

Institut de la Francophonie pour l'Informatique

Rapport de stage de fin d'études

**MODULE D'EXTRACTION FOCALISE ET
ANALYSE AUTOMATIQUE
LINGUISTIQUE DU WEB**

NGUYEN Hong San

Janvier 2007

Lieu de stage : **Institut de Recherche en
Informatique de Toulouse (IRIT)**
Période de stage : du **15/03/2006** au **30/09/2006**
Tuteur de stage : **Bruno GAUME**

Remerciements

Je tiens à remercier tout particulièrement Monsieur Bruno GAUME, tuteur de stage et professeur de l'Université Paul Sabatier, qui m'a accueilli de faire ce stage dans l'IRIT et m'a dirigé mon travail de recherche. Il m'a aussi donné des conseils dans le domaine de recherche et ainsi ceux dans la vie quotidienne.

Je remercie aussi Franck SAJOUS, ingénieur de l'Équipe de Recherche en Syntaxe et Sémantique pour son soutien technique pendant mon stage.

Je tiens également à exprimer toute ma sympathie à Alain MONIER pour ses aides précieuses dans la démarche de mon séjour à Toulouse.

J'adresse en fin mes reconnaissances aux professeurs de l'Institut de la Francophonie pour l'Informatique, pour m'avoir aidé à effectuer ce stage.

Résumé

Ce stage se déroule dans un cadre d'une collaboration entre l'Institut de Recherche en Informatique de Toulouse (IRIT) et l'Équipe de Recherche en Syntaxe et Sémantique (ERSS). Notre objectif est de développer un outil informatique pour la construction automatique des corpus à partir du web en utilisant les outils analyse linguistique existés. Il s'agit de la construction d'un crawl focalisé du web et de l'intégration des outils d'analyse linguistique pour analyser les pages Web. Dans un premier temps, nous présentons un modèle de crawl focalisé qui parcourait le Web pour télécharger les pages concernées à un sujet spécifique. Le crawl doit faire sortir deux résultats importants: les contenus textuelle des pages Web et le graphe des hyperliens des pages Web. Dans un deuxième temps, nous faisons une études sur les outils d'analyse linguistique TreeTagger, Syntex et Upery et les intégrons dans le système pour l'analyse des pages Web. Nous effectuons aussi le prétraitement des textes récupérés par le crawl avant de les passer à des outils linguistique. Le résultat final est des corpus analysés qui parlent d'un sujet spécifique.

Abstract

This internship proceeds within a collaboration between the IRIT and ERSS. Our objective is to develop a tool for the automatic construction of the corpus from the Web by using the existed tools of linguistic analysis. It is about construction of a focused crawler of the Web and integration of the linguistic tools to analyze the Web pages. Initially, we present a focused crawler model which traversed the Web to download the pages concerned on a specific topic. The crawler must give two important results: contents textual of Web pages and graph of hyperlinks of Web pages. In the implementation of crawler, we pay attention at all the technical problems: constitution of the starting germ, parallelism, strategies of crawl, politeness and spider trap. In the second time, we study the tools for linguistic analysis TreeTagger, Syntex and Upery and integrate them in the system for the analysis of the Web pages. We carry out also the pretreatment of the texts recovered by the crawl before passing to linguistics tools. The final result is analyzed corpus which speaks about a specific topic.

Table des matières

Introduction.....	1
1. Environnement de stage.....	1
1.1. IRIT	1
1.2. ERSS.....	2
2. Problématique	2
2.1. Crawl focalisé	2
2.2. Graphes du Web.....	4
2.3. Analyse linguistique.....	4
3. Objectif du stage.....	5
4. Organisation du rapport.....	7
Crawl du Web.....	8
1. Introduction	8
2. Définitions	8
3. Architecture générale	9
3.1. Architecture de 2-modules.....	10
3.2. Architecture de 4-modules.....	10
3.3. Algorithme de crawl.....	12
4. Stratégies de crawl	12
5. Respect de la politesse	15
Construction du crawl focalisé	17
1. Suppositions et notations	17
1.1. Page Web	17
1.2. Germe de départ.....	18
1.3. Graphes.....	19
2. Constitution du germe de départ	20

3. Architecture	21
3.1. Composantes.....	21
3.2. Base de données	33
3.3. Interface d'utilisateur.....	34
4. Environnement de programmation et dépendances	35
Analyse linguistique des pages Web	37
1. Outils d'analyse linguistique	37
1.1. TreeTagger.....	37
1.2. Syntex	38
1.3. Analyse syntaxique en dépendance.....	38
1.4. Construction du réseau de syntagmes.....	46
1.5. Upery	48
2. Intégration	52
Conclusion	54
1. Résultat obtenu	54
2. Conclusion.....	54
Bibliographie.....	56
Annexe.....	58

Liste des figures

Figure 1: Diagramme des modules du stage.....	6
Figure 2: Architecture de 2-modules	10
Figure 3: Architecture de 4-modules	11
Figure 4: Architecture du crawl	22
Figure 5: Queue de deux niveaux: S1, S2,... sont les sites Web et Px.y est la y ^{ième} page de site x.....	23
Figure 6: Parallélisme.....	24
Figure 7: Liens dans le frameset	28
Figure 8: Liens dans les images mappées	28
Figure 9: Exemple du calcul de la profondeur	33
Figure 10: Interface d'utilisateur 1	34
Figure 11: Interface d'utilisateur 2.....	35
Figure 12: Exemple de l'analyse syntaxique.....	38
Figure 13: Relation de dépendance syntaxique	39
Figure 14: Contrainte 1	39
Figure 15: Contrainte 2	39
Figure 16: Quelques relations principales	40
Figure 17: Algorithme DET.....	40
Figure 18: Algorithme PREP-d	41
Figure 19: Algorithme OBJ	41
Figure 20: Algorithme SUJ	42
Figure 21: Ambiguïté de rattachement des adjectifs	42
Figure 22: Algorithme ADJ: recherche des candidats.....	43
Figure 23: Algorithme ADJ: sélection d'un candidat	44
Figure 24: Ambiguïté de rattachement des prépositions	44

Figure 25: Sélection de candidat par arg	46
Figure 26: Exemple d'extraction des syntagmes	47
Figure 27: Réseau terminologie	47
Figure 28: Réseau terminologie dans un corpus entier.....	48
Figure 29: Exemple de normalisation.....	48
Figure 30: Exemple de la productivité.....	51
Figure 31: Exemple de prox: $\text{prox}(\text{détresse respiratoire}, \text{syndrome}) = 1,10$	51

Chapitre 1

INTRODUCTION

Ce rapport décrira les problèmes autour de mon stage de fin d'études à l'Institut de Recherche en Informatique de Toulouse (IRIT). Le stage se divise en deux parties: **la construction de crawl du Web, l'étude et l'intégration avec les outils de traitement automatique linguistique**. Ce chapitre donnera une vue générale du stage.

1. Environnement de stage

Le stage se déroule grâce à une collaboration entre l'IRIT, et l'ERSS, Équipe de Recherche en Syntaxe et Sémantique. Cette session abordera dans les grandes lignes l'introduction de l'IRIT et l'ERSS.

1.1. IRIT

L'IRIT est une **Unité Mixte de Recherche**, UMR 5505, commune au Centre National de la Recherche Scientifique (CNRS), à l'Institut National Polytechnique de Toulouse (INPT), à l'Université Paul Sabatier (UPS) et à l'Université des Sciences Sociales Toulouse 1 (UT1).

L'IRIT, créé en 1990, représente l'un des plus forts potentiels de recherche en informatique en France, fédérant plus de 190 chercheurs et enseignants chercheurs, relevant non seulement de ses tutelles mais aussi de l'Université Toulouse Le Mirail (UTM).

Les objectifs que l'IRIT se donne sont à la mesure de sa taille, tant sur le plan de la recherche que sur le plan de la formation et du transfert technologique. La diversité des thèmes scientifiques couverts - héritée d'une longue histoire : Toulouse a été l'une des villes pionnières de l'informatique française - permet d'élaborer des projets ambitieux et de répondre à la forte demande du monde socio-économique. Cette

diversité au sein de l'Institut constitue un très important foyer de multidisciplinarité et de complémentarité.

1.2. ERSS

L'ERSS est une unité mixte de recherche (UMR 5610) sous la double tutelle du CNRS et du Ministère de l'Education et de la Recherche. Elle est implantée sur deux sites : l'Université de Toulouse-Le Mirail et l'Université Michel de Montaigne à Bordeaux.

Depuis sa fondation en 1981, l'ERSS se donne pour fin la description scientifique des langues dans leurs différentes composantes (phonologie, morphologie, syntaxe, sémantique, pragmatique, lexicque) et la modélisation des descriptions obtenues, cette activité modélisatrice donnant lieu à des collaborations tant avec les informaticiens (spécialistes de l'intelligence artificielle et de l'ingénierie linguistique) qu'avec les psycholinguistes. Les langues étudiées sont multiples : au français commun - auquel est consacrée la majorité des travaux de l'équipe -, au latin, à l'anglais, à l'espagnol, au coréen et au japonais, sont venus s'ajouter par exemple au cours des quatre dernières années l'arabe et l'amharique, le barasana et le tatuyo, le sarde, l'italien et le serbocroate.

2. Problématique

2.1. Crawl focalisé

Comme la taille entière du Web est trop large et ne cesse pas d'augmenter, même un grand moteur de recherche ne peut couvrir qu'une petite partie du contenu de Web. Selon une étude de Lawrence et Giles (Lawrence and Giles, 2000), aucun moteur de recherche n'indexe plus de 16% du Web. Pour la raison de l'explosion de la taille du Web, les moteurs de recherche deviennent de plus en plus importants comme un moyens primaires de localiser l'information sur Web. Les moteurs de recherche se fondent sur les collections massives de pages Web qui sont acquises à l'aide des **crawl du Web**. Le crawl parcourt le Web en suivant les hyperliens et en stockant une copie des pages visité dans une grande base de données. Dans les quelques dernières années, plusieurs travaux académiques et industrielles ont été portés sur la technologies de recherche d'information sur Web, composant les stratégies de crawl,

le stockage, l'indexation, et quelques techniques dans l'analyse de la structure du Web et du graphe de Web.

La majeure partie des travaux récents sur les stratégies de crawl n'adresse pas du tout les questions des performances, mais essaye de minimiser le nombre de pages qui ont besoin de télécharger, et maximiser les bénéfices obtenus à partir des pages téléchargées. Cette stratégie convient bien aux applications qui ont seulement la largeur de bande très limitée. Cependant, dans le cas d'un plus grand moteur de recherche, on a besoin de combiner la bonne stratégie de crawl et la conception optimisée de système.

Dans ce travail, nous n'avons pas l'intention de développer un crawl de « grand public », ou un **crawl exhaustif**, comportant un très grand nombre de pages, mais nous concentrons sur une technique de crawl, le **crawl focalisé** ou **crawl ciblé**, qui focalise sur quelques type de page, par exemple, les pages d'un domaine particulier ou en une langue particulière, les images, les fichiers mp3, ou les articles scientifiques. L'objectif de crawl focalisé est de chercher un grand nombre de pages intéressées sans utiliser une grande largeur de bande. Alors, la plupart des travaux précédents sur le crawl focalisé n'utilise pas un crawl à haute performance.

Le crawl commence son exécution par une liste des URLs initiaux, ou un **germe de départ**. Le germe de départ est établi selon chaque stratégie de crawl. Dans notre travail, nous utilisons les moteurs de recherche générale comme Google, Yahoo, Alta Vista... pour construire le germe de départ. Le crawl présenté dans ce rapport sera intégré avec les outils de traitement de la langue naturelle afin de construire les corpus d'un domaine particulier. L'utilisateur doit d'abord définir les critères de recherche qui contiennent les mots clés du domaine intéressé, la langue utilisée, les moteurs de recherche générale, la formule propositionnelle... Puis, le crawl lance la recherche sur les moteurs de recherche choisis pour récupérer la liste des URLs de départ. A partir de la liste des URLs de départ, ou le **germe de départ**, le crawl déclenche en suite les agents de recherche pour continuer à chercher les pages pertinentes sur la toile.

Avant d'être enregistrée dans le disque local, la page est prétraitée. Si la page est en HTML, le crawl est chargé de nettoyer toutes les balises HTML et d'extraire le texte clair de la page. Le texte clair est prêt pour les étapes d'analyse linguistique suivantes. Dans le cadre de ce travail, seulement les fichiers HTML et texte sont téléchargés et

indexés. Nous ne traitons pas les autres types de document comme : PDF, Microsoft Word, RTF,...

2.2. Graphes du Web

Le Web est souvent considéré comme des graphes dont les noeuds sont des pages et les arcs sont les relations entre les pages Web comme le hyperlien ou la similarité entre les pages. La construction des graphes du Web est utile pour la stratégie de crawl, de recherche, ou la découverte de la communauté et le phénomène sociologique qui détermine l'évolution du Web.

Dans ce travail, nous nous intéressons à deux types de graphes du Web : **graphe des hyperliens** et **graphe de similarité**. Le **graphe des hyperliens** peut être construit toute de suite pendant le processus de crawl. Chaque page est un noeud et il y a un arc entre deux pages si une page contient le hyperlien vers l'autre. Ce graphe est simple et le moins coûteuse.

Le **graphe de similarité** est déterminé par la similarité entre des pages. Il existe un arc entre deux pages si la similarité de deux pages ne dépasse pas un seuil prédéterminé. Ce graphe est construit après l'étape d'analyse linguistique des pages. Alors, la construction de ce graphe est très coûteuse mais utile pour la recherche d'information.

On peut considérer ces deux graphes comme deux aspect : **physique** et **logique**. Le graphe des hyperliens est comme un graphe physique du Web et le graphe de similarité est le graphe logique du Web. En analysant les deux graphes, on trouvera les caractéristiques de la structure du Web. Par exemple, on peut comparer le graphe des hyperliens avec le graphe de similarité. S'ils sont similaires, on peut exploiter le graphe des hyperliens comme un graphe de similarité mais avec le coût beaucoup moins cher.

2.3. Analyse linguistique

Le Web est toujours le plus grand corpus pour la recherche linguistique. Il fournit un grand nombre des pages dans tous les domaines et en plusieurs langues. Dans ce travail, nous voulons construire des corpus en français dans les domaines particuliers à partir du Web avec l'aide de crawl focalisé.

Après le nettoyage des balises HTML, les pages sont traitées par un étiqueteur morphosyntaxique. Nous utilisons l'étiqueteur TreeTagger développé par l'Université de Stuttgart. Cet étiqueteur supporte plusieurs langues: l'anglais, l'allemand, l'espagnol,... et aussi le français. La sortie est une liste des mots avec les étiquettes correspondants.

Le résultat obtenu par TreeTagger est en suite traité par des outils linguistiques développés par l'ERSS: Syntex et Upery. L'analyseur syntaxique de corpus Syntex effectue l'analyse en dépendance de chacune des phrases du corpus, puis construit un réseau de mots et syntagmes, dans lequel chaque syntagme est relié à sa tête et à ses expansions. A partir de ce réseau, le module d'analyse distributionnelle Upery construit pour chaque terme du réseau l'ensemble de ses contextes syntaxiques. Les termes et les contextes syntaxiques peuvent être simples ou complexes. Le module rapproche ensuite les termes, ainsi que les contextes syntaxiques, sur la base de mesures de proximité distributionnelle. L'ensemble de ces résultats est utilisé comme aide à la construction d'ontologie à partir de corpus spécialisés.

3. Objectif du stage

Ce stage est une partie dans un grand projet de l'IRIT. L'objectif principal de ce stage est de développer un crawl focalisé afin de construire les corpus de texte pour la recherche linguistique. D'autre côté, ce stage demande une étude sur les outils linguistique et l'intégration de ces outils dans le système.

On peut voir dans la Figure 1 le diagramme des modules dans le système que nous allons développer dans ce stage. Dans la première partie du stage, nous construirons le "crawl focalisé", le module principal du stage. Et puis, les outils d'analyse linguistique vont être étudiés et intégrés dans le système. L'automatisation 1 et 2 sont développées pendant la construction de crawl focalisé. L'automatisation 3 est l'intégration de TreeTagger dans le système et l'automatisation 5 est l'intégration de Syntex et Upery dans le système. L'automatisation 4 est un module qui a déjà développé par IRIT pour construire le graphe de similarité à partir de la matrice de fréquence des mots, une matrice de deux dimensions $M(i,j)$, l'une est les documents et l'autre est les mots, les valeurs sont les fréquences des mots i dans le document j . La mission de ce stage est de faire sortir la matrice de fréquence des mots à partir des

pages Web récupérées. Cette tâche est aussi effectuée pendant la construction de crawl.

