attentes esperées.

V.6. Comparaison à d'autres étiqueteurs

Nous allons maintenant effectuer l'évaluation d'autres étiqueteurs dans le but de faire une

comparaison avec le notre. Le premier sera TreeTagger [Schmid 1995] qui est un étiqueteur statistique probabiliste travaillant sur des arbres de décisions. Pour chaque mot à étiqueter, on dispose de contextes plus conséquents et les probabilités sont beaucoup plus précises en particulier sur les mots peu fréquents, voir inconnus (que les probabilités classiques ne peuvent résoudre de manière fiable). Il donne environ 96% de mots correctement étiquetés pour l'Anglais et sur le Penn TreeBank (ayant un jeu d'étiquettes simple). Le deuxième étiqueteur évalué sera SVMTOOL (Gimenez 2004), un outil utilisant le principe du SVM (Support vector Machine) trouvé par (Vapnik 1995) pour étiqueter le texte en entrée. Il revendique environ 97,20% de bon étiquetage sur le Penn TreeBank, toujours pour l'anglais.
Nous avons choisi ces étiqueteurs pour trois raisons principales. Tout d'abord l'apprentissage de l'étiqueteur et l'étiquetage d'un texte se font très simplement. Il suffit de donner les fichiers correspondants, le corpus d'apprentissage et le corpus d'évaluation, pour obtenir le processus souhaité. Tout est automatique. La seconde raison concerne les performances de ces étiqueteurs qui revendiquent des bons taux d'étiquetage pour l'Anglais, et particulièrement sur un corpus souvent utilisé pour les évaluations comme le Penn TreeBank). L'idée serait de tester ces étiqueteurs sur le Français pour observer s'ils ont les même performances sur cette langue. La dernière raison concerne les catégories de performances auxquelles appartiennent ces étiqueteurs. SVMTOOL est l'un des meilleurs étiqueteurs actuellement alors que TreeTagger est un cran en dessous en terme de performance. Il serait intéressant de pouvoir classer notre étiqueteur parmi une des ces catégories de performances.

Méthodes d'évaluation

Comme dit précédemment, nous avons effectué l'apprentissage des étiqueteurs sur le corpus d'apprentissage et l'étiquetage basique sur le corpus d'évaluation. Mais ce qui nous intéresse particulièrement depuis le début de ce rapport, c'est l'utilisation des ressources de mots composés et de mesurer l'impact, en positif ou en négatif de ces mots sur l'étiquetage final. Or ces étiqueteurs ne traitent qu'avec des mots simples, de plus nous ne pouvons intervenir en aucun cas durant le processus d'étiquetage. Nous avons donc mis en oeuvre plusieurs procédés permettant de prendre en compte ces mots composés. Une première série d'évaluation concerne l'algorithme basique d'étiquetage avec une phase d'apprentissage différente. On effectuera un pré-processing sur le corpus d'apprentissage qui va consister à étiqueter différemment les mots simples internes aux mots composés comme pour notre étiqueteur avec le module *AddPaths*. Les mots composés seront donc inscris dans ce corpus. Une autre expérience consiste à faire un post-processing de l'étiquetage final réalisé par l'algorithme basique en repérant les mots composés et en changeant l'étiquetage des mots simples internes à ces mots. Comme pour *AddPaths* ou *DelPaths*, nous prendrons en compte les noms, adverbes et prépositions composés dans le pré-process et dans le post-process.
Evaluations

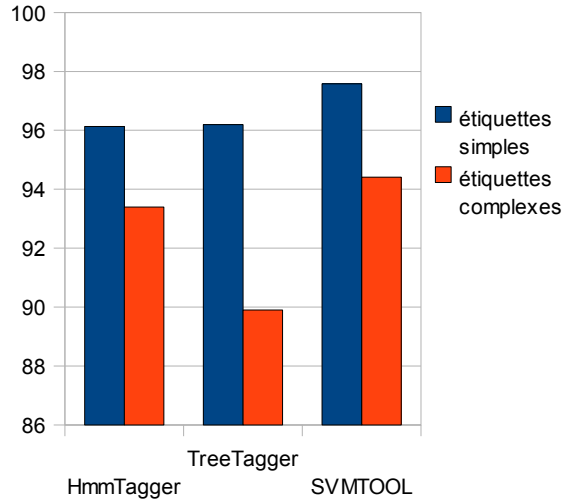
Nous commencons par l'étiquetage basique des étiqueteurs selon les deux jeux d'étiquettes, simple et complexe. Nous appelerons notre étiqueteur HmmTagger pour la suite des évaluations. La table suivante montre les résultats obtenus pour le jeu d'étiquettes simple.

Méthode	Jeu d'étiquettes	Mots connus	Mots inconnus	Total	Évolution
HmmTagger	simple	96,23	93,76	96,13	-
TreeTagger	simple	96,58	87,33	96,19	-0,06
SVMTOOL	simple	97,80	92,04	97,58	-1,45

On observe que SVMTOOL possède 97,58% de mots correctement étiquetés ce qui est légèrement supérieur à l'étiquetage sur un corpus Anglais. De même, TreeTagger nous donne 96,19%. Globalement on peut voir que TreeTagger et notre étiqueteur sont relativement équivalents en terme de performance, le notre pour les mots inconnus (+6,43) et TreeTagger pour les mots connus (+0,25%). Le mauvais étiquetage des mots inconnus par ce dernier s'explique par le fait qu'il donne l'étiquette *nom* à tous ces mots par défaut.
La table suivante donne les résultats selon un jeu d'étiquettes complexe :

Étiqueteur	Jeu d'étiquettes	Mots connus	Mots inconnus	Total	Évolution
HmmTagger	complexe	93,53	90,21	93,40	-
TreeTagger	complexe	93,44	8,80	89,90	+3,50
SVMTOOL	complexe	94,78	85,40	94,41	-1,01

De même que précédemment, on voit que SVMTOOL a de meilleures performances, cependant l'écart entre notre étiqueteur et ce dernier n'est plus que d'environ 1% sur la totalité des mots. En revanche cette fois-ci, notre étiqueteur a globalement de meilleurs performances par rapport à TreeTagger (+3,50), que ce soit sur les mots connus (+0,09) ou sur les mots inconnus (+81,41).



Ce graphique résume bien les remarques précédentes. TreeTagger et notre étiqueteur sont au même niveau de performance dans le cadre d'un jeu d'étiquettes simple. Alors qu'avec un jeu d'étiquettes complexe, ce dernier se rapproche d'un étiqueteur comme SVMTOOL. Nous obtenons donc un étiqueteur plutôt robuste quelque soit la complexité du jeu d'étiquettes.

Les prochaines évaluations vont concerner la prise en compte des mots composés dans l'étiquetage, selon les diverses méthodes, pré-process et post-process. Nous obtenons la table de résultats suivante avec un pré-process du corpus d'apprentissage.