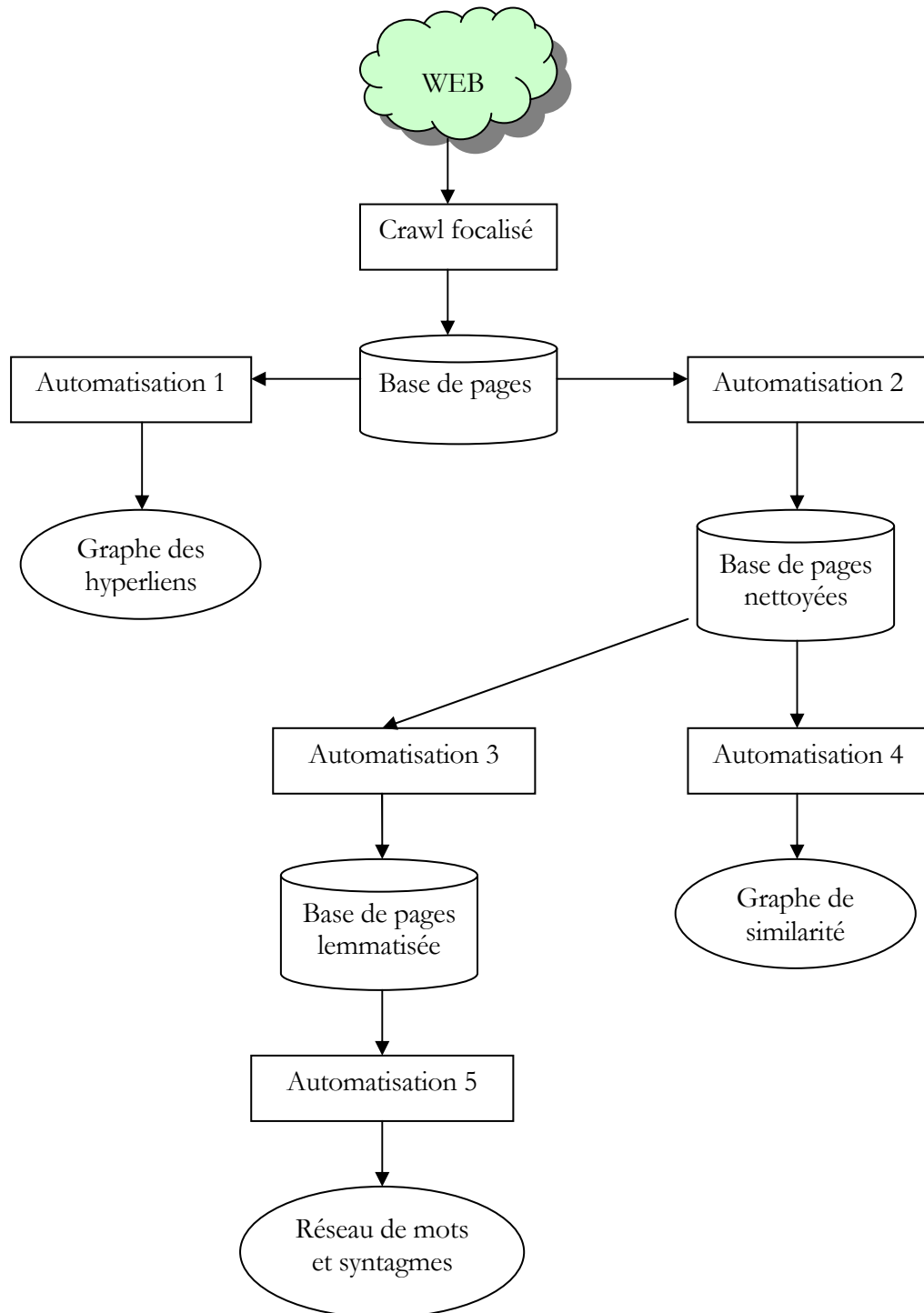


Figure 1: Diagramme des modules du stage

4. Organisation du rapport

Globalement, le rapport se divise en trois chapitres principaux. Après un petit chapitre d'introduction (ce chapitre), on fait ensuite une étude sur tous les problèmes de crawl du Web dans le chapitre 2. Ce chapitre présentera l'architecture générale, l'algorithme de crawl et aussi les stratégies dans la conception d'un crawl du Web.

Puis, le chapitre 3 donnera une vue détaillée sur notre approche de crawl focalisé. En se basant sur les analyses dans le chapitre 2, nous proposerons une architecture de notre crawl pour adapter à l'objectif de ce stage. Tous les problèmes de la conception et les problèmes techniques sont abordés. Nous concentrons sur la conception de crawl: les composants, les mécanismes, la structure de données...

Dans le chapitre suivant, le chapitre 4, l'étude des outils linguistiques et les problèmes techniques de l'intégration ces outils dans le système sont présentés. On parlera de le fonctionnement des outils linguistiques: TreeTagger, Syntex, Upery et aussi de le prétraitement de texte, de l'intégration.

Dernièrement, la conclusion va faire un résumé des résultats obtenus et donner l'évaluation de notre travail.

Chapitre 2

CRAWL DU WEB

1. Introduction

Le crawl du Web est un programme informatique qui explore le Web d'une façon méthodique et automatisée. Les crawls du Web sont principalement employés pour créer une copie de toutes pages visitées pour traiter plus tard par un moteur de recherche, celui classera les pages téléchargées pour fournir des recherches rapides.

Les crawls sont utilisés largement aujourd'hui. Les crawls pour les moteurs de recherche (par exemple, AltaVista, INFOSEEK, Excite, et Lycos) essayent de visiter la plupart des pages Web des textes, afin d'établir des index de contenu. D'autres crawls peuvent également visiter beaucoup de pages, mais peuvent regarder seulement pour certains types d'information (par exemple, adresses email). Et il existe aussi des crawls qui détectent des pages d'intérêt d'un utilisateur particulier, afin d'établir une cache d'accès rapide (par exemple NetAttche).

La conception d'un bon crawl présente beaucoup de défis. Extérieurement, le crawl doit éviter la surcharge des sites Web ou des liens de réseau pendant son parcours [12]. Intérieurement, le crawl doit traiter les volumes de données énormes. Pour la raison des ressources informatiques illimitées et le temps illimité, le crawl doit soigneusement décider quel URLs à visiter et dans quel ordre. Il doit également décider comment fréquemment revisiter des pages qu'il a déjà vues, afin d'avertir son client courant des changements sur le Web.

2. Définitions

- Le **Web** est un graphe orienté $G = (P, L)$ où $p \in P$ est une page Web, $(p, q) \in L$ est un lien entre page p et page q .
- L'**arbre** visité $T = (V, L_c)$ où V est la collection des pages visité, et $L_c = \{(p, q) / q \text{ est visité à partir } p\}$

- Le **crawl** C est un programme qui commence avec une page initiale s s'appelant le **germe de départ**, explore le Web G afin de construire l'arbre visité T .
- La **frontière** B est l'ensemble de pages dans T qui ont les liens vers les pages hors de T
- La **profondeur** d_0 est la distance maximale à partir germe de départ s , $d(p)$ est la distance de la page p au germe de départ s .
- $E(p)$ est l'ensemble de pages où la page p se dirige.
- Le **score** de la page $S(p)$ est l'importance de page p sur un sujet particulier.

3. Architecture générale

Un crawl devrait avoir une architecture fortement optimisée. Shkapenyuk et Suel (Shkapenyuk et Suel, 2002) ont noté que : « Tandis qu'il est assez facile de construire un crawl lente qui télécharge quelques pages par seconde pendant une période courte, établir un système à haute performance qui peut télécharger des centaines de millions de pages au-dessus de plusieurs semaines présente un certain nombre de difficultés dans la conception de système, l'entrée-sortie et l'efficacité de réseau, et la robustesse et l'administration. »

Les crawls du Web sont une pièce centrale de moteurs de recherche, et des détails sur leurs algorithmes et architecture sont normalement caché comme les secrets d'affaires. Quand des conceptions de crawl sont publiées, il manque souvent les détails importants afin d'empêcher de reproduire le travail. Il y a également des soucis concernant le « Search Engine Spamming »¹, qui empêchent les moteurs de recherche de publier leurs algorithmes et leurs architectures.

¹ **Search Engine Spamming** ou **Spammdexing** est un ensemble de techniques consistant à tromper les moteurs de recherche sur la qualité d'une page ou d'un site afin d'obtenir, pour un mot-clef donné, un bon classement dans les résultats des moteurs (de préférence dans les tous premiers résultats, car les utilisateurs vont rarement au-delà de la première page qui, pour les principaux moteurs, ne comprend par défaut que dix adresses). – Source: <http://fr.wikipedia.org/wiki/Spamdexing>

3.1. Architecture de 2-modules

L'architecture d'un crawl la plus compact contient deux composants principaux: **downloader** et **scheduler**. Le **scheduler** calcule le score des pages pour déterminer l'ordre de traitement des pages. Le **downloader** télécharge les pages, les analyse, extrait les nouveaux liens et maintient la structure des liens.

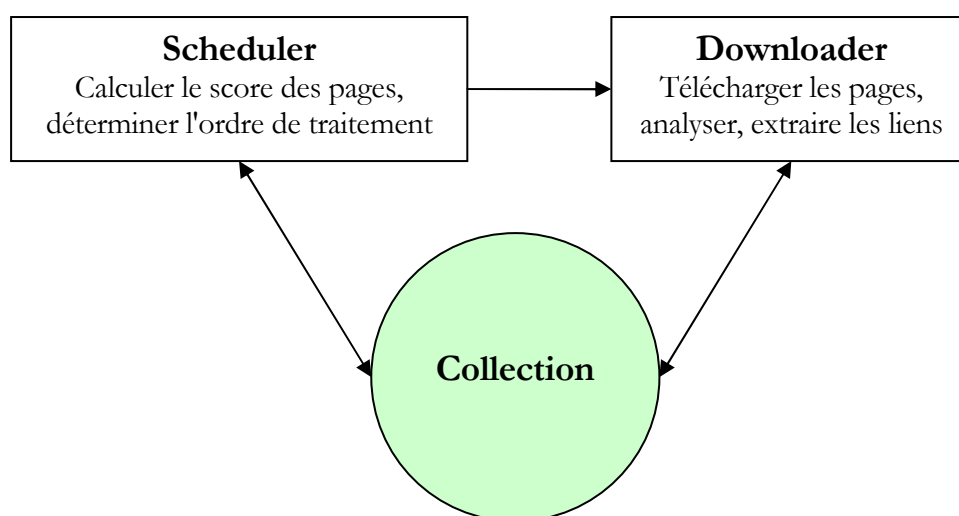


Figure 2: Architecture de 2-modules

3.2. Architecture de 4-modules

Il y a quelques problèmes avec l'architecture de 2-module. Premièrement, lorsque le **scheduler** travaille sur le graphe de Web, le graphe de Web ne peut pas être changé. Alors, le temps de la modification du graphe de Web doit être le plus court possible. Mais, on constate que la tâche d'analyse de page peut être longue. Pendant la durée d'analyse, le graphe de Web est occupé. La solution pour ce problème est d'analyser toutes les pages téléchargées en même temps, collecter les liens et enfin les ajouter dans la collection [3].

Un autre problème se trouve dans l'organisation du module **downloader**. La tâche d'analyse d'une page peut être très chère tandis que la tâche de téléchargement nécessite seulement une bonne connexion de réseau et les disques dur rapides.

D'ailleurs, les téléchargements des pages sont souvent portés par des processus parallèle, alors, chaque tâche de téléchargement est très légère. Pour résoudre ces questions nous divisons le module downloader en 2 modules: l'une se charge de téléchargement, l'autre est pour l'analyse des pages.

L'architecture de 4-modules est proposée par Carlos Castillo pour améliorer la performance des crawls du Web. Elle se compose de 4 modules suivants:

- **Manager**: calcule le score des pages et génère la liste de K URLs pour le téléchargement dans un cycle de traitement.
- **Harvester**: télécharge les pages
- **Gatherer**: analyse les pages et extrait les liens
- **Seeder**: maintien la structure de liens

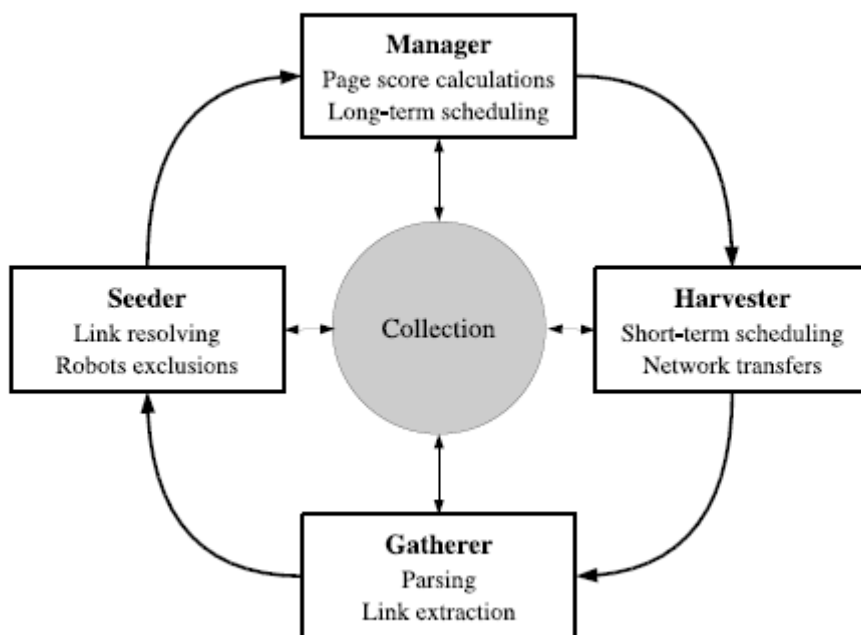


Figure 3: Architecture de 4-modules

3.3. Algorithme de crawl

Comme nous abordons au dessus, les algorithmes et les stratégies de crawl sont toujours gardées comme le secret de chaque moteur de recherche. Pourtant, les étapes principales dans l'algorithme sont similaires. Cet algorithme utilise les définitions dans la section au dessus.

```

function crawl(s, d0)
{
    T ← ({s}, ∅)
    B ← {s}
    while (B ≠ ∅)
    {
        Choisir dans B la pages p∈B: S(p) > S(q) ∀q∈B
        if (d(p) < d0)
        {
            for (q∈E(p))
            {
                if (q∉ V)
                {
                    B ← B ∪ {q}
                    V ← V ∪ {q}
                    Lc ← Lc ∪ {p, q}
                }
            }
        }
        B ← B \ {p}
    }
}

```

4. Stratégies de crawl

Lors de la construction un crawl, il est nécessaire de tenir compte la performance car la largeur de bande n'est ni infinie ni gratuit. Un crawl doit soigneusement choisir à chaque étape quelle page pour visiter à la fois prochaine.

Etant donné la taille courante du Web, même un grand moteur de recherche peut couvrir seulement une partie de l'Internet disponible. Car un crawl télécharge toujours juste une fraction des pages Web, c'est fortement souhaitable que la fraction téléchargée contient les pages les plus appropriées, et pas simplement un échantillon aléatoire du Web.

Ceci exige un métrique pour donner la priorité à des pages Web. L'importance d'une page est une fonction de sa qualité, de sa popularité en termes de liens ou visites, et même de son URL (le dernier est le cas de moteurs de recherche limités à un domaine spécifique, ou des moteurs de recherche limités à un site Web fixe). Concevoir une bonne stratégie de choix a une difficulté supplémentaire : il doit fonctionner avec l'information partielle, car l'ensemble complet de pages Web n'est pas connu pendant le parcours de Web.

Comme dans la section de définition, l'importance d'une pages p est $S(p)$, nous pouvons calculer l'importance de la page p par une des manière suivant:

- **Breadth-first (SB1):** Cette métriques est le plus simple mais dans quelques cas elle peut donner un bon résultat. Nous utilisons $SB1(p)$ au lieu de $S(p)$ pour le distinguer avec les autres métriques. La valeur de $SB1(p)$ est égale la profondeur de la page p par rapport le germe de départ, alors $SB1(p)=d(p)$.
- **BackLink-count (SB2):** On constate que la page p qui est pointé par plusieurs pages est plus importante que celles qui ont moins des pages de référence. La valeur de $SB2(p)$ est le nombre de page dans le Web entier qui ont le lien pointé à p . En effet, on ne peut pas exactement calculer $SB2(p)$ car cela demande d'un parcours di Web entier. Alors, un crawl peut souvent estimer la cette valeur $SB2'(p)$ par le nombre de liens qui ont déjà été visité par le crawl.
- **PageRank (SR):** tous les liens sont considérés également dans la métrique $SB2(p)$. Donc, il n'y a pas de différence entre le lien de la page d'accueil Yahoo et un lien d'une page individuelle. Pourtant, le lien à partir du site de Yahoo est toujours plus important, il doit avoir une valeur $SB2$ plus grande. La métrique PageRank, $SR(p)$, définit récursivement l'importance d'une page étant égale le somme de toutes l'importances des pages qui ont le "backlink"

ver la page p . Nous utilisons $SR'(p)$ pour estimer $SR(p)$ car nous avons seulement un sous-ensemble des pages du Web.

- **FowardLink-Count (SF)**: Cette métrique $SF(p)$ est similaire la métrique BackLink-Count $SB(p)$ mais elle se base sur le nombre des liens dans une pages. La valeur $SF(p)$ est directement calculé à partir de la page p , alors $SF'(p) = SF(p)$.
- **OPIC² (SO)**: Dans la métrique OPIC, toutes les pages commencent par la même valeur "cash". Chaque fois qu'une pages est récupérée, sa valeur "cash" est divisée parmi les pages qu'elle pointe vers. La valeur "cash" d'une page, ou la valeur SO , est le somme des pages qui ont le backlink pointé à elle. Cette métrique est similaire à la métrique PageRank mais le calcul est beaucoup plus rapide.
- **Lager-site-first (SL)**: Le but de cette métrique est d'éviter d'avoir trop de pages en suspens dans n'importe quel site Web. Un site qui a le nombre de pages en suspens plus grand est le site qui a plus de priorité. Alors, les pages dans ce site ont l'ordre plus haut. La valeur $SL(p)$ est égale au nombre des pages en suspens du site que p appartient à.

Plusieurs de travaux sont consacrés sur les métriques de l'importance dans les crawls du Web. Les auteurs essayent de tester et de comparer les méthodes traditionnels comme: Breadth-First, BackLink-Count et PageRank. La première étude sur les métriques de l'importance est de Cho *et al.* En utilisant 180.000 pages de teste dans le domaine de *stanford.edu*, ils trouvent que la métrique de calcul partiel de PageRank est la meilleure, suivi de la métrique Breadth-First et BackLink-Count. Najork et Wiener (Najork et Wiener, 2001) est testé son crawl avec 328 millions de pages, en utilisant la métrique Breadth-First. Ils ont constaté qu'un crawl de Breadth-First capturait les plus tôt des pages avec le rang de la page haut. L'explication par les auteurs pour ce résultat est que « les pages les plus importantes ont beaucoup de liens à eux, et ces liens seront trouvés tôt ». L'étude de Baeza-Yates *et al.* (Baeza-Yates *et al.*, 2005) montre que les métrique OPIC et Larger-Site-First sont bons mais le calcul de PageRank est lent et ne donne pas des bons résultats.

² OPIC: On-line Page Importance Computation

5. Respect de la politesse

Evidemment, les crawls recherchent des données beaucoup plus vite et plus détaillé que l'humain. Cela peut influencer à la performance des sites Web si un crawl effectue plusieurs requêtes par second et/ou télécharge un gros fichier sur un même site.

Comme la remarque de Koster (Koster, 1995), l'utilisation des crawls du Web est utile pour l'un certain nombre de tâches, mais vient avec un prix de la communauté générale. Les coûts de l'utilisation des crawls du Web incluent :

- Les ressources de réseau: comme les crawls exigent la largeur de bande considérable et fonctionnent avec un degré élevé de parallélisme pendant une longue période.
- Le surcharge de serveur: particulièrement si la fréquence des accès à un serveur indiqué est trop haute.
- Les crawls mal écrits: qui peuvent se briser des serveurs ou des routeurs, ou qui téléchargent des pages qu'elles ne peuvent pas manipuler.
- Les crawls personnelles: si ils sont déployés par trop d'utilisateurs, ils peuvent perturber des réseaux et des serveurs de Web.

Une solution partielle à ces problèmes est le protocole d'exclusion de robots, également connu sous le nom de protocole de robots.txt (Koster, 1996) qui est une norme pour que les administrateurs indiquent quelles pièces de leurs serveurs de Web ne devraient pas être accédées par des crawls. Cette norme n'inclut pas une suggestion pour l'intervalle des visites au même serveur, quoique cet intervalle soit la manière la plus efficace d'éviter la surcharge de serveur. Les moteurs de recherche commerciaux récemment comme Ask Jeeves, MSN et Yahoo peuvent employer des frais supplémentaires « Crawl-Delay», une paramètre dans le dossier de ***robots.txt*** pour indiquer le nombre de secondes pour retarder entre les demandes.

La première proposition pour l'intervalle entre les connexions a été donnée dedans et était de 60 secondes. Cependant, si des pages étaient téléchargées à ce taux d'un site Web avec plus de 100.000 pages au-dessus d'une connexion parfait avec la latence nulle et la largeur de bande infinie, cela prendrait plus de 2 mois pour télécharger

seulement ce site Web entier ; aussi, seulement une fraction des ressources de ce Web serveur serait utilisé. Ceci ne semble pas acceptable.

Cho (Cho et Garcia-Molina, 2003) utilise 10 secondes comme intervalle pour des accès, et le crawl WIRE (Baeza-Yates et Castillo, 2002) emploie 15 secondes comme défaut. Le crawl MercatorWeb (Heydon et Najork, 1999) suit une stratégie adaptative de politesse : si cela prenait des secondes de t pour télécharger un document d'un serveur donné, le crawl attend $10t$ des secondes avant de télécharger la prochaine page. Dill *et al.* (Dill *et al.*, 2002) utilisent 1 seconde.

Chapitre 3

CONSTRUCTION DU CRAWL FOCALISE

Ce chapitre présentera les problèmes du crawl focalisé et aussi notre solution pour la construction de crawl focalisé. On commence par des définitions utilisées dans la conception de notre crawl. L'architecture et tous les problèmes du crawl focalisé sont abordés dans les sections suivantes.

1. *Suppositions et notations*

1.1. Page Web

- La page Web p est un document informatique qui est identifié par une adresse. Ici, on supposera que toute page a une seule adresse, on ne tient pas compte des alias.
- Si p est une page alors $Lien(p)$ est l'ensemble de tous les hyperliens contenus dans la pages p .
- Si P est un ensemble des page Web alors $ad(P)$ est l'ensemble des adresses de ces pages.
- Si A est un ensemble d'adresses de pages Web, alors, $Page(A)$ est l'ensemble des pages correspondant aux adresses de A . On suppose que toute adresse est valide, c'est-à-dire que pour toute adresse a , il existe une page p d'adresse a .

On a donc que :

$$ad(Page(A)) = A$$

et

$$Page(ad(P)) = P$$

1.2. Germe de départ

- Soit R un ensemble de moteur de recherche définis par leurs adresses.

ex : Si on a deux moteurs de recherche Google et Yahoo, on a :

$R = \{\text{http://www.google.fr}; \text{http://fr.search.yahoo.com}\}$

- Soit D un domaine du Web

ex : tout *oubien* francophone *oubien* .fr *oubien* .ca *oubien* .vn *oubien* .inria.fr ...

- Soit C un ensemble de chaîne lexical

ex : $C = \{\text{"graphe"}, \text{"graphes"}\}$

- Soit $G_{R,D,C}$ l'ensemble des adresses récupérées sur R où $G_{R,D,C}$ est un ensemble d'adresses de pages du domaine D contenant au moins l'une des chaînes de C . $G_{R,D,C}$ doit contenir le plus grande possible d'adresses à ajuster selon les possibilité des moteurs de recherche.

ex : $G_{\{\text{Google}\}, \text{francophone}, \{\text{"graphe"}\}} =$

<http://grapheeasy.alrj.org/>,

<http://www.graphe.org/>,

<http://chilton.com/paq/archive/PAQ-04-065.html>,

<http://www.grapheeasy.com/index2.php?lang=fr>,

<http://dilan.irit.fr/...>}

- Soit F une formule propositionnelle de mots

ex : $F = (\text{NON "graffiti"}) \text{ ET } (\text{"graphe"} \text{ OU } \text{"graphes"})$

- Soit $G_{F,R,D,C}$ le sous ensemble maximal des adresses de $G_{R,D,C}$ tel que $\forall x \in G_{F,R,D,C} \text{ et } F(x) = \text{TRUE}$.

ex : $G_{((\text{NON "graffiti"}) \text{ ET } (\text{"graphe"} \text{ OU } \text{"graphes"})), \{\text{Google}\}, \text{francophone}, \{\text{"graphe"}\}} =$

{<http://grapheeasy.alrj.org/>,
<http://www.graphe.org/>,
<http://www.grapheeasy.com/index2.php?lang=fr>,
[http://dilan.irit.fr/ ...](http://dilan.irit.fr/)}

L'adresse <http://chilton.com/paq/archive/PAQ-04-065.html> a été supprimée car $Page(\{http://chilton.com/paq/archive/PAQ-04-065.html\})$ contient le mot "graffiti"

$G_{F,R,D,C}$ est appelé le *germe de départ*, c'est cet ensemble d'adresses qui va servir de germe pour aspirer le plus possible de pages p du domaine D du Web telles que $F(p) = \text{TRUE}$.

- Soit E l'ensemble des pages enregistrées sur le disque tel que:
 1. E contient toutes les pages dont l'adresse appartient à $G_{F,R,D,C}$
 $(Page(G_{F,R,D,C}) \subseteq E)$
 2. E contient plus grand nombre possible de pages (aspirées à partir du germe de départ $G_{F,R,D,C}$ en temps réaliste)
 3. $\forall x \in G_{F,R,D,C}, F(x) = \text{TRUE}$
- Soit E_N l'ensemble des pages de E qui ont été nettoyées et mises en formes: suppression des images, des scripts... dans le code HTML de la page. E_N doit être prêt pour passage aux outils de traitement linguistiques.
- Soit E_{NL} est l'ensemble des pages de E qui ont été lemmatisées et étiquetées morpho-syntaxiquement.

1.3. Graphes

- Le **graphe des hyperliens** G_H est le graphe dont les arcs sont définis par les hyperliens entre les pages.

- Le **graphe de similarité sémantiques** G_s entre les pages dont les arcs sont définis par: il existe une arête $r \leftrightarrow s$ entre les pages r et s si et seulement si la distance sémantique entre r et s ne dépasse pas un seuil prédéterminé.

$$d(r,s) < S_0$$

2. Constitution du germe de départ

Le fonctionnement du crawl nécessite d'abord une liste de URLs initiale. Dans ce travail, nous utilisons les moteurs de recherche Yahoo et Google pour construire le germe de départ car seulement Yahoo et Google fournissent l'API pour accéder leur service de recherche sur Web.

L'ensemble des adresses des pages à partir desquelles le crawl va s'effectuer est constitué par l'interrogation des moteurs de Yahoo et Google, via leur APIs respectives, selon le respect des limites du nombre de requêtes imposées. En effet pour une requête donnée, nous ne pouvons accéder qu'aux 1000 premiers résultats sur l'ensemble des pages indexées par Google et 5000 résultats de pages indexées par Yahoo. Google donne la permission d'utilisation de son API par une clé. La clé est fournie pour chaque personne qui a un compte d'accès de Google. Yahoo demande une inscription du nom de crawl avant le téléchargement son API. Ce nom est utilisé comme la clé pour l'utilisation de l'API de Yahoo. Toutes les clés d'accès seront fournies via l'interface utilisateur ou le fichier de paramètres

Pour atteindre le plus grand nombre d'adresses, l'interrogation devra jouer sur les différentes manières d'exprimer une même requête. Si nous faisons varier l'ordre dans lequel ces résultats sont affichés, nous pouvons espérer atteindre un nombre de pages plus important. Par exemple, les requêtes suivantes provoquent un ordonnancement différent des résultats alors que les pages correspondant sont a priori les mêmes :

- graphe
- graphe graphe
- graphe OR graphe

A partir du premier germe ainsi constitué, on peut l'augmenter en adressant aux moteurs des requêtes du type :

- *liste_de_mots* site:hostname
- *liste_de_mots* related:url

La requête 1 correspond aux pages correspondant à la liste de mots spécifiée pour un site entier. La requête 2 comprend les pages contenant la liste de mots et pointant vers une url donnée. Par exemple, si le germe contient les urls <http://dilan.irit.fr/Projets/Projets.php> et <http://www.apprendre-en-ligne.net/graphes/>, on ajoute au germe initial les résultats des requêtes :

- *liste_de_mots* site:dilan.irit.fr
- *liste_de_mots* related:dilan.irit.fr/Projets/Projets.php
- *liste_de_mots* site:www.apprendre-en-ligne.net
- *liste_de_mots* related:www.apprendre-en-ligne.net/graphes/

3. Architecture

Le but de la conception de ce crawl est de récupérer le plus grand nombre de pages sur le Web qui concernent à un sujet déterminant par la chaîne lexicale **C**. Nous voulons télécharger tôt les pages les plus concernées au sujet. Toutes les analyses dans le chapitre 2 sont utilisées pour la conception de ce crawl.

3.1. Composantes

Pour la construction du crawl focalise, nous suivons l'architecture de 4-modules. Dans cette conception, nous ajoutons un module pour valider la pertinence du contenu de la page. Ce module est correspondant à la formule propositionnelle.

3.1.1. Launcher

Ce module va lire le fichier des paramètres pour établir les méta-données dans le mémoire. Ensuite, il vérifie si le germe est constitué ou non. Si le germe de départ n'est pas encore existé, le module Launcher va le créer par interroger les moteurs de recherche Google et Yahoo.

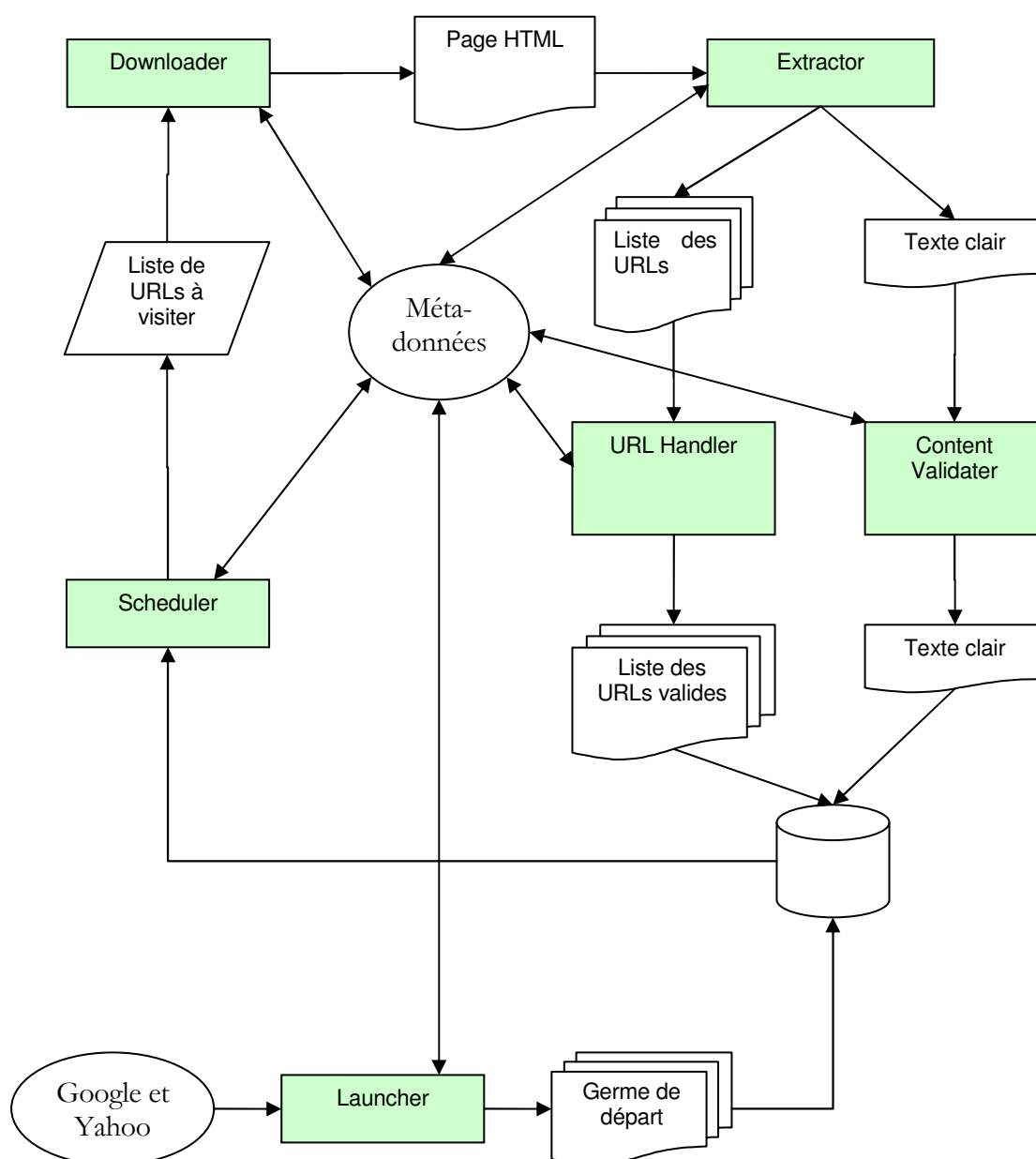


Figure 4: Architecture du crawl

3.1.2. Scheduler

Ce module utilise les informations des pages dans la base de données pour générer la liste des URLs à visiter pour chaque cycle de traitement. Cette liste contient K URLs ($K = 1000$ par défaut).