Étiqueteur	Algorithme	Jeu d'étiquettes	Mots connus	Mots inconnus	Total	Évolution
TreeTagger	basic	simple	96,58	87,33	96,19	-
TreeTagger	pré-process	simple	96,38	87,99	96,02	-0,17
SVMTOOL	basic	simple	97,80	92,04	97,58	-
SVMTOOL	pré-process	simple	97,67	91,15	97,39	-0,19

On peut voir que la méthode pré-process couplé à un algorithme basique donne de moins bons résultats qu'un algorithme basique, à la fois pour TreeTagger et SVMTOOL. Ce qui veut dire que l'étiquetage des mots composés dans le corpus d'apprentissage fait baisser les performances des étiqueteurs. Le problème viendrait peut-être de l'augmentation de la taille du jeu d'étiquettes, à cause du réétiquetage des noms, adverbes et prépositions composés.
Voyons à présent les résultats pour un jeu d'étiquettes complexe.

Étiqueteur	Algorithme	Jeu d'étiquettes	Mots connus	Mots inconnus	Total	Évolution
TreeTagger	basic	complexe	93,44	8,80	89,90	-
TreeTagger	pré-process	complexe	93,21	24,88	90,34	+0,44
SVMTOOL	basic	complexe	94,78	85,40	94,41	-
SVMTOOL	pré-process	complexe	94,60	90,60	94,43	+0,02

Contrairement à l'évaluation précédente, sur un jeu d'étiquettes complexe, on observe une évolution positive négligeable pour SVMTOOL mais importante pour TreeTagger avec +0,44%. Malgré tout, l'évolution concerne uniquement les mots inconnus, les mots connus étant toujours en évolution négative. Passons aux évaluations de l'algorithme basique combiné au post-process. La table suivante montre les résultats obtenus.

Étiqueteur	Algorithme	Jeu d'étiquettes	Mots connus	Mots inconnus	Total	Évolution
TreeTagger	basic	simple	96,58	87,33	96,19	-
TreeTagger	post-process	simple	96,42	88,38	96,08	-0,11
SVMTOOL	basic	simple	97,80	92,04	97,58	-
SVMTOOL	post-process	simple	97,62	92,40	97,50	-0,08

On retrouve ici les mêmes évolutions négatives que pour l'algorithme pré-process, pour les deux étiqueteurs. On peut distinguer plusieurs erreurs d'étiquetage. Tout d'abord, le post-process ne tient pas compte du contexte, alors si deux mots composés sont possibles à un endroit du texte, on devra faire un choix arbitraire. C'est le cas par exemple du mot *le plus.ADV+PN* qui est étiqueté plusieurs fois à la place de *le.DET plus.ADV*. L'autre source d'erreurs provient des noms propres. *Etats-Unis* est étiqueté *N+NN* dans le corpus alors que dans le dictionnaire delacf il s'agit de *N+NA*. Ce problème a déjà été énoncé lors de la description du corpus annoté du Français.

Étiqueteur	Algorithme	Jeu d'étiquettes	Mots connus	Mots inconnus	Total	Évolution
TreeTagger	basic	complexe	93,44	8,80	89,90	-
TreeTagger	post-process	complexe	93,46	13,25	90,09	+0,19
SVMTOOL	basic	complexe	94,78	85,40	94,41	-
SVMTOOL	post-process	complexe	94,85	88,60	94,51	+0,10

Les évolutions positives sont équivalentes à l'algorithme pré-process.

V.7. Conclusion des évaluations

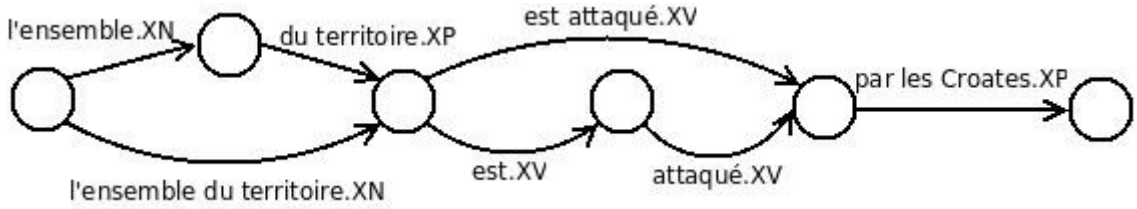
En conclusion, nous avons pu voir à travers toutes ces évaluations que notre étiqueteur hybride, utilisant un désambiguiseur statistique combiné à un désambiguiseur symbolique, donne des résultats plutôt bons. Les performances sont relativement équivalentes à un étiqueteur comme TreeTagger. Et elles sont même meilleures lorsque le jeu d'étiquettes est complexe. C'est intéressant de voir que les performances annoncées par TreeTagger et SVMTOOL pour l'Anglais se retrouvent

pour le Français. On peut donc en conclure que ces outils marchent indépendamment de la langue. Il serait utile de tester notre étiqueteur sur l'Anglais pour voir si cela se confirme.
En ce qui concerne la prise en compte des mots composés, le bilan est plutôt mitigé en terme de performance. Les algorithmes *AddPaths* et *DelPaths*, qui utilisent ces mots composés, n'augmentent en moyenne que de 0,25% le bon étiquetage du corpus d'évaluation.
Le principe de l'étiqueteur hybride, à la fois statistique et probabiliste, était plutôt intéressant. Mais l'impact de la désambiguisation symbolique faite par ELAG est finalement négligeable. Il faudrait voir si l'écriture de règles supplémentaires permettrait d'augmenter les résultats.

VI. Extension au super-chunking

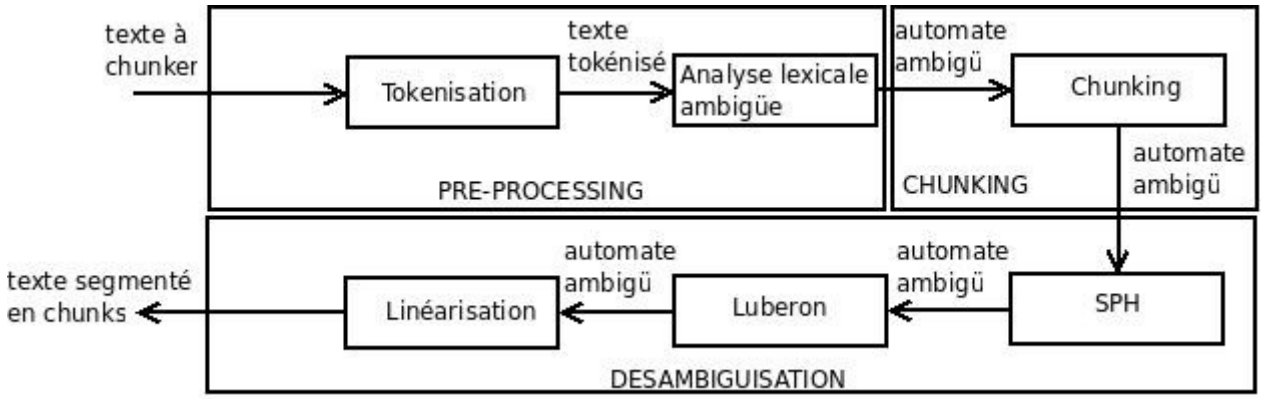
VI.1. Principe

Le chunking consiste à découper la phrase en séquences de mots, que l'on appelera chunks, auxquelles on attribue une classe grammaticale comme XN pour les chunks nominaux, XP pour les chunks prépositionnels...On définit une tête d'un chunk comme le mot le plus important de ce dernier. Il s'agit du dernier mot de la séquence de mots du chunks, excepté pour le cas des chunks verbaux négatifs où il s'agit du dernier verbe du chunk. Le chunk *n'était pas* a pour tête *était*. Prenons pour exemple la phrase *Brill vit l'homme sur la colline*. L'analyse en chunks de cette phrase est *[XN Brill][XV vit][XN l'homme][XP sur la colline]*. C'est ce qu'on aimerait obtenir automatiquement avec un chunker.
Le super-chunking reprend les principes du chunking, avec une notion supplémentaire sur la composition des chunks. Plutôt que de considérer que les chunks sont des séquences de mots simples, ils vont prendre en compte maintenant les unités multi-mots telles que les mots composés et les expressions figées. Si on avait les chunks suivants, crées par un chunker classique *[XN cordon][XA bleu]*, le super-chunker nous donnerait un seul chunk *[XN cordon bleu]* car *cordon bleu* est un nom composé et donc considéré comme une unité lexicale à part entière. Le problème principal du chunking et donc du super-chuking est le même que pour l'analyse morpho-syntaxique. L'analyse en chunks est ambigüe, ce qui veut dire qu'on peut avoir un découpage en chunks différents ou avoir une étiquette grammaticale différente pour un même découpage. Ces ambiguités seront représentées par un automate acyclique que l'on devra désambiguiser. Comme énoncé durant l'introduction et l'état de l'art, nous avons utiliser une approche hybride pour résoudre ce problème d'ambiguités, en combinant, pour l'approche symbolique, le super-chunker *Pom* [Constant 2007] avec, pour l'approche statistique, notre algorithme de désambiguisation statistique probabiliste (basé sur les HMM). Ce dernier est identique à celui vu lors de l'étiquetage morpho-syntaxique, c'est pour cela que nous parlons d'extension au super-chunking.
Pom utilise une technique classique de cascade de transducteurs trouvée par [Abney 1996]. Cette cascade est une succession de transducteurs (de grammaires) où chacun permet de reconnaître un type de chunk. L'entrée de chaque transducteur est constituée par la sortie du transducteur précédent. Ces grammaires sont en général créées à la main par des linguistes. En sortie de cette cascade de transducteurs, on obtient un automate ambigü acyclique qui contient toutes les analyses en chunks reconnus par les grammaires. On pourrait avoir par exemple la cascade simple suivante :
1: XN -> D? A N+ | PRO*
XV -> V+
2: XP -> P XN
3: S -> XN XP XV+ XN? XP**
Cette cascade a 3 niveaux, sachant que le niveau 0 est la liste des mots associés à leurs étiquettes morpho-syntaxiques possibles. De plus, on voit qu'il y a trois grammaires de chunks, pour les chunks nominaux, verbaux (au niveau 1) et prépositionnels (au niveau 2). Le niveau 3, et donc le dernier niveau, représente la grammaire d'une phrase de la langue Française. La cascade appliquée sur la phrase *l'ensemble du territoire est attaqué par les Croates* nous donne l'automate ambigü des chunks suivant :

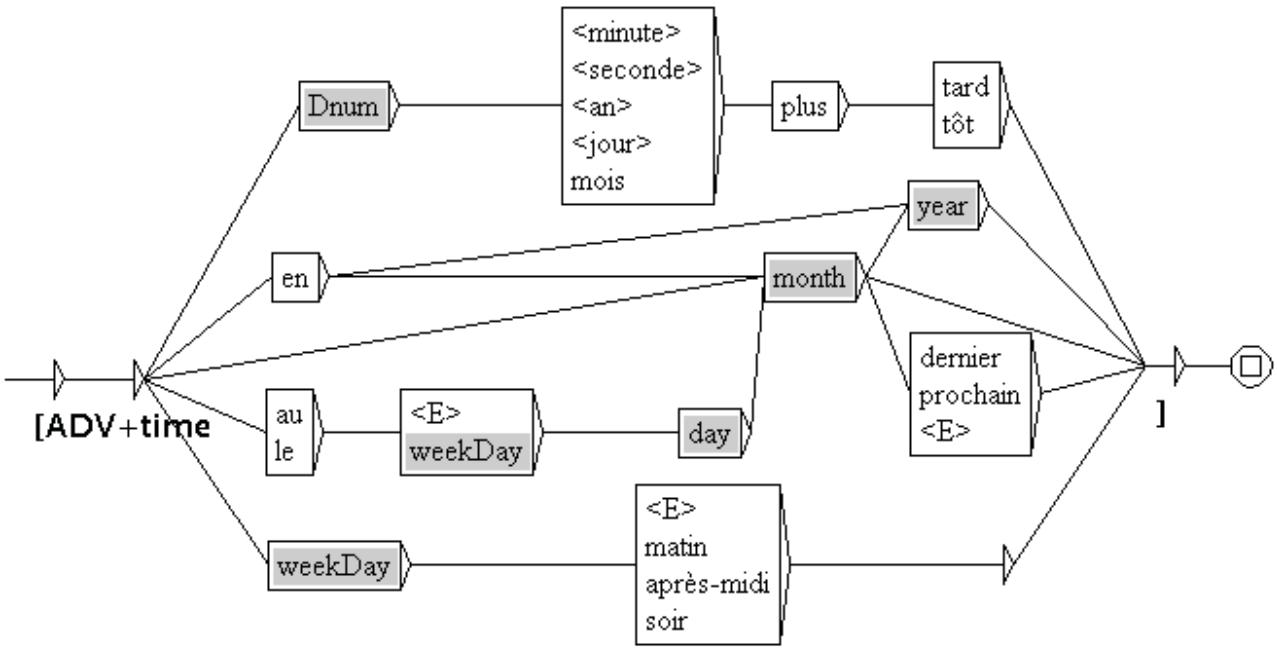


VI.2. Pom, super-chunker

Le processus d'étiquetage de ce super-chunker peut se découper en trois phases distinctes.