Ce crawl vise à récupérer le plus grand nombre de pages Web concernées à un sujet particulier. On peut le considérer comme un crawl "exhaustif" dans l'espace de ce

sujet. Mais on veut que le crawl télécharge tôt les pages plus importantes. Alors la métrique de l'importance des pages doit être simple mais rapide dans le calcul car l'ordre de téléchargement des pages n'est pas trop important.

D'ailleurs, à partir du graphe des hyperliens, on peut facilement saisir les informations des liens entrés et des liens sortis d'une page. Le graphe des hyperliens est construit pendant le parcours du Web. Alors, le calcul des métriques BackLink-Count (**SB2**) et ForwardLink-Count (**SF**) est gratuit. C'est la raison que nous choisissons ces deux métriques pour la stratégie de crawl.

Pourtant, la valeur de **SB2** et **SF** est calculée pour le graphe courant pas pour le graphe de Web entier. Donc, en fait, ce sont les deux valeurs **SB2'** et **SF'**.

3.1.3. Downloader

Le module downloader reçoit la liste de **KURLs** générée par le module Scheduler et essaie de les télécharger. Pour chaque cycle de traitement, toutes les pages dans cette liste sont téléchargées.

i. Queue des URLs

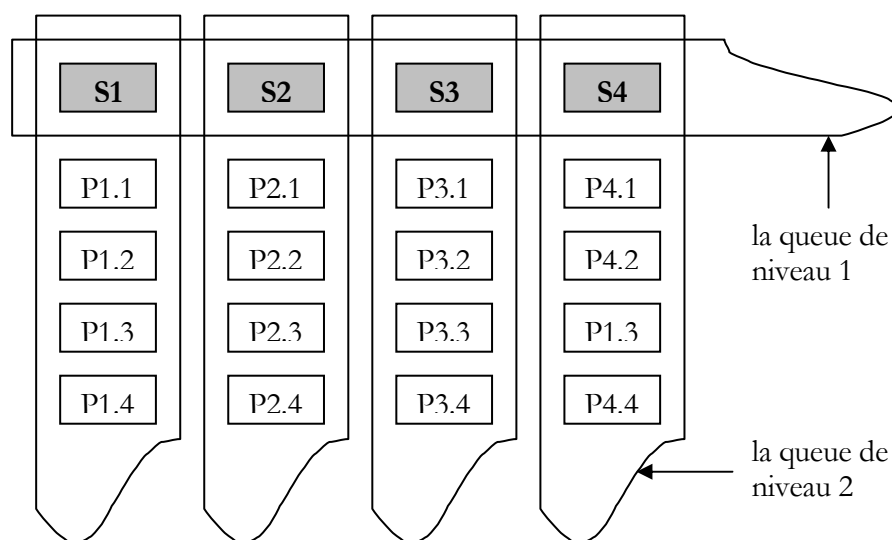


Figure 5: Queue de deux niveaux: S1, S2,... sont les sites Web et P_{x.y} est la y^{ème} page de site x.

La liste des **KURLs** générée par le module Scheduler est réorganisée dans une queue de deux niveaux. On groupe tous les URLs appartenis le même site dans une queue

qu'on l'appelle la queue de niveau 2 ou la queue des pages. Alors chaque site est correspondant à une queue.

La queue de niveau 1 est la queue des sites. Les éléments de cette queue sont les sites. On peut considérer cette organisation comme une queue de queue. Ce type de queue semble efficace pour le parallélisme et le respect de la politesse du module Downloader. Le crawl WIRE utilise aussi une queue comme ceci pour la gestion des URLs dans son module Harvester.

ii. Parallélisme

Afin de profiter de la connexion du réseau, ce module est organisé en plusieurs processus légers (multi-thread). Chaque processus se charge de télécharger les pages à partir les URLs données. Les étapes pour télécharger une page sont:

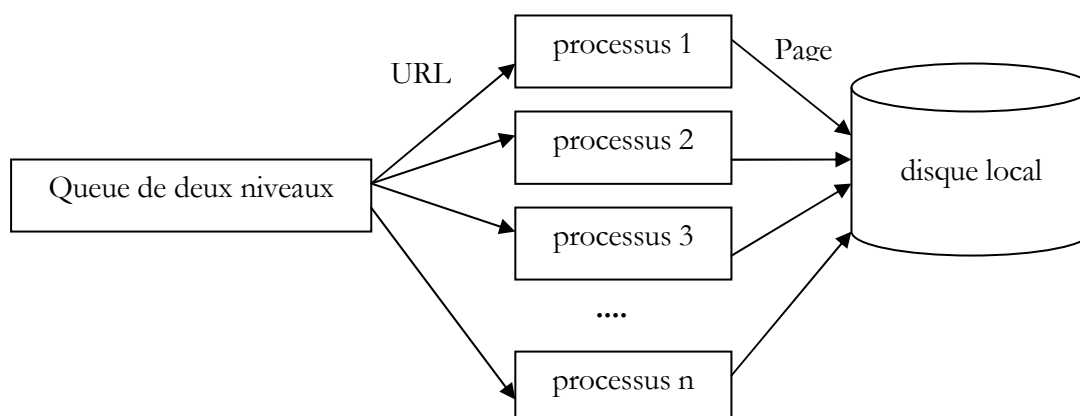


Figure 6: Parallélisme

1. *Récupérer l'en-tête HTTP de la page*: L'en-tête HTTP contient les métas données de la page: le type, l'encodage... On utilise la méthode HEAD dans le protocole HTTP pour récupérer cet en-tête sans télécharger le contenu entier de la page.
2. *Vérifier le type de la page*: Dans l'implémentation courant, on n'acceptera que les pages du type HTML ou texte (text/html, text/plain).
3. *Télécharger et enregistrer la page*: Si la page n'est pas en HTML ou texte, on ne la télécharge pas et on marque dans la base de page le type de la page. Si la page est en HTML et texte, on la télécharge et enregistre dans le disque local. Ce fichier est plus tard traité par le module Extractor.

iii. Respect de la politesse

La politesse d'accès des sites est très importante pour maintenir un vrai crawl. Dans ce crawl, la politesse choisie est de ne pas établir plus d'une connexion à un site dans un même temps, d'attendre pour un nombre de seconds (10 seconds par défaut) parmi les téléchargements.

La politesse est maintenue par la queue de deux niveaux. Chaque fois, les processus demandent un nouveau URLs pour le téléchargement, la queue de deux niveaux va choisir un site qui n'est pas dans l'état "occupé", un site est "occupé" si le temps d'attente n'est pas suffisant. Parce que les pages dans un site sont organisées dans une queue de niveaux 2, alors après avoir choisi un site, la page dans la queue de ce site est retirée. L'algorithme pour retirer un URL à partir de la queue de deux niveaux q est au dessous. Dans cet algorithme, $q2$ est la queue de niveaux 2 qui est correspondant à un site dans la queue q , $w0$ est le temps d'attente pour chaque téléchargement. La sortie de la fonction **getURL()** est un URL ou nul s'il n'y a aucun URL dans la queue.

```

function getURL( q )
{
    if (q.isEmpty()) return null
    q2 = dequeue(q)
    if (q2.isEmpty())
        return getURL(q)
    if (q2.getWaitTime()<w0)
    {
        enqueue(q, q2)
        return getURL(q)
    }
    p = dequeue(q2)
    q2.lastAccessTime=now
    enqueue(q, q2)
    return p
}

```

iv. Redirections et erreurs liées au réseau

Lorsqu'une page est redirigée (header HTTP 3xx ou méta-balise HTML équivalente), on note le statut de cette redirection dans la base de données, l'url de redirection est insérée et le lien **page redirigée** → **adresse de redirection** est inséré.

Lorsqu'une page isolée provoque une erreur réseau, on peut la revisiter ultérieurement un nombre donné de fois (nombre à paramétrer).

Lorsque c'est un domaine entier qui provoque des erreurs réseau, on peut stopper temporairement son parcours et y revenir ultérieurement, puis décider de l'abandonner complètement.

Lorsqu'enfin les pages de plusieurs domaines différents provoquent des erreurs réseau, on peut supposer qu'il s'agit d'une panne générale. Dans ce cas, le programme s'arrête temporairement (eg. 10 minutes, à paramétrer) et reprend automatiquement. On peut s'assurer que le réseau ne dysfonctionne en se connectant à des adresses de référence (supposées toujours valides) et en s'assurant qu'elles ne fonctionnent pas.

3.1.4. Extractor

Le module Extractor reçoit les données binaires de la page Web et l'analyse. L'analyseur HTML utilisé dans ce module est un analyseur DOM, qui va construire l'arbre DOM à partir de la page HTML.

Pendant l'analyse, les URLs sont détectés et ajoutés dans une liste. L'analyseur va aussi extraire le texte clair de la page. Il nettoie toutes les balises HTML, les informations concernant les couleurs, les images, les fonts,... sont supprimées. Il reste seulement le contenu textuel de la page.

i. Suivi de liens

Lors de l'analyse d'une page, tous les liens sont extraits. Il s'agit de liens HTML classiques (`...`) et également des liens contenus dans :

- les frames (voire Figure 7);
- les iframes ;
- les images mappées (voire Figure 8);

- le code javascript.

Les URLs dans les liens HTML classique et dans les frame, iframe, les images mappées est facile à parser. Mais le suivi de liens dans le code javascript est compliqué à trouver exactement. Normalement, les URLs se situe dans deux commandes de navigation de javascript:

```
windows.open('agenda.html')
document.location.href='agenda.html'
```

Cependant, javascript est un morceau de programme, et alors, il peut contenir des variables et des appels de fonction, comme dans les cas suivants:

Cas 1: l'appel de la fonction

```
function ouvrirFenetre (adresse)
{
    windows.open (adresse);
}
ouvrirFenere ('agenda.html');
```

Cas 2: les variables

```
var base = 'www.irit.fr';
url = base + '/faq.html';
document.location.href=url;
```

Pour faire émerger tous les liens dans le code javascript, on a besoin une analyseur complet de javascript. Nous simplifions ces cas en supposant que le code javascript contient seulement les navigations directes, cela veut dire, il n'y a pas de variables ou des appel de fonction. Nous ne traitons pas les autres cas.

Pour chaque lien extrait, l'URL est résolue dans sa forme canonique. L'URL résultant sera éventuellement filtrée en tenant compte des différents critères (domaine, extension de fichier). Si l'URL ne répond pas à ces critères, elle n'est pas insérée dans la base de données. Dans le cas contraire, on note l'existence du lien **url source** → **url destination** dans la base de données.

Pour chaque sommet inséré (i.e. ne faisant pas partie du germe initial), on note dans la base l'adresse de la page pointant vers le sommet inséré. Ce renseignement peut être utile à la détection de pièges à robots.

- `<frameset cols="210,*">`
`<frame src="menu.html" />`
`<frame src="framesetAccueil.html" />`
`</frameset>`



Figure 7: Liens dans le frameset

- `<map>`
`<area polygon="..." src="no.html" />`
`<area polygon="..." src="ne.html" />`
`<area polygon="..." src="se.html" />`
`<area polygon="..." src="so.html" />`
`</map>`

Figure 8: Liens dans les images mappées

ii. Du HTML au texte

Avant de commencer l'analyse proprement dite d'une page, il est possible de récolter des méta-données (lorsqu'elles sont présentes) dans les en-têtes HTTP et les méta-balises HTML de la section **head**. Ces méta-données peuvent éventuellement renseigner sur le type de page (texte, html ou autre), l'encodage (jeu de caractères) de la page et la langue du document (français ou autre).

Si une page apparaît comme n'étant pas en français, contenant un média non textuel ou étant encodée dans un jeu de caractère qui révèle un autre alphabet que le français, elle ne sera pas analysée et la raison de ce rejet sera mentionnée dans la base de données.

Suppression des balises

En extrayant le texte du HTML, il est nécessaire de porter attention au fait que certaines balises segmentent les mots et d'autres non :

- tues ! {*tues*}
- tu<div>es ! {*tu, es*}

Une liste de balises qui jouent le rôle de segmenteurs est fournie.

Recodage vers le jeu de caractères de travail

Le texte est, si nécessaire, recodé vers le jeu de caractères de travail (latin9, par exemple) à partir de l'encodage initial de la page (latin1, windows-1252, Unicode).

Résolution des entités HTML

Les entités HTML sont ensuite résolues. Une même entité peut être exprimée de différentes manières. Ainsi le caractère **é** peut être écrit :

- é ;
- é ;
- é ;

Les mots contenant des entités non résolues sont supprimés. La liste des différentes entités HTML est fournie.

Segmentation en mots et construction de la matrice de fréquence des mots

Du texte, nous pouvons obtenir une liste de mots en projetant sur le texte l'expression régulière, non sensible à la casse :

$$\backslash\text{b}[a-zA\grave{\text{a}}\hat{\text{a}}\acute{\text{e}}\acute{\text{c}}\acute{\text{e}}\grave{\text{i}}\grave{\text{i}}\ddot{\text{o}}\ddot{\text{o}}\grave{\text{u}}\grave{\text{u}}\grave{\text{u}}\grave{\text{c}}\grave{\text{a}}\grave{\text{e}}'\-]+\backslash\text{b}$$

Pendant la segmentation en mots, l'analyseur compte aussi la fréquence de chaque mot dans une page. Ces valeurs de fréquence des mots dans une page sont enregistrées dans un fichier texte correspondant. L'ensemble des fichiers de fréquences sont actuellement la matrice de fréquence de mots.

3.1.5. URL Handler

Ce module se charge de la résoudre les URLs et d'examiner les conditions nécessaires avant de les insérer dans la base de données. Les URLs détectés par le module Extractor sont résolues dans la forme canonique. Puis, ils sont examinés par le filtre de domaine, la règle d'exclusion, et le piège de robot.

i. Filtre de domaine

Ce filtre permet le crawl d'éviter de visiter les page ou les site dans un domaine particulier comme: .com, .uk, .us ou .ebay.com... L'utilisateur peut définir le filtre par l'expression régulière ou le caractère wildcard (*).

ii. Règle d'exclusion

Les URLs doivent respecter les règles d'exclusion définies dans les fichiers **robots.txt**³ des sites ainsi que les méta-balises HTML **noindex**, **nofollow** et **noarchive**.

iii. Piège à robot

Un piège à robot (Spider Trap⁴) est un ensemble de pages Web qui peut intentionnellement ou involontairement être employé pour faire briser un crawl mal construit. Des pièges à robot peuvent être créés « pour attraper » les spambots⁵ ou d'autres crawl qui gaspillent la largeur de bande d'un site Web. Ils peuvent également être créés involontairement par des pages dynamiques avec les liens qui se dirigent continuellement au jour suivant ou à l'année.