La première phase est identique à celle de l'analyse morpho-syntaxique. On tokenise le texte en token-mots et on effectue ensuite une analyse lexicale ambigüe grâçe à des ressources lexicales. Elle va nous produire un automate acyclique ambigü qui contient toutes les étiquettes possibles pour chaque token-mot de la phrase. Les ressources qui contiennent ces informations lexicales sont des dictionnaires de mots simples et de mots composés pour le Français écrits par des linguistiques de l'Université Paris 7 [Courtois 1990] [Courtois 1997]. Ce processus prend en compte les unités multi-mots autres que les mots composés (expressions figées, ...). Plusieurs grammaires, sous formes de graphes, permettant de les reconnaître sont appliquées sur l'automate ambigu. Par exemple, l'adverbe de temps *cinq minutes plus tard* (d'étiquette *ADV+TIME*) sera reconnu par le graphe suivant :

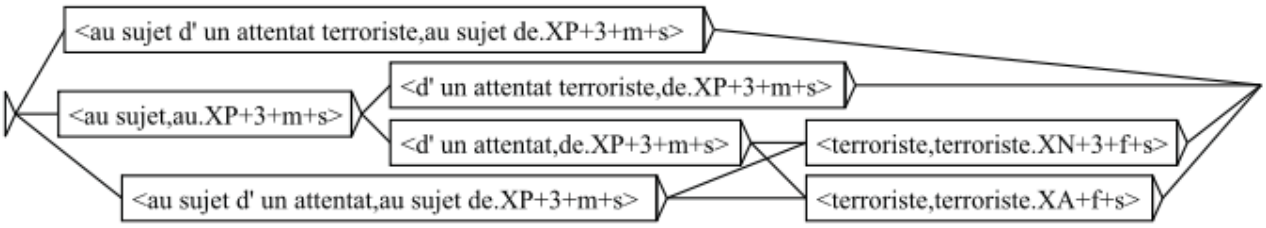


Pom utilise environ 190 graphes, contenant des grammaires locales, qui permettent de reconnaître plusieurs types d'unités multi-mots parmi lesquelles, les :

- noms : noms de professions, *ministre anglais de l'Agriculture.*

- prépositions : prépositions locatives, *à dix kilomètres*.
- déterminants : déterminants numériques, *des milliers de*.
- adverbes : adverbes de temps, *octobre 2006*.

La phase suivante consiste à segmenter la phrase en chunks. Pour ce faire, on applique une cascade de 18 graphes sur l'automate du texte ambigu. Ils vont nous permettre de reconnaitre et délimiter les séquences de mots de la phrase qui sont des chunks grâçe aux grammaires contenus dans ces graphes. Puis l'automate ambigü est nettoyé, c'est à dire que l'on va supprimer toutes les analyses qui ne font pas partie d'un chunk (ayant les catégories nom, adverbe, préposition,...).
La dernière phase est une phase de désambiguisation de l'automate ambigü car la segmentation en chunks nous a produit toutes les analyses possibles en chunks de la phrase (toujours sous forme d'automate acyclique ambigu). Il y a trois étapes distinctes facultatives. Le module SPH pour Shortest Path Heuristic définit une heuristique des plus courts chemins dérivée de [Dijkstra 1959]. Seules les analyses les plus courtes en terme de chemins sont gardées dans l'automate, les autres sont supprimées. Cela favorise évidemment les unités multi-mots par rapport aux mots simples.



L'application de l'heuristique sur l'automate ambigü ci-dessus produit une seule analyse possible qui est *<au sujet d'un attentat terroriste.XP>*. Le deuxième module de désambiguisation, appelé *Luberon*, applique un système de règles contextuelles à la manière de Elag ou LearningErrors. Une règle est constituée de 3 éléments, 2 sont les contextes gauches et droits représentés par des grammaires locales ou vides, et une partie centrale ambigüe qui liste les analyses possibles. Si l'ambiguïté est trouvée dans l'automate du texte avec les contextes gauches et droits de la règle alors la première analyse de la liste est choisie, les autres sont supprimées de l'automate. La dernière étape de la désambiguisation linéarise l'automate ambigü et produit donc une unique analyse en chunks en sortie pour le texte. Cette étape utilise un désambiguiseur statistique probabiliste pour résoudre les ambiguïtés sur les mots. Il est assez basique dans son fonctionnement puisque si un mot a toujours plusieurs étiquettes possibles après les désambiguisations précédentes, alors on va tout simplement choisir l'analyse la plus probable pour ce mot.
C'est dans cette optique de désambiguisation, et plus particulièrement la linéarisation, que nous allons adapter notre module de désambiguisation statistique probabiliste, que nous appelerons *StatProb*, en sortie du super-chunker Pom. Tout d'abord il est intéressant de tester si notre module nous donne des meilleurs résultats qu'avec la linéarisation de Pom. Ensuite l'idée est d'utiliser notre module avec différentes combinaison de modules optionnels de Pom. Par exemple, nous pouvons combiner *SPH* avec *StatProb*, *Luberon* avec *StatProb*, ou encore uniquement *StatProb*. Le chapitre suivant montre les résultats obtenus lors des évaluations.

VI.3. Méthodes d'apprentissage

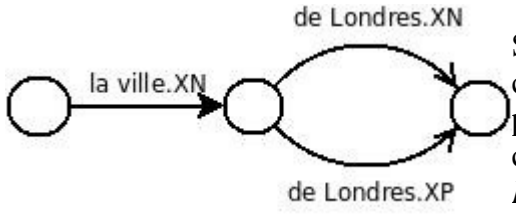
Nous rappelons que les probabilités d'émission et de transition dans un HMM utilisant un modèle trigramme sont respectivement $P(w_i/t_i)=\frac{freq(w_i,t_i)}{freq(w_i)}$ et $P(t_i/t_{i-2}t_{i-1})=\frac{freq(t_{i-2}t_{i-1}t_i)}{freq(t_{i-2}t_{i-1})}$ (voir *II.2 Modèle probabiliste de Markov* pour une description des variables).
Les deux méthodes d'apprentissage que nous avons expérimenté sont identiques à celles utilisées dans notre étiqueteur morpho-syntaxique. Elles effectuent toutes les deux un apprentissage sur le corpus d'apprentissage d'Anne Abeillé, qui est constitué d'environ 250000 chunks. Dans la première

méthode, dite *"first-head"*, nous allons prendre en compte uniquement la tête de chaque chunk, représentant le chunk entier, dans le calcul de la probabilité d'émission. Par exemple, si l'on avait le chunk nominal <*la jolie pomme.XN*>, on calcule les statistiques sur *pomme* ayant l'étiquette XN. La formule de probabilité d'émission devient donc $P(c_i/t_i)=\frac{freq(c_i,t_i)}{freq(c_i)}$.

Sachant que :

- c_i est la tête du ième chunk du corpus d'apprentissage.
- t_i est l'étiquette grammaticale de c_i .
- $freq(c_i,t_i)$ est la fréquence de la paire (tête,étiquette grammaticale).
- $freq(c_i)$ est la fréquence de la tête c_i .
- $freq(t_{i-2},t_{i-1},t_i)$ est la fréquence de la séquence des trois étiquettes grammaticales.
- $freq(t_i)$ est la fréquence de l'étiquette t_i .