Un piège à robot fait entrer un crawl du Web quelque chose comme une boucle infinie, qui gaspille les ressources du crawl, abaisse sa productivité, et, dans le cas d'un crawl mal écrit, peut se briser le programme.

³ voir <http://www.robotstxt.org/wc/robots.html>

⁴ http://en.wikipedia.org/wiki/Spider_trap

⁵ <http://en.wikipedia.org/wiki/Spambot>

- La page satisfait-elle la requête initiale ? Suivant la réponse (que l'on reporte dans la base de données), et en fonction de la profondeur du crawl, on décide d'insérer ou non dans la base les liens présents dans la page.
- La page paraît-elle être en français ? Le critère pour répondre est le suivant :
 - %mots connus: SeuilMC
 - %mots grammaticaux français: SeuilGrFR
 - %mots grammaticaux autres langues: SeuilGrA

Nous disposons de liste de mots connus, de mots grammaticaux du français et d'autres langues.

La contenu de la page doit être satisfait la formule propositionnelle des mots et le profondeur. La formule propositionnelle contient les mots, les opérations logiques **and** (&), **or** (|) et **not** (^) et les parenthèses.

Par exemple:

F1 = (graphe | graphes)

F2 = (graphe | graphes) & (^image)

La profondeur d'une page est calculée comme l'algorithme suivant: si le contenu de la page satisfait la formule propositionnelle, la profondeur est toujours égale à 0; dans le cas contraire, la profondeur est égale à la profondeur de la page parent plus 1.

```

function depth(p)
{
    if (F(p)=TRUE) return 0
    else return p.parent.depth+1
}

```

Par exemple, si on limite la profondeur de 3, on examine l'arbre de parcours suivant: Les pages P1, P2, P4, P7 ont la profondeur 0. La page P6 a la profondeur 3, on s'arrête de parcourir les URLs dans la page P6.

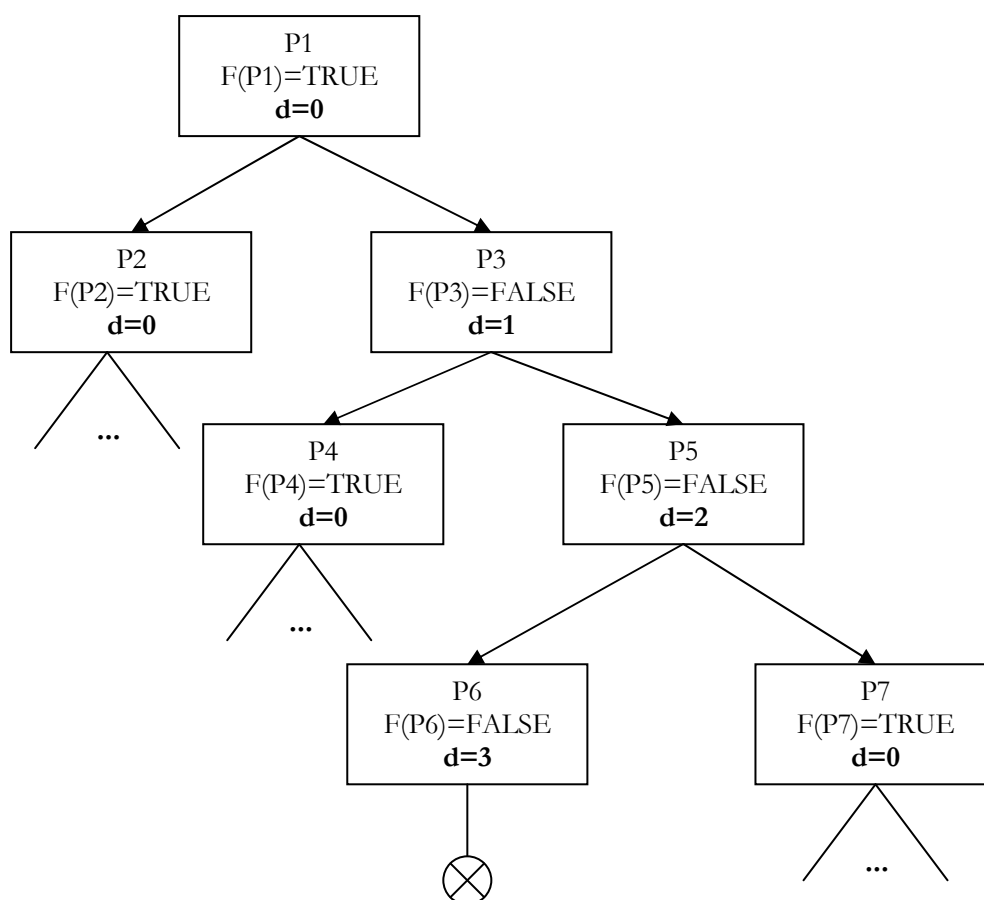


Figure 9: Exemple du calcul de la profondeur

3.2. Base de données

La base de données contient deux tables pour stocker les informations des pages (table **page_node**) et le graphe des hyperliens (table **link_edge**).

Table :page_node		
Ordre	Champ	Description
1	id	Identifiant de la page, les identifiants est en fait le hachage MD5 du URL de pages
2	url	Le URL de la page
3	hasword	= 0 : la page ne satisfait pas la fonction logique = 1 : la page satisfait la fonction logique
4	status	= 0 : la page n'est pas encore visitée = 1 : la page est visité et satisfait la fonction logique = 2 : la page est visité mais ne satisfait pas la fonction logique = 3 : s'il y a des erreur durant le traitement de la page

		= 4 : si la page est déjà existée dans la base de données mais avec un autre URL = 5 : s'il y a la réorientation de la page (HTTP 3xx) = 6 : la page est déjà analysé par Syntex
--	--	--

Table : link_edge		
Ordre	Champ	Description
1	id	Identifiant de l'arête, La valeur de l'identifiant est le hachage MD5 de la concaténation de deux URLs
2	page1	identifiant de la page de source
3	page2	identifiant de la page de cible

Le graphe des hyperliens est construit pendant le processus de crawl. Chaque page est correspondant à un sommet du graphe et est stocké dans la table **page_node**. Les arcs du graphe sont stockés dans la table **link_edge**. Le contenu de la page est stocké dans un répertoire sur le disque local.

3.3. Interface d'utilisateur

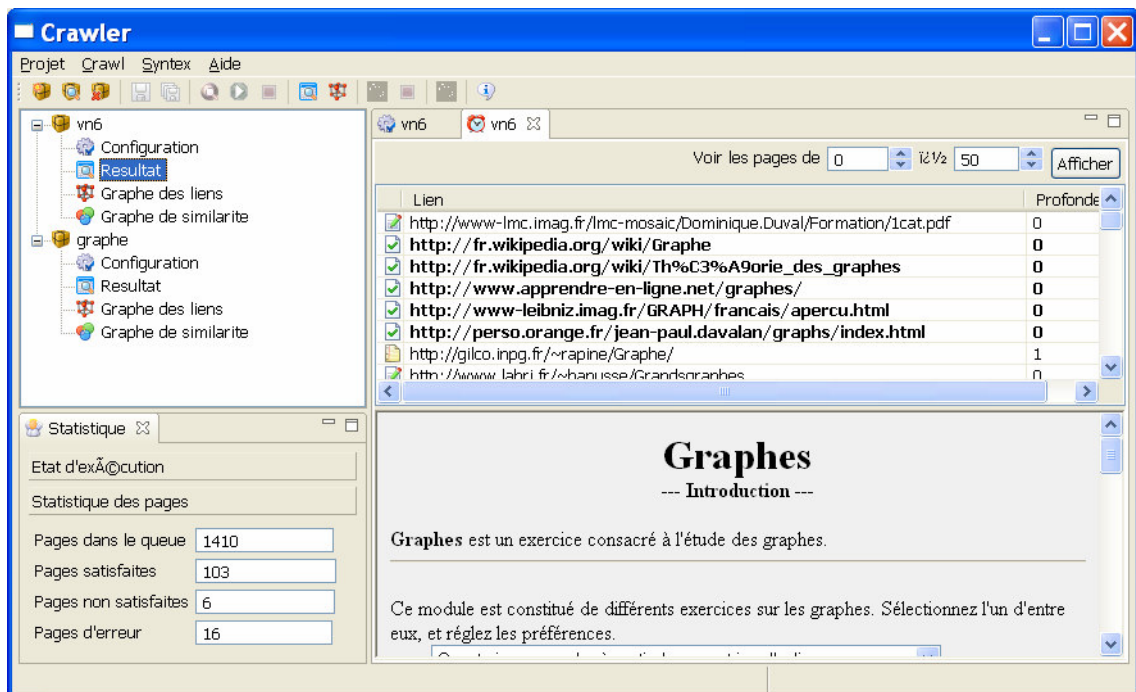


Figure 10: Interface d'utilisateur 1

Le crawl peut fonctionner en mode de texte mais, nous avons développé aussi une interface graphique pour mieux gérer les projets de crawl, les configurations, la visualisation des résultats et la suivi de traitement. Maintenant, le module de

visualisation des graphes (graphe de hyperliens et graphe de similarité) n'est pas encore intégré dans cette interface. Ce module est développé par le projet **prox** de l'IRIT (<http://prox.irit.fr>) pour visualiser le graphe en 3 dimensions.

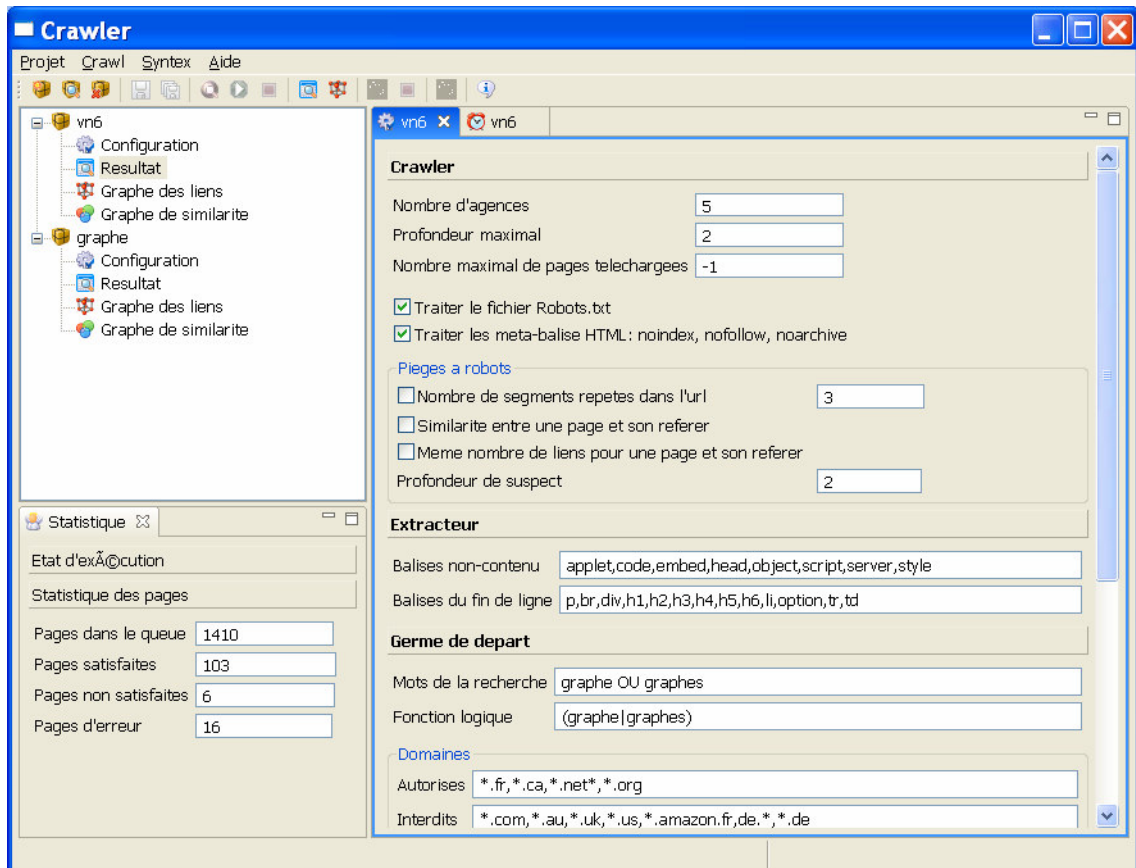


Figure 11: Interface d'utilisateur 2

4. Environnement de programmation et dépendances

Le langage de programmation utilisé est Java en utilisant l'environnement de programmation Eclipse. Pour l'implémentation, nous utilisons les bibliothèques suivantes:

HttpClient: Cette bibliothèque permet de effectuer les connexions HTTP à des sites Web et télécharger les pages. Elle donne aussi des outils pour la résolution des URLs.

JTidy: JTidy est une implémentation en Java de Tidy. Cette bibliothèque se charge d'analyser les pages HTML, résoudre le problème de l'encodage. JTidy peut aussi réparer les mauvaises balises HTML pendant l'analyse.

Eclipse RCP: Eclipse RCP est un framework pour le développement des applications en Java. Ce framework a l'architecture de "plugin", qui permet aux développeurs de facilement brancher ou supprimer les modules de l'application. Eclipse RCP donne l'ensemble de modules pour le développement de l'interface graphique en se basant sur les bibliothèques SWT et JFace.

Chapitre 3

ANALYSE LINGUISTIQUE DES PAGES WEB

L'objectif de la deuxième partie de stage est d'étudier les outils d'analyse linguistique existants et les intégrer dans le système pour analyser les pages Web récupérées par le crawl. Il existe actuellement des outils développés par l'ERSS pour l'analyse linguistique. Il s'agit de deux outils: Syntex et Upery pour l'analyse syntaxique et l'analyse distributionnel. Syntex utilise le résultat d'un étiqueteur morphosyntaxique qui s'appelle TreeTagger développé par l'Université de Stuttgart. Dans ce chapitre, nous introduisons ces trois outils et l'intégration de ces outils dans le système.

1. Outils d'analyse linguistique

1.1. TreeTagger

TreeTagger est un bon outil d'analyse morphosyntaxique développé par l'Université de Stuttgart. TreeTagger a été avec succès employé pour étiqueter les textes français, allemands, anglais, français, italiens, espagnols, bulgares, russes, grecs, portugais et est facilement adaptable à d'autres langues si un lexique et un corpus manuellement étiquetés pour l'apprentissage sont disponibles.

TreeTagger	NOM	TreeTagger
est	VER:pres	être
facile	ADJ	facile
à	PRP	à
utiliser	VER:infi	utiliser

Exemple de la sortie de TreeTagger

Le fichier de paramètre pour le français a été fourni avec bonté par Michel Génèreux. La liste détaillée des catégories morphosyntaxiques est citée dans le Tableau 1 dans l'annexe.

1.2. Syntex

Syntex (Bourigault & Fabre, 2000) est un analyseur syntaxique de corpus. Il existe actuellement une version pour le français et une version pour l'anglais. L'entrée de Syntex est un corpus étiqueté par TreeTagger. La sortie est un corpus analysé syntaxique et un réseau de syntagmes. Il y a donc deux étapes de traitement de Syntex: *l'analyse syntaxique en dépendance et la construction du réseau de syntagmes.*

1.3. Analyse syntaxique en dépendance

L'analyse syntaxique est de trouver les relations entre les mots dans une phrase. La Figure 12 illustre la chaîne de traitement pour l'analyse syntaxique pour la phrase "*Le chat de Marie mange une petite souris*".

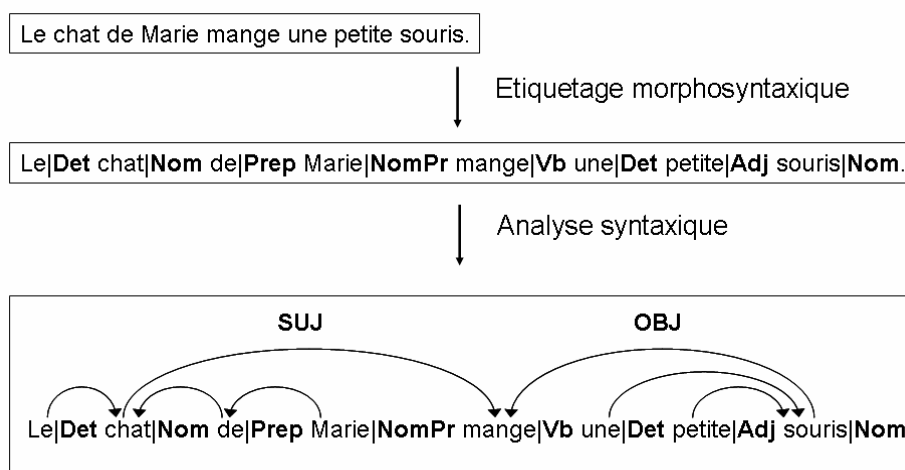


Figure 12: Exemple de l'analyse syntaxique

1.3.1. Relations syntaxiques

Dans chaque phrase, Syntex pose des relations de dépendance syntaxique entre les mots, les mots ont été préalablement étiquetés. Une relation de dépendance syntaxique se compose de deux parties (un *régi* et un *recteur*) et une orientation du régi vers son recteur. L'orientation est libellée par le nom de la relation.

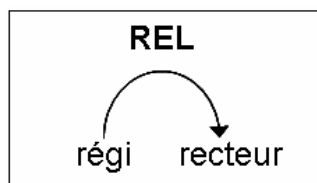


Figure 13: Relation de dépendance syntaxique

Les relations syntaxiques doivent respecter deux contraintes suivantes:

Contrainte 1: Un mot ne peut avoir plus d'un recteur, mais un mot (recteur) peut avoir plusieurs régés et un mot peut être recteur ou régi à la fois.

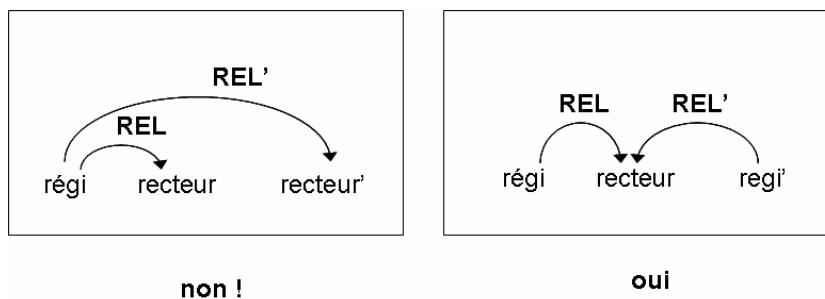


Figure 14: Contrainte 1

Contrainte 2: Les relations de dépendance ne peuvent pas se croiser.

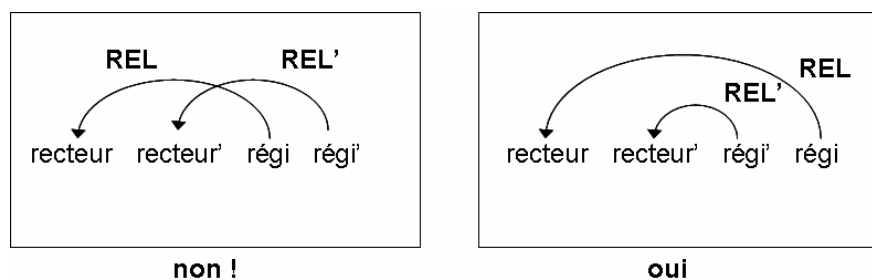


Figure 15: Contrainte 2

Dans cette section, nous parlerons de quelques relations principales qui sont traitées par Syntex.

Relation	Recteur	Régi	
SUJ	Verbe	Nom, Pronom	Marie → mange
OBJ	Verbe	Nom, Pronom, Verbe infinitif	Marie regarde → Jean
PREP	Verbe, Nom, Adjectif	Préposition	le chat → de Marie
ADJ	Nom	Adjectif	un chat → noir
ATT	Nom, Pronom	Adjectif, Nom	le chat → est noir
PREP-d	Préposition	Nom, Verbe infinitif	le chat → de Marie
DET	Nom	Déterminant	le → chat

Figure 16: Quelques relations principales

1.3.2. Module d'analyse syntaxique automatique

Pour chaque relation, Syntex implémente un module séparé pour traiter la relation. Chaque module prend en entrée les résultats des modules précédents. Chaque module est constitué d'un ensemble d'heuristiques de parcours de la séquence annotée. Dans cette section, nous présentons les algorithmes de traitement des relations syntaxiques et la solution pour l'ambiguïté.

i. Algorithme DET

- Point de départ : le déterminant (le régi)
- Direction : droite
- Arrêt : au premier Nom



Figure 17: Algorithme DET

ii. Algorithme PREP-d

- Point de départ : la préposition (le recteur)
- Direction : droite
- Avec saut d'un régi jusqu'à son recteur
- Arrêt : au premier Nom ou Pronom ou Verbe à l'infinitif

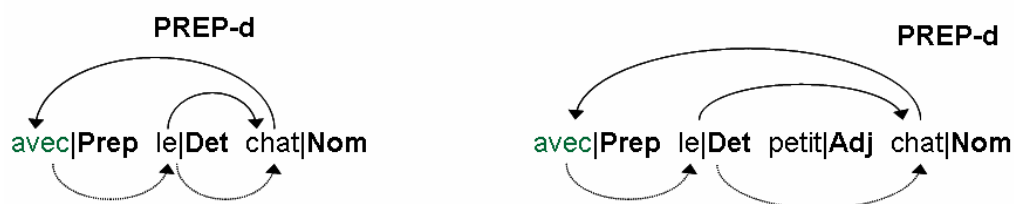


Figure 18: Algorithme PREP-d

iii. Algorithme OBJ : « premier nom à droite »

- Point de départ : le verbe (le recteur)
- Si pronom clitique objet juste à gauche ou pronom relatif que à gauche: choix, arrêt
- Sinon direction : droite
- Saut de certaines séquences entre virgules (incises)
- Arrêt : au premier Nom (ou verbe à l'infinitif)

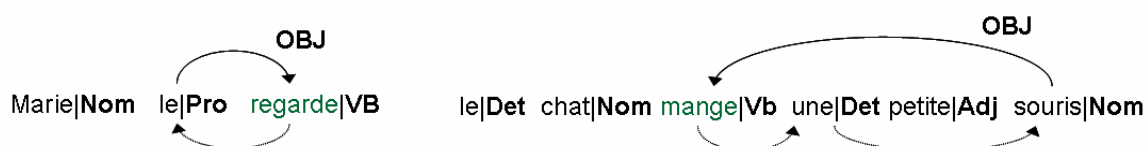


Figure 19: Algorithme OBJ

iv. Algorithme SUJ : « dernier nom à gauche »

- Point de départ : le verbe (le recteur)
- Direction : gauche
- Saut de certaines séquences entre virgules (incises)
- Arrêt : au dernier Nom ou Pronom

- Si échec (sujet inversé) :
 - Point de départ : le verbe
 - Direction : droite
 - Arrêt : au premier Nom ou Pronom

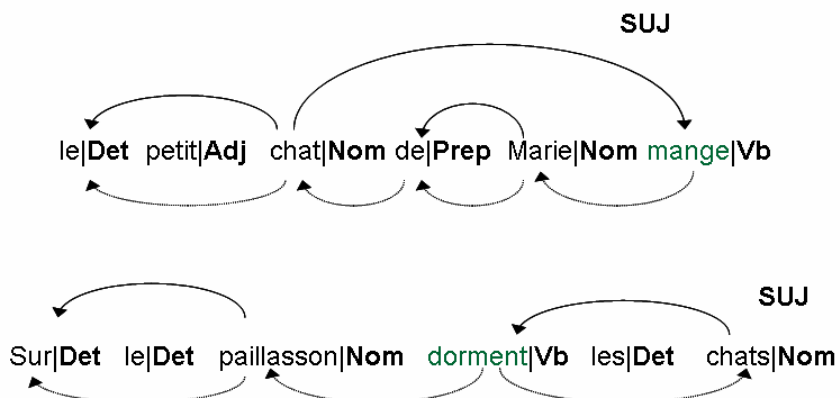


Figure 20: Algorithme SUJ

v. Algorithme ADJ

Dans cet algorithme, il existe les cas d'ambiguïté. Pour le cas où le nom est juste à droite de l'adjectif, ce nom est le recteur. Mais si il y a pas plusieurs noms à gauche coté de l'adjectif, l'ambiguïté se produit.

Par exemple: [Nom1 de Nom2 *Adjectif*]

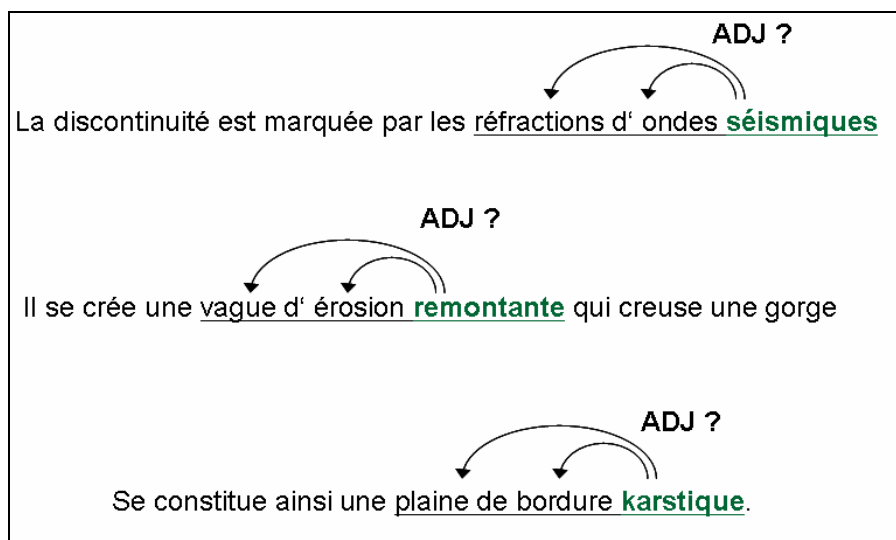


Figure 21: Ambiguïté de rattachement des adjectifs

Dans ce cas, on doit utiliser les autres informations pour choisir le vrai recteur. Syntex utilise le historique pendant l'analyse: les indices endogènes calculées sur le corpus. Ce sont actuellement l'acquisition de propriétés de rection dans les contextes non ambigus.

Pour chaque adjectif, on peut trouver quelques candidats de recteur. On va affecter les indices aux candidats. Chaque candidat est relié avec un argument *arg*, nombre de fois que l'adjectif est régi par le candidat dans un contexte non ambigu. On choisit le candidat qui a le score d'indice le plus élevé. S'il y a des concurrences ou aucun candidat n'a reçu d'indice, on choisit le candidat le plus proche de l'adjectif. Alors, l'algorithme ADJ est :

- Point de départ : l'adjectif (le régi)
- Si nom juste à droite : arrêt
- Sinon direction : gauche
 - Recherche des candidats
 - Affectation des indices aux candidats
 - Choix de celui
 - qui a le score d'indice le plus élevé
 - ou qui est le plus proche de l'adjectif
 - si concurrence
 - Ou si aucun candidat n'a reçu d'indice

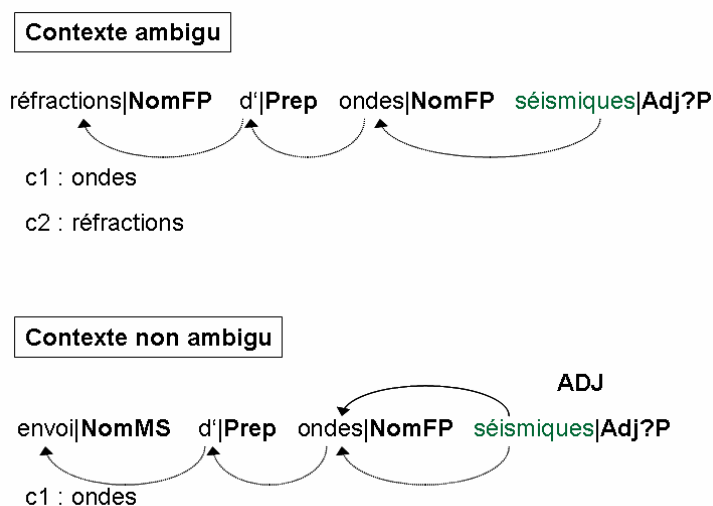


Figure 22: Algorithme ADJ: recherche des candidats

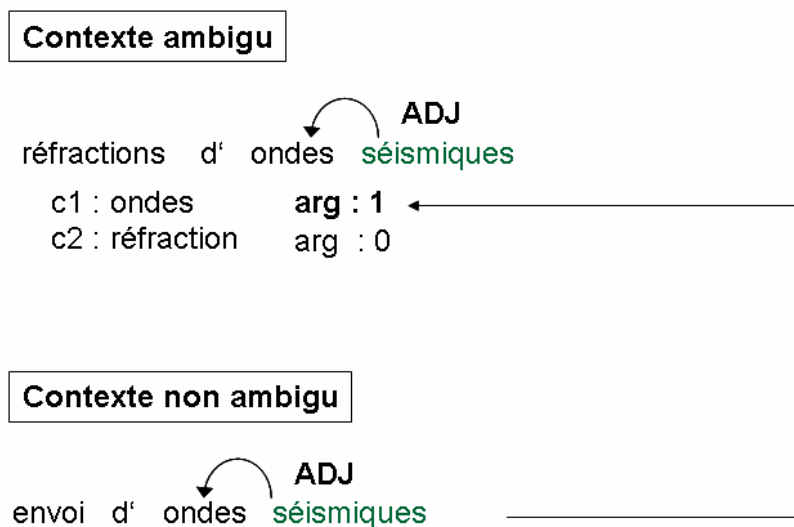


Figure 23: Algorithme ADJ: sélection d'un candidat

vi. Algorithme PREP

La relation PREP a aussi les ambiguïtés. Les ambiguïté dans cette relation sont plus souvent que dans la relation ADJ et plus difficile à résoudre.

Par exemple: [Verbe Nom Adjectif *en*]

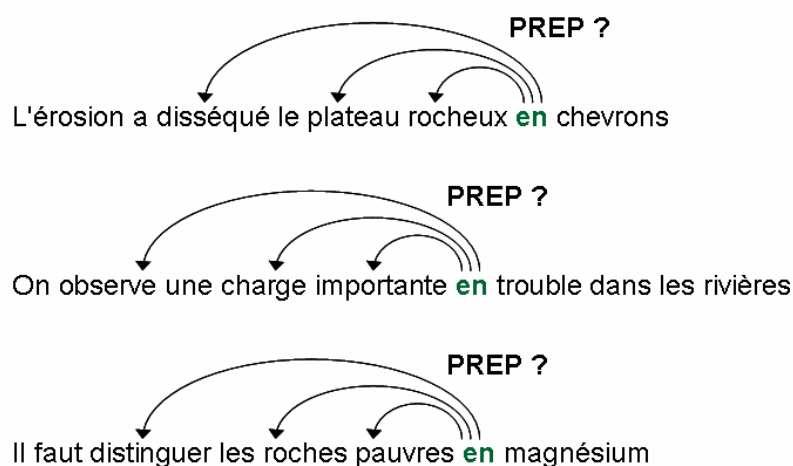


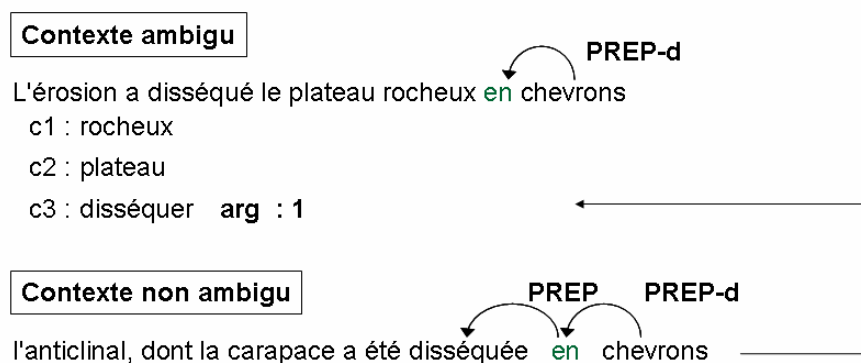
Figure 24: Ambiguïté de rattachement des prépositions

Les indices utilisés pour résoudre les ambiguïtés sont les contextes non ambigus du corpus (*arg*) et la propriété de sous-catégorisation syntaxique (*pEndo* et *pExo*). Les sous-catégorisations syntaxiques sont les groupes des mots qui sont souvent reliés par une relation (par exemple, [disséquer, en] [donner,à] [taxe,sur] [apte,à]). Syntex obtenu les propriétés de sous-catégorisation syntaxique à partir d'un corpus *partiellement* analysé syntaxiquement par d'acquisition les probabilités de sous-catégorisation sur deux types de ressource.