Cependant, cette modification des probabilités n'est pas suffisante pour résoudre certaines phénomènes d'ambiguïtés de l'automate. Prenons l'automate ambigü suivant :



Si nous appliquons le principe précédent, l'ambiguïté sur le chunk de Londres sera résolue uniquement avec le nom propre Londres puisque c'est la tête du chunk. Or si dans le corpus, on possède plus de chunks XN avec comme tête *Londres*, de <*Londres.XN*> sera plus probable que <*de Londres.XP*>. Ce qui est faux dans ce cas. En revanche si on prend en compte le premier mot du chunk, c'est à dire *de*, cela peut nous aider à désambiguiser ce cas. L'idée est donc de modifier les probabilités d'émission pour tenir compte du premier mot du chunk. La probabilité d'émission de la transition étiquetée par le chunk est égale à l'addition des probabilités de la tête et du premier mot. Dans l'exemple précédent, on a donc *P(de Londres/XP) = P(de/XP) + P(Londres.XP)* et *P(de Londres/XN) = P(de/XN) + P(Londres/XN)*. Le choix probable se portera donc plus sur <*de Londres.XP*>.

La deuxième méthode, appelée "start-end", effectue le réétiquetage des chunks dans le corpus d'apprentissage. Plutôt que de prendre en compte uniquement les têtes dans chaque séquence de mots (et le premier mot), on va essayer de calculer les probabilités pour chaque mot de la séquence. Pour ce faire, nous allons réétiqueter le chunk de la façon suivante. Le premier mot aura comme étiquette grammaticale, l'étiquette du chunk plus *Start* pour indiquer le début d'un nouveau chunk. Quant à la tête du chunk, on rajoute *End*. Les mots internes au chunk ont simplement l'étiquette du chunk. Par exemple, prenons, le chunk du corpus <*une nouvelle police.XN*>, le réétiquetage produira la suite de mot *une.XNStart*, *nouvelle.XN* et *police.XNEnd*. Le processus d'apprentissage est ensuite aisé puisque pour chaque étiquette grammaticale, on possède un seul mot. La formule de probabilité est identique à celle détaillée au-dessus (sachant que *ci* est un mot du chunk).

VI.4. Evaluations

Toutes les évaluations que nous avons effectué sont basées sur trois critères majeurs. Il y a tout d'abord le rappel et la précision où l'on considère qu'un chunk produit par Pom est valide s'il correspond à celui du corpus d'évaluation, c'est à dire si la séquence de mots associée à sa catégorie grammaticale sont identiques. De plus, nous allons calculer le nombre d'analyses, issues du corpus d'évaluation, qui sont dans l'automate ambigü donné par Pom. De même, nous calculons la couverture des chunks du corpus d'évaluation qui sont dans l'automate ambigu. On pourra ainsi savoir si le découpage effectué par Pom est relativement identique à celui du corpus et si l'analyse est possible. Le premier corpus de comparaison utilisé est le corpus d'évaluation issu du corpus d'Anne Abeillé. Il est composé de 31715 chunks répartis dans 2440 phrases.

L'automate utilisé pour ces évaluations est l'automate élagué par *SPH* et *Luberon*. Nous obtenons

les résultats suivants :

Algorithme de désambiguisation	Méthode d'apprentissage	Rappel	Précision	Analyses possibles	Couverture des chunks
SPH,Luberon	-	65,69	69,11	7,5	74,35
SPH,Luberon,StatProb	first-head	68,68	72,26	7,5	74,35
SPH,Luberon,StatProb	start-end	67,26	71,48	7,5	74,35

Les résultats sont assez faibles, moins de 70%, tant sur le rappel que sur la précision. Nous sommes loin des 80-90% de l'état de l'art. Ce qui est confirmé par le faible taux d'analyse possible du corpus dans l'automate ambigü du texte et la faible couverture des chunks. Le principal problème vient donc du fait que le découpage en chunks du corpus d'Anne Abeillé est différent de celui effectué par Pom. Par exemple, les chunks nominaux tels que *<l'ensemble du territoire.XN>* codé *<l'ensemble.XN><du territoire.XP>* dans le corpus. Pom considère que *<l'ensemble du.XN>* est un "déterminant", ce qui est l'analyse juste. On peut voir en revanche que notre module de désambiguisation donne des meilleurs résultats quelque soit la méthode d'apprentissage utilisée, avec un avantage pour "first-head".

Ce corpus n'est donc pas adapté pour une comparaison et une évaluation de l'étiquetage en chunks effectué par Pom. Nous allons donc prendre un autre corpus d'évaluation, celui crée par Takuya Nakamura et Stavroula Voyatzi. Il est constitué de 3336 chunks répartis dans 453 phrases. Avec le même automate que précédemment, nous obtenons :

Algorithme de désambiguisation	Méthode d'apprentissage	Rappel	Précision	Analyses possibles	Couverture des chunks
SPH,Luberon	-	78,97	73,33	48,79	87,07
SPH,Luberon,StatProb	first-head	81,33	75,32	48,79	87,07
SPH,Luberon,StatProb	start-end	81,25	75,24	48,79	87,07

On peut voir que les résultats sont très nettement meilleurs qu'avec le corpus d'Anne Abeillé. On passe de 70% à environ 80% pour le rappel et la précision à légèrement augmentée. De plus le nombre d'analyses possibles passe à 1/2 des phrases du corpus avec une couverture de chunks élevée. Le corpus correspond donc mieux à l'étiquetage de Pom. Cependant, de nombreux mots composés sont présents dans ce corpus et certains ne sont pas présents dans les dictionnaires de Pom, car ils sont surtout issus du *langage parlementaire* comme les noms composés *association capital travail* et *grille de salaires*. Nous avons donc décidé de les inclure dans un dictionnaire externe et réeffectué le chunking du corpus. Les résultats sont les suivants.

Algorithme de désambiguisation	Méthode d'apprentissage	Rappel	Précision	Analyses possibles	Couverture des chunks
SPH,Luberon	-	81,73	77,86	53,87	88,84
SPH,Luberon,StatProb	first-head	83,54	79,54	53,87	88,84
SPH,Luberon,StatProb	start-end	83,46	79,48	53,87	88,84

Les résultats ont augmentés d'environ 2% ainsi que le nombre d'analyses possibles. De plus on voit que l'étiquetage se rapproche de la couverture maximale des chunks, c'est à dire qu'on obtient environ 5% d'erreurs uniquement par rapport au maximum que l'on pourrait avoir. On est donc

relativement autour de 95% de rappel par rapport à cette limite optimale. Comme énoncé précédemment, nous allons maintenant combiner notre module statistique aux autres modules de d'élagage de Pom (indépendamment), voire sans aucun de ces modules. Nous allons tester quelle est la meilleure combinaison de modules possibles.

Algorithme de désambiguisation	Méthode d'apprentissage	Rappel	Précision	Analyses possibles	Couverture des chunks
Uniquement linéarisation de Pom	-	-	-	58,72	91,03
Uniquement StatProb	first-head	55,10	38,62	58,72	91,03
Uniquement StatProb	start-end	59,81	44,63	58,72	91,03

La linéarisation de Pom dans ce cas de figure ne nous a pas donné de chunks, nous ne pouvons donc l'évaluer. On voit que les résultats ne sont pas bons, comparés aux résultats précédents, le rappel et la précision ont baissé d'environ 25%. En revanche, La méthode *start-end* est meilleure que *first-head*. La raison principale de ces mauvais résultats vient sûrement du fait qu'il y a trop d'ambiguïtés dans l'automate pour obtenir un chunking correct, le nombre d'analyses possibles ayant légérement augmenté avec +1,42%.

Algorithme de désambiguisation	Méthode d'apprentissage	Rappel	Précision	Analyses possibles	Couverture des chunks
Luberon	-	-	-	55,63	90,14
Luberon,StatProb	first-head	55,89	38,73	55,63	90,14
Luberon,StatProb	start-end	62,22	46,95	55,63	90,14

Les remarques sont équivalentes aux précédentes. L'utilisation du module Luberon n'a que peu d'impact sur le chunking de Pom, environ +3% avec *start-end*.

Algorithme de désambiguisation	Méthode d'apprentissage	Rappel	Précision	Analyses possibles	Couverture des chunks
SPH	-	76,14	72,49	55,19	89,40
SPH,StatProb	first-head	82,30	78,34	55,19	89,40
SPH,StatProb	start-end	81,18	77,29	55,19	89,40

L'utilisation du module SPH améliore grandement les résultats. StatProb fait de même avec environ +6% de chunks pertinents en plus. On remarque que *first-head* a les meilleurs résultats lorsqu'il y a moins d'ambiguïtés dans l'automate.

Nous savons maintenant que le chunker optimal est celui qui utilise les modules désambiguisation SPH, Luberon et StatProb avec un apprentissage *first-head*. Regardons quels sont les résultats en focalisant sur les types de chunks les plus importants :

Chunk	Chunks trouvés	Total dans le corpus	Pourcentage
XN	777	867	89,62%
XV	708	799	88,61%

XP	516	605	85,23%
XADV	200	344	58,12%
XA	118	158	74,68%
det, conjc, conjs...	398	457	87,32%

On voit que les chunks fréquents dans les phrases, XN, XV et XP sont bien étiquetés avec environ 90% des chunks, contrairement aux XA et XADV. La plupart des XADV du corpus non trouvées par Pom sont des expressions semi-figées de temps, par exemple *à minuit précis* ou encore *à des heures chronométriquement déterminées*. Ces adverbes sont en général découpées en deux chunks par Pom, *<à minuit.XP><précis.XA>* et *<à des heures.XP><chronométriquement déterminées.XV>*. Au niveau des XA, on remarque quelques formes adverbes suivis d'un adjectif non détectées par Pom comme *invariablement créditeur, <invariablement.XADV><créditeur.XA>* dans l'étiquetage de Pom.

La table suivante montre les résultats de la couverture des chunks du corpus d'évaluation qui sont dans l'automate ambigü avant linéarisation :

Chunk	Chunks trouvés	Total dans le corpus	Pourcentage
XN	810	867	93,31%
XV	726	799	90,97%
XP	530	605	87,75%
XADV	216	344	62,42%
XA	138	158	87,34%
det, conjc, conjs...	400	457	89,30%

Les résultats sont relativement équivalents à la table précédente, ce qui veut dire que notre algorithme de désambiguisation statistique probabiliste associé à *Pom* s'approche de la couverture des chunks qui sont à la fois dans l'automate et dans le corpus.

Conclusion des évaluations

En conclusion, nous avons pu voir que l'application d'un module de désambiguisation statistique probabiliste appliqué sur l'automate ambigü produit par le super-chunker *Pom* améliore les résultats. Deux méthodes d'apprentissage du module ont été testées *first-head* et *start-end*. Elles nous donnent environ les mêmes performances. L'heuristique des plus courts chemins SPH déjà testée pour l'étiqueteur morpho-syntaxique est très performante car on passe d'un rappel et d'une précision de 55% à 80%. Le module de désambiguisation statistique augmente peu le rappel et la précision lorsque SPH n'est pas appliqué sur l'automate ambigü (avec ou sans Luberon), avec un avantage pour la méthode *start-end*. Globalement le super-chunker avec les différents modules de désambiguisation nous donnent des résultats similaires à ceux de l'état de l'art pour certains types de chunks (XN, XV, XP, ...) et d'autres moins bien détectés (XA, XADV, ...). De plus, la couverture des chunks est presque atteinte avec *StatProb*, ce qui est contrasté par le pourcentage moyen d'analyses du corpus présentes dans l'automate ambigü (~50% des phrases du corpus).

VII. Perspectives

En ce qui concerne l'analyse morpho-syntaxique faite par notre étiqueteur hybride, nous prévoyons d'associer les lemmes aux mots avec leur étiquette morpho-syntaxique, en sortie de l'étiquetage. Les performances devraient rester inchangées car pour un mot donné, nous n'avons qu'en général un seul lemme. Ensuite, nous avons vu au cours des évaluations, que ELAG améliorait peu l'étiquetage. Il faudrait voir à écrire d'autres règles lexicales et contextuelles à la main. Et enfin, en terme d'évaluation, il serait intéressant de tester notre étiqueteur sur l'Anglais, sur le corpus Penn TreeBank par exemple. Ceci toujours dans le but de comparer avec d'autres étiqueteurs.

Pour le chunking, l'idée serait d'implémenter un chunker basé sur l'approche utilisant un texte étiqueté au préalable avec des étiquettes morpho-syntaxique puis segmenté en chunks. Cette analyse morpho-syntaxique pourrait être effectuer avec notre étiqueteur. Ensuite, il faudrait le comparer avec *Pom* utilisant notre module de désambiguisation statistique. De même, on pourrait transformer *Pom* en chunker standard et comparer avec le corpus d'évaluation d'Anne Abeillé dont on aurait retirer les mots composés. Ca nous permettrait de comparer vraiment à l'état de l'art.

VIII. Annexes

VIII.1. Jeu d'étiquettes

Jeu d'étiquettes de l'étiqueteur morpho-syntaxique

Les étiquettes suivantes sont à la fois dans le corpus annoté du Français et utilisées par notre étiqueteur morpho-syntaxique. L'étiqueteur transforme certaines étiquettes du corpus annoté pour les harmoniser au maximum avec le tagset d'Unitex.