- **Ressource endogène** : acquise au moment de l'analyse à partir du corpus en cours d'analyse (*pEndo*)
- **Ressource exogène** : construite préalablement à partir d'un « gros » corpus d'apprentissage (140 M mots du Monde), utilisée pour chaque corpus (*pExo*)

Alors, l'algorithme PREP contient deux étapes comme suivant:

- Recherche des candidats recteurs
 - Point de départ : la préposition (le régi)
 - Direction : gauche
 - Noms, participe passé, adjectif, verbe
 - Et acquisition de propriétés de rection dans les contextes non ambigus (indices *arg* et *pEndo* endogènes calculés sur le corpus)
- Sélection d'un candidat
 - Affectation des indices aux candidats
 - **arg** : nombre de fois que l'adjectif est régi par le candidat dans un contexte non ambigu
 - **pEndo** : probabilité endogène de sous-catégorisation (calculée sur le corpus d'analyse)
 - **pExo** : probabilité exogène de sous-catégorisation (calculée préalablement sur un corpus d'apprentissage de grande taille)
 - Choix de celui :
 - qui a le score d'indice le plus élevé
 - Ou du dernier

Figure 25: Sélection de candidat par **arg**

1.4. Construction du réseau de syntagmes

A partir des résultats de l'analyse syntaxique des phrases du corpus, un module d'extraction de syntagmes (ES) construit un réseau de mots et syntagmes, calculé à partir des relations de dépendance identifiées dans chacune des phrases. Nous décrivons dans cette section comment est construit ce réseau, qui fournira les données de base à l'analyse distributionnelle étendue.

1.4.1. Principe général

Dans un premier temps, pour chaque phrase, le module ES procède à l'identification des constituants syntaxiques maximaux (verbaux, nominaux, adjectivaux) que détermine la structuration en relation de dépendance. Pour chaque mot recteur, il construit un syntagme maximal en parcourant toutes les relations de dépendance syntaxique dont ce mot est la cible jusqu'à aboutir à des mots qui soit ne sont pas recteurs, soit sont tête d'un syntagme maximal déjà construit. La caractérisation de la structure d'un syntagme est la suivante :

- une tête, qui est constituée du mot recteur avec sa catégorie;
- une liste de expansions, chaque expansion étant un mot régi ou un syntagme dont la tête est un mot régi

On commence par les mots les plus « bas » dans la structure de dépendance. Puis, on construit le syntagme associé à un recteur dès que les syntagmes associés à ses régis ont été construits.

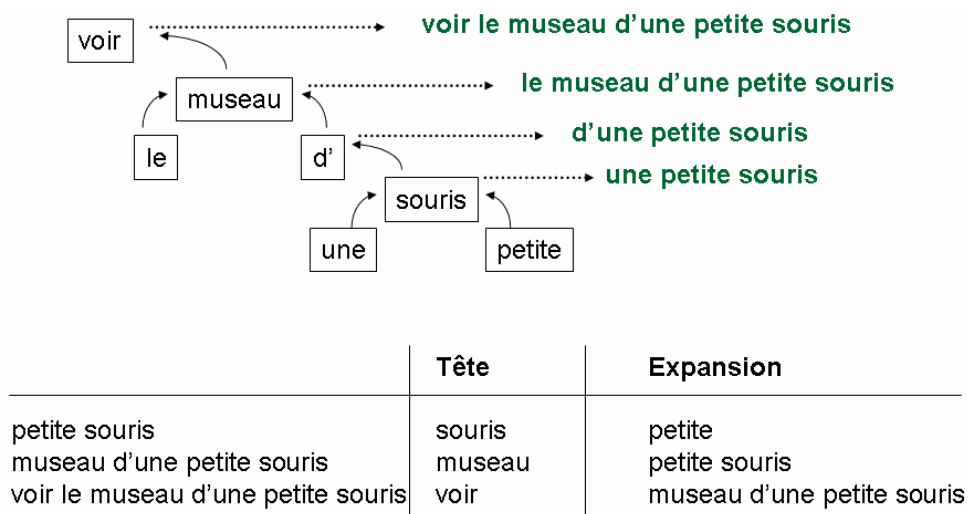


Figure 26: Exemple d'extraction des syntagmes

Dans un second temps, le module ES construit le réseau de dépendance en ajoutant pour chaque syntagme maximal différent rencontré : (1) un nœud dont le label est la forme normalisée du syntagme, (2) des liens vers les nœuds correspondant à ses expansions.

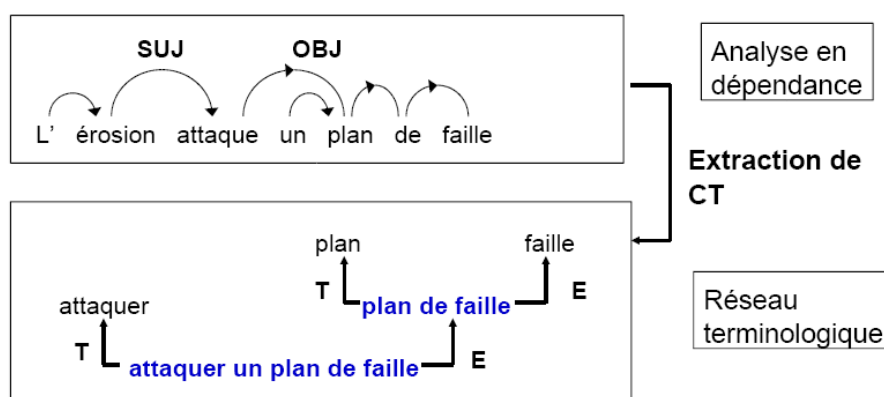


Figure 27: Réseau terminologie

Par conséquent, on a des syntagmes. Chaque syntagme est relié à sa Tête et à ses Expansions. Puis, on regroupe les syntagmes qui partagent la même Tête ou la même Expansion dans le corpus entier. Cela nous donne un réseau terminologie (voir Figure 27 et Figure 28).

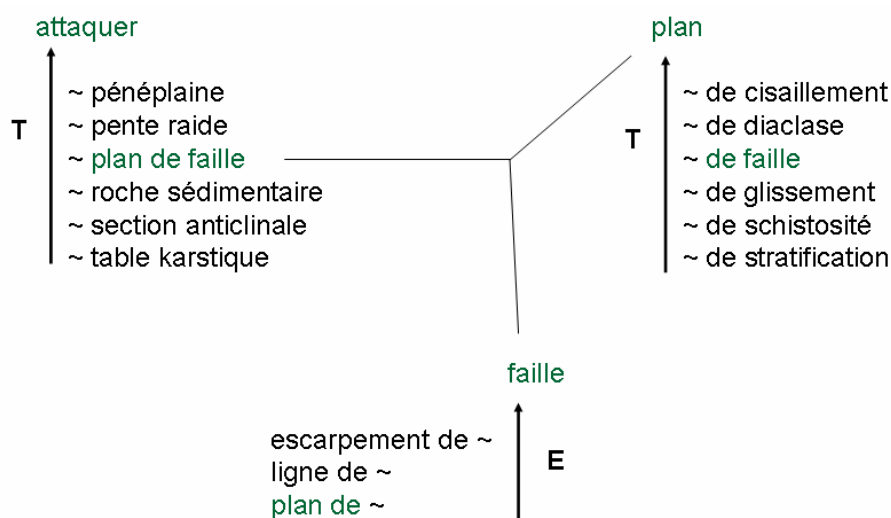


Figure 28: Réseau terminologie dans un corpus entier

Dans la Figure 26, on constate qu'il y a un syntagme "*d'une petit souris*". La Tête de ce syntagme est la préposition "de". On veut seulement extraire les syntagmes nominaux, verbaux et adjectivaux car les autres types de syntagmes apportent les même sens sémantiques que les syntagmes nominaux, verbaux ou adjectivaux. Pour seulement extraire les syntagmes nominaux, verbaux, adjectivaux, les syntagmes sont normalisés par quelques méthodes:

- Passif → actif
- « Saut » de préposition
- « Saut » de pronom relatif

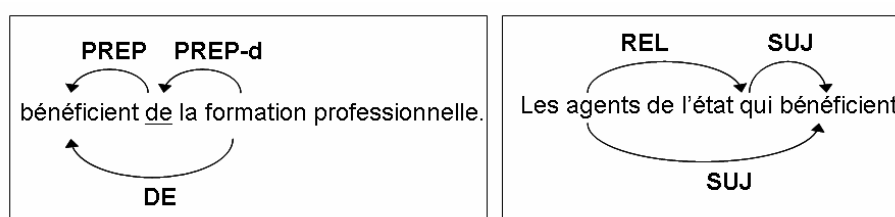


Figure 29: Exemple de normalisation

1.5. Upery

Upery (Bourigault, 2002) est un outil d'analyse distributionnelle. Il exploite l'ensemble des données présentes dans le réseau de mots et syntagmes construits par Syntex pour

effectuer un calcul des proximités distributionnelles entre ces unités. Ce calcul s'effectue sur la base des contextes syntaxiques partagés. Il s'agit d'une mise en œuvre du principe de l'analyse distributionnelle du linguiste américain Z. S. Harris, réalisée dans la lignée des travaux de H. Assadi (Assadi & Bourigault, 1996). L'analyse distributionnelle rapproche d'abord deux à deux des candidats termes qui partagent un grand nombre de contextes syntaxiques.

1.5.1. Données pour l'analyse distributionnelle

Le module d'analyse distributionnelle UPERY exploite l'ensemble des données présentes dans le réseau pour effectuer un calcul des proximités distributionnelles entre les mots et syntagmes du réseau. Ce calcul s'effectue sur la base des contextes syntaxiques partagés. Il s'agit d'une mise en œuvre du principe de l'analyse distributionnelle "à la Harris". Les données de l'analyse sont constituées ainsi :

(1) pour chaque syntagme du réseau ayant *une seule expansion*, le module construit une information élémentaire pour le calcul distributionnel. Celle-ci se formalise sous la forme d'un couple (contexte, terme) :

- le *contexte* est le couple constitué de la tête et de la relation de dépendance ; Il s'agit d'un contexte *simple*.
- le *terme* est l'expansion.

(2) pour chaque syntagme du réseau ayant *plus d'une expansion* (N expansions, N supérieur ou égal à 2), le module construit $N(N-1)$ informations élémentaires pour le calcul distributionnel. Pour chaque expansion E, il construit N-1 couples (contexte, terme), un pour chacune des autres expansions E' :

- le *contexte* est le couple constitué du syntagme réduit construit avec la tête et l'expansion E, et de la relation de dépendance R' correspondant à l'expansion E'; il s'agit d'un contexte *complexe*.
- le *terme* est l'expansion E'.

Par exemple, avec le syntagme nominal "formation professionnelle" on a une seule donnée { *professionnel* , (formation , ADJ) }. Avec le syntagme verbal "agent bénéficiaire de formation" on a deux données:

- $\{ agent, (bénéficiaire de formation, SUJ) \}$
- $\{ formation, (agent bénéficiaire, DE) \}$

1.5.2. Trois mesures de proximité

L'analyse distributionnelle rapproche d'abord deux à deux des termes qui partagent les mêmes contextes. L'analyse distributionnelle est symétrique, en ce sens qu'elle peut rapprocher aussi les contextes, en fonction des termes qu'ils partagent. Nous travaillons actuellement sur trois mesures qui permettent d'appréhender la proximité entre deux unités (termes ou contextes). Ces mesures ont l'avantage d'être simples à appréhender par l'utilisateur final, et de recouvrir des aspects différents et complémentaires des conditions dans lesquelles deux unités peuvent être jugées plus ou moins proches. Notre application cible est l'aide à la construction de ressources terminologiques ou ontologiques à partir de textes. Notre objectif est de fournir à l'utilisateur, avec ces quelques mesures de proximité, différents outils pour l'aider à trouver au plus vite les relations qu'il jugera les plus intéressantes.

On dispose pour un contexte donné de l'ensemble des termes (mots ou syntagmes) qui apparaissent dans ce contexte, et pour un terme donné l'ensemble des contextes (simples ou complexes) dans lesquels il apparaît. On définit la productivité d'un contexte et la productivité d'un terme ainsi :

- la *productivité d'un contexte* est égale au nombre de termes qui apparaissent dans ce contexte ;
- la *productivité d'un terme* est égale au nombre de contextes dans lesquels ce terme apparaît.

Les trois mesures de la proximité sont les suivantes :

Le coefficient a . Soient deux termes t_1 et t_2 . Le coefficient a est égal au nombre de contextes syntaxiques partagés par les deux termes. Cette mesure donne une première indication de la proximité entre deux termes, facile à interpréter. Mais l'expérience montre que cette mesure reflète de façon insatisfaisante la proximité : il faut tenir compte, d'un côté, de la productivité des contextes partagés (coefficient $prox$), d'un autre côté, du nombre de contextes que chaque terme a en propre (coefficients j_1 et j_2).

Le coefficient *prox*. Avec ce coefficient, nous visons à formaliser le fait si un contexte partagé par deux termes est très productif, sa contribution au rapprochement des deux termes est a priori plus faible que celle d'un contexte peu productif. Le coefficient *prox* est calculé ainsi :

- $prox = \sum_{c \in C} 1 / prod(c)^{1/2}$
- où C est l'ensemble des contextes partagés par t_1 et t_2 , et $prod(c)$ la productivité du contexte c

Contextes pour le terme : <i>murmure vésiculaire</i>	Termes pour le contexte : (patient présenter , OBJ)
(abolir , OBJ)	<i>amyotrophie</i>
(abolir à gauche , OBJ)	<i>détresse</i>
(abolition , DE)	<i>douleur</i>
(diminuer , OBJ)	<i>douleur thoracique</i>
(diminuer à gauche , OBJ)	<i>dyspnée</i>
(diminution , DE)	<i>fièvre</i>
(percevoir , OBJ)	<i>fracture</i>
	<i>hématome</i>
	<i>Syndrome</i>
productivité = 7	productivité = 9

Figure 30: Exemple de la productivité

contexte	prod	1/ prod(c) ^{1/2}
(patient développer , OBJ)	14	0,26
(développer , OBJ)	16	0,25
(tableau , DE)	35	0,17
(patient présenter , OBJ)	58	0,13
(épisode , DE)	71	0,12
(présenter , OBJ)	112	0,09
(apparition , DE)	169	0,08

Figure 31: Exemple de *prox*:
 $prox(détresse\ respiratoire, syndrome) = 1,10$

Les coefficients j_1 et j_2 Pour évaluer la proximité entre deux unités, il est important de tenir compte non seulement de ce qu'elles partagent, mais aussi de ce qu'elles ont en propre. Un certain nombre de mesures statistiques implémentent cette idée, sous

des formes diverses (e.g. information mutuelle, Jaccard, Anderberg). Ces mesures présentent presque toujours la particularité de "symétriser" la relation de proximité. Cette propriété, qui dans beaucoup de contextes d'application, constitue un avantage, voire une nécessité, nous est apparue finalement comme masquant un phénomène marquant à l'œuvre dans les corpus : la dissymétrie de la relation de proximité. Quand deux termes partagent un certain nombre de contextes en commun, il arrive le plus souvent que l'un des deux termes possède un nombre élevé de contextes, tandis que l'autre en possède beaucoup moins et en partage l'essentiel avec le premier. C'est pourquoi, nous caractérisons la proximité entre deux termes à l'aide de deux indices, simples et eux aussi faciles à interpréter : rapport entre le nombre de contextes partagés et le nombre total de contextes

- $j_1 = a / \text{prod}(t_1)$
- $j_2 = a / \text{prod}(t_2)$

2. Intégration

Pour l'intégration, TreeTagger n'est pas un logiciel "open source". Il ne publie pas aussi son API pour que les applications puissent utiliser son fonctionnement. Les deux outils Syntex et Upery sont développés en perl. L'entrée et sortie de Syntex et Upery sont les fichiers texte. Cela ne permet pas d'avoir une intégration "proche" par l'utilisation des API. Nous proposons une intégration de "batch" via le script du shell. Il s'agit des appels d'exécution des scripts du shell par le code Java.