Le jeu d'étiquettes est extrait automatiquement du corpus d'apprentissage, ce n'est donc qu'une estimation :

- simple (catégories grammaticales uniquement) : 11 étiquettes
- complexe (catégories + informations morphologiques) : 116 étiquettes

Les repères sont :

- *u* pour étiquette utilisée dans Unitex (et donc par l'étiqueteur)
- *c* pour étiquette utilisée dans le corpus
- si présente dans les deux, aucune indication

Les étiquettes sont (hors étiquettes de ponctuations) :

- N noms communs
- ET(c)/N+Pr(u) noms propres (ET est plutôt ambigüe)
- ADV adverbes
- D(c)/DET(u) déterminants
- CL(c)/PRO(u) pronoms
- V verbes
- A adjectifs
- C conjonctions
- P(c)/PREP(u) prépositions
- PREF(c)/PFX(u) préfixes
- INT(c)/I(u) interjections

Et pour certaines catégories grammaticales, on ajoute des traits morphologiques à l'étiquette (dans le cadre d'un jeu d'étiquettes complexe) :

- genre, pour N, V, A, DET, PRO :
  - masculin *m*
  - féminin *f*
- nombre, pour N, V, A, DET, PRO :
  - singulier *s*
  - pluriel *p*

- temps, pour V :
    - présent *P*
    - imparfait *I*
    - ...
- personne, pour V, PRO :
    - première *1*
    - deuxième *2*
    - troisième *3*

Dans le cadre de l'expérience *AddPaths* (voir *III.2. AddPaths*), les étiquettes morpho-syntaxiques (et en particulier les catégories grammaticales) sont précédées de :

- N+, pour indiquer que le token fait parti d'un nom composé
- NEnd+, pour indiquer le token fait parti d'un nom composé et que c'est le dernier de celui-ci.

Jeu d'étiquettes du super-chunker

Les étiquettes suivantes sont à la fois dans le corpus annoté du Français et utilisées par notre le super-chunker *Pom*. Nous transformons certaines étiquettes pour harmoniser les chunks. Le jeu d'étiquettes est constitué de 10 étiquettes de chunk.

Les repères sont :

- *p* pour étiquette utilisée dans *Pom*
- *c* pour étiquette utilisée dans le corpus

Les étiquettes sont (hors étiquettes de ponctuations) :

- XN(p)/NP(c) chunk nominal
- XP(p)/PP(c) chunk prépositionnel
- XV(p)/VN(c) chunk verbal
- XA(p)/AP(c) chunk adjectival
- XADV(p)/Adj(c) chunk adverbal
- XVP(p)/VPinf(c) chunk verbal infinitif
- conjc(p)/COORD(c) conjonction de coordination
- conjs(p)/Ssub(c) conjonction de subordination
- pro(p)/Srel(c) pronom
- prep(c)/P(c) préposition

Dans le cadre de l'expérience de réétiquetage des chunks (voir *VI.3 méthodes d'apprentissage*), les étiquettes des premier et dernier mot du chunk sont terminées par :

- Start, pour indiquer que le mot est le premier mot d'un chunk.
- End, pour indiquer que le mot est le dernier mot du chunk.

VIII.2. Graphes Unitex reconnaissant les mots composés

Dans le cadre de l'enrichissement du corpus du français d'Anne Abeillé (voir *II.5.1 enrichissement du corpus*), nous avons décidé de créer des graphes Unitex permettant de donner à chaque token-mot interne à un mot composé, ses informations morpho-syntaxiques.

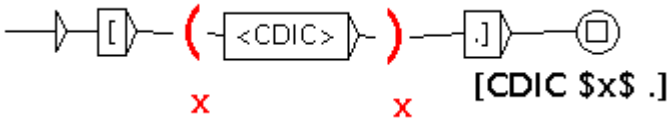
Nous allons prendre en exemple deux mots composés issus du corpus annoté. Ils sont mis dans un texte avec un mot composé par ligne et donné en entrée d'Unitex :

1. [régime de retraite .]
2. [inspecteur d'académie .]

Les signes de ponctuations servent à délimiter le mot composé.

1ère étape, repérage des formes composées:

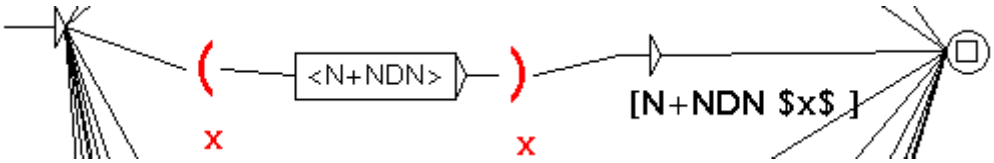
Nous appliquons un premier graphe qui va tenter de reconnaître les formes composées en entrée. Si un mot composé est dans le ou les dictionnaires de mots composés utilisés par Unitex, il est repéré et nous mettons alors des balises "[CDIC" avant et "]" après le mot composé.



- *régime de retraite* est dans le Delacf donc en sortie nous aurons *[CDIC régime de retraite]*
- *inspecteur d'académie* n'est pas dans le Delacf donc aucuns changements [inspecteur d'académie]

2ème étape, repérage des catégories grammaticales :

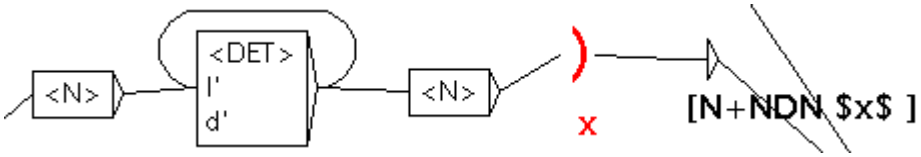
Nous appliquons un deuxième graphe qui va tenter de reconnaître, de quelle catégorie grammaticale et de quelle sous-catégorie grammaticale, sont les formes composées reconnue. En sortie, nous obtenons un nouveau balisage indiquant la catégorie et la structure interne du mot composé courant.



Le mot *régime de retraite* est reconnu en tant que *N+NDN* donc nous aurons *[N+NDN régime de retraite]*

Pour les formes composées non reconnues lors de la première étape, nous avons crée des graphes

spécifiques à chaque catégorie et pour chaque structure interne (dans la mesure où elles ont été observées dans des mots composés du corpus d'apprentissage).

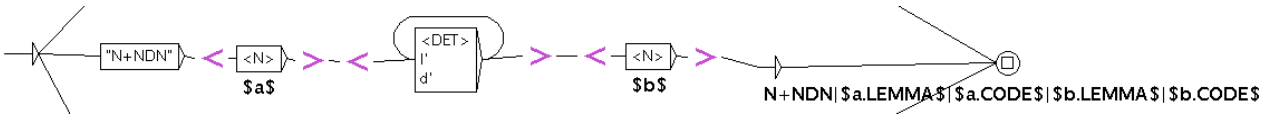


Le mot *inspecteur d'académie* est reconnu par ce graphe donc nous aurons *[N+NDN inspecteur d'académie]*.

3ème étape, extraction des traits morpho-syntaxiques :

Nous appliquons un troisième graphe qui va tenter d'extraire pour chaque token-mot simple de chaque mot composé, les traits morpho-syntaxiques (N, masculin, pluriel, ...).

Ceci est effectué grâce au mode morphologique d'Unitex.



- *[N+NDN régime de retraite]* est reconnu par ce graphe donc nous aurons le découpage en tokens-mots suivant :
  - régime, catégorie: N, traits: ms, lemme: régime
  - de : DET, traits: indéfinis, lemme: de
  - retraite, catégorie: N, traits: fs, lemme: retraite
- *[N+NDN inspecteur d'académie]* est reconnu par ce graphe donc nous aurons le découpage en tokens-mots suivant :
  - inspecteur, catégorie: N, traits: ms, lemme: inspecteur
  - d' : DET, traits: indéfinis, lemme: de
  - académie, catégorie: N, traits: fs, lemme: académie

En sortie de ces différentes phases d'application de graphes, nous obtenons un ensemble de mots composés reconnus découpés en tokens avec leurs traits morpho-syntaxiques avec lesquelles nous faisons un dictionnaire de mots composés pour Unitex.

IX. Bibliographie

Eric Laporte and Anne Monceaux. 1999.

"Elimination of lexical ambiguities by grammars. The ELAG system"

In XXII, Amsterdam-Philadelphie : Benjamins, pp. 341-367.

Eric Brill. 1995.

"Transformation-Based Error-Driven Learning and Natural Language

Processing: A case study in part-of-speech tagging "

Thorsten Brants. 2000.

"TnT - A statistical part-of-speech tagger".

Helmut Schmid. 1995.

"Probabilistic part-of-speech tagging using decision trees"

Kristina Toutanova, Dan Klein, Christopher D. Manning and Yoram Singer. 2003.

"Feature-rich part-of-speech tagging with a cyclic dependency network"

In HLT-NAACL 2003, Edmonton : pp. 173-180.

Libin Shen, Giorgio Satta and Aravind K. Joshi. 2007.

"Guided learning for bidirectional sequence classficiation"

Joshua Kupiec. 1992.

"Robust part-of-speech tagging using a hidden Markov model"

In Computer speech and language 6.

Andrew J. Viterbi. 1967.

"Error bounds for convolutional codes

and an asymptotically optimum decoding algorithm"

In IEEE Transactions on Information Theory 13(2) : pp. 260-269.

Scott M. Thede and Mary P. Harper. 1999.

"A second-order Hidden Markov Model for part-of-speech tagging"

In proceedings of the 37th annual meeting of the Association for Computational

Linguistics on Computational Linguistics, College Park, Maryland : pp. 175-182.

Anne Abeillé, Lionel Clément and Fabien Toussenel. 2001.

"A treebank for French: some experimental results".

In Corpus Linguistics Conference, CL01, Lancaster.

Violeta Seretan and Luka Nerima and Eric Wehrli. 2004.

"Multi-word collocation extraction by syntactic composition of collocation bigrams"

In Recent Advances in Natural Language Processing III : pp. 91–100.

Amalia Todirascu and Christopher Gledhill. 2008.

"Collocations en contexte : extraction et analyse contrastive".

Vladimir Vapnik and Corinna Cortes. 1995.

"Support-Vector Networks"

In Machine Learning, 20.

Jesus Gimenez and Lluis Marquez . 2004.

"SVMTool: A general POS tagger generator based on Support Vector Machines"

Edsger W. Dijkstra. 1959.

"A short introduction to the art of programming"

Franck A. Smadja. 1993.

"Retrieving collocations from text : Xtract"

In Computational Linguistics, 19(1) : pp. 143-177.

Steven Abney. 1996.

"Partial parsing via finite-state cascades"

J. of Natural Language Engineering, 2(4) : pp. 337-344.

Olivier Blanc, Matthieu Constant and Patrick Watrin. 2007.

"Segmentation in super-chunks with a finite-state approach"

Wojciech Skut and Thorsten Brants. 1997.

"A maximum entropy partial parser for unrestricted text"

Antonio Molina and Ferran Pla. 2002.

"Shallow parsing using specialized HMMs"

J. of Machine Learning Research 2 : pp. 595-613.

Blandine Courtois. 1990.

"Un système de dictionnaires électroniques pour les mots simples du français"

Langue Française 87 : pp. 11–22 .

Blandine Courtois, Mylène Garrigues, Gaston Gross, Maurice Gross, René Jung, Michel Mathieu-Colas, Max Silberztein, Robert Vivès. 1997.

″Dictionnaire électronique des noms composés DELAC : les composants NA et NN″

Rapport Technique du LADL 55, Paris, Université Paris 7.

Ian C. Ross and John W. Tukey. 1975.

"Introduction to these volumes"

In Index to Statistics and Probability : pp. 4-10.

Didier Bourigault. 1992.

"Surface grammatical analysis for the extraction of terminological noun phrases"

In COLING-92, Vol. III : pp. 977–981.

Kenneth Church. 1988.

"A stochastic parts program and noun phrase parser for unrestricted texts"

In Proceedings of the Second Conference on Applied Natural Language Processing.

Aravind Joshi. 1996.
"A Parser from Antiquity: An Early Application of Finite State Transducers to Natural Language Parsing"
In Proceedings ECAI '96 workshop on "Extended finite state models of language".
Aravind Joshi and B. Srinivas. 1994.
"Disambiguation of super parts of speech (or supertags): Almost parsing."
In COLING-94.
Kimmo Koskenniemi, Pasi Tapanainen, and Atro Voutilainen. 1992.
"Compiling and using finite-state syntactic rules"
In Proceedings of the Fourteenth International Conference on Computational Linguistics COLING-92 vol. I : pp. 156-162.
Kimmo Koskenniemi.
"Finite-state parsing and disambiguation"
In COLING-90 : pp : 229–232.
Kimmo Koskenniemi, Pasi Tapanainen, and Atro Voutilainen. 1992.
"Compiling and using finite-state syntactic rules."
In COLING-92 : pp. 156–162.
Atro Voutilainen and Pasi Tapanainen. 1993.
"Ambiguity resolution in a reductionistic parser"
In Proceedings of the Sixth Conference of the European Chapter of the Association for Computational Linguistics : pp. 394-403, Utrecht.
Aït-Mokhtar and J.P. Chanod. 1997.
"Incremental Finite-State Parsing."
In Proceedings of the 5th Conference on Applied Natural Language Processing, Washington D.C., USA.
Nivre and Nilsson. 2004.
"Multiword units in syntactic parsing."
In Dias, G., Lopes, J.G.P., Vintar, S., eds.: Proceedings of the Workshop on Methodologies and Evaluation of Multiword Units in Real-World Applications, Language and ressource Evaluation Conference : pp. 39–46.