L'intégration des outils traitement linguistique est une chaîne d'analyse syntaxique organisée en 5 phases :

- 1- *Préétiquetage* : ce module effectue la segmentation en phrases, la tokénisation en mots et le préétiquetage des mots.
- 2- *Étiquetage morphosyntaxique* : l'étiquetage est effectué par l'outil Treetagger de l'Université de Stuttgart.
- 3- *Conversion* : ce module effectue la conversion des sorties du Treetagger aux entrées attendues par l'analyseur Syntex ; il a été développé conjointement par l'ERSS et la société Synomia.
- 4- *Analyse syntaxique* : l'analyse syntaxique est réalisée par l'analyseur Syntex ; il a été développé par l'ERSS.

- 5- *Analyse distributionnelle*: l'analyse distributionnelle est réalisée par l'analyseur Upery; il a été développé par l'ERSS.

La sortie de la phase d'analyse HTML de crawl est l'ensemble des fichiers texte. Avant d'être traité par les outils linguistiques, ces fichiers sont prétraités par 3 scripts perl.

- Segmentation en phrases et en mots: ce traitement est nécessaire car l'entrée de TreeTagger est un fichier texte, chaque ligne contient un mot ou un point, une virgule, un point-virgule,....
- Pré-étiquetage des éléments communs: on détecte quelques formes spécifiques dans le texte comme: la date, les adresses email, les URL, la monnaie, les mesures,... Ce pré-étiquetage est réalisé par un script perl et un ensemble des expressions régulières.
- Pré-étiquetages des groupes de mots communs: TreeTagger ne peut pas détecter les groupes de mots comme: c'est-à-dire, d'un côté, nulle part, peu ou prou, tôt ou tard... Pour avoir un résultat plus précis dans les phases suivantes, ces pré-étiquetages sont nécessaires. On utilise un script perl et 6 fichiers de ressources (fr-preproc.ADV, fr-preproc.ADVGP, fr-preproc.CON, fr-preproc.DET, fr-preproc.DIV et fr-preproc.PRP) pour pré-étiqueter les groupes de mots. Chaque fichier de ressource contient une liste des groupes de mots pré-déterminés par l'humain.

Dans les 5 phases de traitement, les 4 premières phases sont traitées par un script du shell et la dernière phase (exécution de Upery) est traitée par un script séparé. Le processus de traitement est: le code Java utilise les méta-données dans la base de données pour choisir le fichier ayant besoin de traitement. Chaque script va séparément traiter un ensemble de fichiers. Alors, on peut pareillement exécuter ces scripts car ils sont exécutés dans les processus de fond et l'avancement de traitement est suivi par l'interface d'utilisateur.

CONCLUSION

1. Résultat obtenu

Le crawl est testé dans plusieurs jours. Quelques erreurs ne sont trouvés qu'après un long temps d'exécution. La version finale fonctionne dans une semaine et le résultat répond la demande de projet: de 90.000 à 500.000 pages visitées par jour dont 8-13% des pages qui sont satisfaites la conditions accepté, 40-50% des pages non satisfaites les conditions, 31-37% des pages d'erreur. Le crawl peut aussi détecter les pièges de robot. Les pièges de robot sont souvent trouvés dans des forums et les sites qui affiche le temps d'accès dans les pages.

Pour les outils analyse linguistiques, le temps de traitement d'un fichier est beaucoup long par rapport au traitement de crawl. On constate que TreeTagger consomme le plus de ressource dans son exécution. C'est peut-être à cause du le fichier de paramètre pour la langue française. Chaque fois de traitement d'un fichier, le programme doit charger 17Mo du fichier paramètre. C'est un point faible de l'intégration par les scripts du shell. On peut éviter ce problème si on a l'API de TreeTagger.

Un autre problème se trouve dans l'analyse linguistique est que dans le texte extrait à partir des pages HTML, il existe plusieurs des éléments non textuelle comme: le source code, les zones de menus et de liens... Cela ne donne pas des bons résultats dans l'analyse syntaxique et il existe beaucoup de mots bizarres. Pour l'instant, nous acceptons ce résultat mais dans l'avenir, il faut améliorer le module d'analyse de pages HTML pour qu'il puisse enlever les zones non informant.

2. Conclusion

En conclusion, l'objectif principal de ce travail a été atteint. Il s'agissait, nous le rappelons, de construire un crawl focalisé et appliquer les analyses linguistiques pour la construction des corpus du Web. Le crawl est construit en respectant les normes de crawl du Web: respecte de la politesse, le piège de robot, la parallélisme,... La construction du germe de départ est spécifique pour la recherche des pages dans des

domaines particuliers. Le crawl est testé avec deux germes de départ pour les mots "*graphie/graphes*" et "*métaphore/métaphores*". L'interface d'utilisateur de crawl est facile à utiliser. Les outils d'analyse linguistiques sont bien intégrés dans le système pour traiter les pages du web. Un pré-traitement est donné avant de passer ces outils. Il reste encore quelques questions pour les résultats des outils linguistiques pour analyser les pages Web (ces outils sont souvent utilisé avec les textes formel)

Ce stage a été très enrichissant d'un point de vue professionnel car j'ai découvert le travail du domaine de recherche d'information et de traitement automatique de la langue naturelle (TALN). J'ai réalisé que la constitution d'un crawl focalisé qui prend énormément de temps. J'ai également pu mettre en oeuvre les compétences informatiques et linguistiques dans un projet réel et non juste une maquette comme les projets universitaires. De plus, j'ai eu l'opportunité d'approcher les bons outils et les techniques de TALN.

BIBLIOGRAPHIE

- [1] Baeza-Yates, R. and Castillo, C. Balancing volume, quality and freshness in web crawling. In *Soft Computing Systems – Design, Management and Applications*, pages 565–572, Santiago, Chile. IOS Press Amsterdam, 2002
- [2] Baeza-Yates, R., Castillo, C., Marin, M. and Rodriguez, A. Crawling a Country: Better Strategies than Breadth-First for Web Page Ordering. In *Proceedings of the Industrial and Practical Experience track of the 14th conference on World Wide Web*, pages 864–872, Chiba, Japan. ACM Press, 2005.
- [3] Bourigault D., Assadi H. Analyse syntaxique et analyse statistique pour la construction d'ontologie à partir de textes, in Charlet J, Zacklad M., Kassel G., Bourigault D. édts. *Ingénierie des connaissances. Tendances actuelles et nouveaux défis*. Editions Eyrolles/France Telecom, Paris, 2000.
- [4] Bourigault D, Fabre C. Approche linguistique pour l'analyse syntaxique de corpus. *Cahiers de grammaire*, 25, 131-151, Université Toulouse le Mirail, 2000.
- [5] Bourigault D., Lame G. Analyse distributionnelle et structuration de terminologie. Application à la construction d'une ontologie documentaire du Droit. *Revue Traitement automatique des langues*, n° 47:1, Hermès, Paris, 2002.
- [6] Castillo, C. Effective Web Crawling. PhD thesis, University of Chile, 2004
- [7] Chakrabarti, S., van den Berg, M., and Dom, B. Focused crawling: a new approach to topic-specific web resource discovery. *Computer Networks*, 31(11–16):1623–1640, 1999.
- [8] Cho, J., Garcia-Molina, H., and Page, L. "Efficient crawling through URL ordering". In *Proceedings of the seventh conference on World Wide Web*, 1998.
- [9] Cho, J. and Garcia-Molina, H. Effective page refresh policies for web crawlers. *ACM Transactions on Database Systems*, 28(4), 2003.
- [10] Heydon, A. and Najork, M. Mercator: A scalable, extensible Web crawler. *World Wide Web Conference*, 2(4):219–229, 1999.

- [11] Koster, M. Guidelines for robots writers, 1993.
- [12] Koster, M. Robots in the web: threat or treat ?. *ConneXions*, 9(4), 1995.
- [13] Koster, M. A standard for robot exclusion, 1996.
- [14] Marc Najork and Janet L. Wiener. Breadth-first crawling yields high-quality pages. In *Proceedings of the Tenth Conference on World Wide Web*, pages 114–118, Hong Kong, May 2001.
- [15] Shkapenyuk, V. and Suel, T. Design and implementation of a high performance distributed web crawler. In *Proceedings of the 18th International Conference on Data Engineering (ICDE)*, pages 357-368, San Jose, California. IEEE CS Press, 2002.

ANNEXE

Tableau 1 : liste des catégories morphosyntaxiques

Adj		
	Adj??	adjectif de genre et nombre indéterminés
	Adj?P	adjectif de genre indéterminé et de nombre pluriel
	Adj?S	adjectif de genre indéterminé et de nombre singulier
	AdjFP	adjectif de genre féminin et de nombre pluriel
	AdjFS	adjectif de genre féminin et de nombre singulier
	AdjM?	adjectif de genre masculin et de nombre indéterminé
	AdjMP	adjectif de genre masculin et de nombre pluriel
	AdjMS	adjectif de genre masculin et de nombre singulier
Adv		
	Adv	adverbe
	AdvGP	groupe prépositionnel adverbial
CCoord		
	CCoord	Conjonction de coordination
	CCoordCAT	Conjonction de coordination (où CAT est la catégorie des éléments coordonnés, quand la relation de coordination a été identifiée)
CSub		
	CSub	conjonction de coordination
Det		
	Det	déterminant
	Det??	déterminant de genre et nombre indéterminés
	DetFS	déterminant de genre féminin et de nombre singulier
	DetMP	déterminant de genre masculin et de nombre pluriel
	DetMS	déterminant de genre masculin et de nombre singulier
	DetNum	déterminant numérique
Elim		
	Elim	catégorie éliminatoire
xxx		
	NUM	
Nom		
	Nom??	nom de genre et nombre indéterminés
	Nom?P	nom de genre indéterminé et de nombre pluriel
	Nom?S	nom de genre indéterminé et de nombre singulier
	NomFP	nom de genre féminin et de nombre pluriel
	NomFS	nom de genre féminin et de nombre singulier
	NomInc	nom inconnu

	NomM?	nom de genre masculin et de nombre indéterminé
	NomMP	déterminant de genre masculin et de nombre pluriel
	NomMS	déterminant de genre masculin et de nombre singulier
NomPr		
	NomPr	nom propre
	NomPrXXInc	nom propre inconnu (n'appartenant pas aux dictionnaires)
	NomPrXXPrénom	prénom
Nom		
	NomXXAdr	adresse
	NomXXDate	date
	NomXXHeure	heure
	NomXXMail	adresse email
	NomXXMes	mesure
	NomXXMon	monnaie
	NomXXNum	numérique
	NomXXTitre	titre
	NomXXUrl	url
Ppa		
	Ppa??	participe passé de genre et de nombre indéterminés
	PpaFP	participe passé de genre féminin et de nombre pluriel
	PpaFS	participe passé de genre féminin et de nombre singulier
	PpaMP	participe passé de genre masculin et de nombre pluriel
	PpaMS	participe passé de genre masculin et de nombre singulier
	PpaMSp	participe passé passif (été)
Ppr		
	Ppr	participe présent
	Pprp	participe présent passif
Prep		
	Prep	préposition
PrepDet		
	PrepDet	"de" non désambiguïsé Det ou Prep)
Pro		
	Pro	pronom personnel
ProRel		
	ProRel	pronom relatif
Typo		
	Typo	marque de typographie
	TypoCAT	Virgule coordonnante (où CAT est la catégorie des éléments coordonnés, quand la relation de coordination a été identifiée)
VCONJ		
	VCONJP	verbe conjugué au pluriel
	VCONJPP	verbe conjugué au pluriel au passif
	VCONJS	verbe conjugué au singulier
	VCONJSP	verbe conjugué au singulier au passif
VINF		

	VINF	verbe à l'infinitif
	VINFp	verbe à l'infinitif au passif

Tableau 2 : liste des relations de dépendance

Un certain nombre de cas ne sont pas encore traités de façon satisfaisante. Ils sont marqués :-()

ADJ			
	Nom	Adj	chat gris
		Ppa	chat allongé
		Ppr	chat miaulant
	NomPr	Adj	Jean, heureux d'être là :-()
		Ppa	Jean, allongé sur le canapé :-()
		Ppr	Jean, observant le chat :-()
	Pro	Adj	celui, heureux :-()
		Ppa	celui vu :-()
		Ppr	celui allant :-()
ADV			
	Adj	Adv	très rapide
	Adv		très rapidement
	Det		environ 10000
	Nom		ex- président
	NomPr		ex- KGB
	Ppa		souvent vu
	Ppr		voyant souvent
	VCONJ		court vite
	VINF		courir vite
ATTO			
	Ppr	Adj	rendant joyeux le chat
		Ppa	rendant énervé le chat
	VCONJ	Adj	rend joyeux le chat
		Nom	a été nommé directeur
		NomPr	estsurnommé Milou
		Ppa	rend énervé le chat
	VINF	Adj	rendre joyeux le chat
		Nom	être nommé directeur
		Ppa	rendre énervé le chat
ATTS			
	Ppr	Adj	étant gris
		Nom	étant le chat
		Ppa	semblant énervé
	VCONJ	Adj	est gris
		Nom	est le chat
		NomPr	est Paris
		Ppa	semble énervé
		ProRel	que sont les chats
	VINF	Adj	être gris
		Nom	étant le chat
		NomPr	être Paris
		Ppa	sembler énervé
		ProRel	que peuvent être les chats
AUX			
	Ppa	Ppa	a été vu

	Ppr		étant vu
	VCONJ		a vu
	VINF		avoir vu
COMP			
	CSub	Adj	aussi malin que rapide
		Adv	plus que souvent
		Nom	autre que le chat
		NomPr	moins que Paris
		Ppa	aussi rapide qu'énervé
		Prep	moins qu'avec le chat
		Pro	moins que celui
		VCONJ	vouloir qu'il soit
		VINF	que boire
CPL			
	Adj	CSub	aussi grand que
	Adv		plus que
	Nom		le fait que
	NomPr		:- (
	Ppa		plus énervé que
	Ppr		:- (
	Prep		autant à Marie qu'à Jean
	PrepDet		:- (
	VCONJ		c' est ici que
	VINF		:- (
DET			
	Nom	Det	le chat
	NomPr		la France
	Pro		le mien, la nôtre
EPI			
	Nom	Nom	le coin cuisine
		NomPr	le chat Mistigri
		Pro	:- (
	NomPr	Nom	Toulouse II
		NomPr	San Antonio
I_ADJ			
	Typo	Typo	, rapide ,
I_ADV			
	Typo	Typo	, lentement ,
I_PREP			
	Typo	Typo	, avec le chat ,
NNPR			
	NomPr	Nom	Monsieur Dupont
		NomPr	Jean Dupont
NOMPREP			
	Prep	Adv	:- (d'autant
		Nom	avec le chat
		NomPr	avec Marie
		Ppr	en mangeant
		Pro	avec lui
		ProRel	pour lequel
		VINF	pour prendre
	PrepDet	Nom	du chat
		NomPr	du Liban
		VINF	de la voir
OBJ			
	Ppr	CSub	voyant que
		Nom	voyant le chat

		NomPr	voyant Marie
		Pro	le voyant
		VINF	voulant manger
	VCONJ	CSub	il voit que
		Nom	il voit le chat
		NomPr	il voit Marie
		Pro	il le voit
		ProRel	le chat qu'il voit
		VINF	il veut voir
	VINF	CSub	voir que
		Nom	voir le chat
		NomPr	voir Marie
		Pro	le voir
		ProRel	que je tente de voir
		VINF	laisser faire
OBJ1			
	Ppr	Nom	en voyant le chat manger
		NomPr	en voyant Marie manger
		Pro	en la voyant manger
	VCONJ	Nom	je vois le chat manger
		NomPr	je vois Marie manger
		Pro	je la vois manger
	VINF	Nom	voir le chat manger
		NomPr	voir Maire manger
		Pro	la voir manger
PAR			
	Typo	Typo	()
PREP			
	Adj	Prep	facile à
	Nom		chat de
	NomPr		Afrique du Sud
	Ppa		équipé de
	Ppr		regardant vers
		Pro	lui donnant
	Pro	Prep	celui de
	VCONJ		il mange avec
		Pro	je lui donne
		ProRel	auquel je tiens
	VINF	Prep	manger
		Pro	pour lui donner
		ProRel	auquel je veux parler
REF			
	Ppr	Pro	se voyant
	VCONJ		il se voit
	VINF		se voir
REL			
	ProRel	Nom	l'homme qui
		NomPr	Marie qui
		Pro	celui qui
SUJ			
	VCONJ	Nom	le chat mange
		NomPr	Marie Mange
		Pro	il mange
		ProRel	qui mange