

Université Montpellier II
Licence IUP GMI

Rapport TER

**Extraction d'informations lexicales et
thématiques à partir de sites Web**

Recherche de collocations et de cooccurrences

Thibaud ZAMORA
Serge BARBERIS
Sébastien MARTIN
Céline BANCAREL

Tuteurs:
Mathieu LAFOURCADE
Didier SCHWAB

Montpellier, mars-juin 2003

Résumé

Ce rapport présente notre travail de recherche effectué lors de notre stage de recherche dans le cadre de notre licence au sein de l'IUP GMI de l'université de Montpellier. Nous avons dû développer des algorithmes capables de trouver des collocations et des cooccurrences pour un terme donné. Il présente notre approche, les différents outils utilisés ainsi que nos résultats.

Abstract

This report presents our research task carried out at the time of our training course of research within the framework of our licence IUP GMI of the university of Montpellier. We developed algorithms able to find collocations and co-occurrences for a given term. It presents our approach, the various tools used, and our results.

Mots clés

collocation, cooccurrence, champ sémantique, langage naturel, java

Keywords

collocation, co-occurrence, semantic field, natural language, java

Remerciements

Nous tenons à remercier tout particulièrement nos tuteurs Mathieu Lafourcade et Didier Schwab qui nous ont permis de travailler sur ce sujet particulièrement intéressant. Leur aide nous a été précieuse et leurs conseils éclairés.

Table des matières

1	Outils	6
1.1	Récupération des pages webs	6
1.2	Pré-traitement	7
1.2.1	Page francophone	7
1.2.2	Lemmatisation	7
1.3	Pondération	8
1.4	Dictionnaire / Anti-dictionnaire	8
1.4.1	Arbre à lettres	9
1.4.2	Automate	9
1.4.3	Table de hachage	10
1.4.4	Comparatif	10
1.5	Lemmatiseur	11
2	Recherche de collocations	14
2.1	Objectif	14
2.2	Notre algorithme	14
2.3	Nos résultats	17
3	Recherche de cooccurrences	19
3.1	Objectif	19
3.2	Principe	19
3.3	Recherche des entités lexicales	20
3.4	Pré-traitement	20
3.5	Différenciation des champs lexicaux	21
3.5.1	Graphe	21
3.5.2	Liste	23
3.5.3	Conclusion	26
4	Algorithmes généraux et architecture	28
5	Résultats complémentaires	30
5.1	Amour	30
5.2	Chaos	32

5.3	Chirurgie esthétique	34
5.4	Dragon	35
5.5	Geek	36
5.6	Google	37
5.7	Mythologie	39
5.8	Napoléon	41
5.9	Quidditch	42
5.10	Tigre	44

Introduction

Dans le cadre d'un système d'indexation textuelle (par exemple, articles de presse), nous constatons qu'un certain nombre de termes sont inconnus et ne sont présents dans aucun dictionnaire. Il s'agit, en particulier, de noms d'entreprises, de personnes, de produits. De plus, les langues évoluent - en particulier le français. Des mots nouveaux apparaissent, et des mots déjà existants ont des sens nouveaux ou leur sens est modifié.

Nous cherchons, à partir de sites webs référencés par des moteurs de recherche (comme Google <http://www.google.fr>), à extraire pour un terme inconnu une liste pondérée de termes associés. Certains sites proposent déjà ce type de services (par exemple : Vivisimo <http://www.vivisimo.com>)

Ainsi, pour le terme Usinor, nous pourrions procéder ainsi :

- lancer une requête à google
- extraire les termes des quelques premières pages
- éventuellement fusionner les ensembles de termes en essayant de tenir compte de la polysémie
- rendre un résultat, par exemple : ((acier 10) (métallurgie 4) (Sacilor 2))

Un terme candidat peut correspondre à plusieurs thèmes différents pour une même cible. Le mot peut avoir plusieurs champs sémantiques, il faut alors les différencier et proposer des listes de termes pour chaque sens rencontré. La liste résultat fera référence aussi bien à des termes connus (stockés dans un dictionnaire) qu'à des termes inconnus (qui feront aussi l'objet d'une recherche). Ainsi, au fur et à mesure des recherches, des termes seront associés pour former un réseau de termes.

Chapitre 1

Outils

Afin de travailler sur un corpus de textes important, nous avons choisi d'utiliser la plus grosse mine de documents à notre disposition : Internet. Cette base textuelle possède un nombre important de documents mais leur qualité est variable, des pages peuvent être en même temps dans plusieurs langues, des fautes d'orthographe présentes. De plus l'Internet est en constante évolution, des documents trouvés aujourd'hui peuvent disparaître demain et de nouveaux peuvent apparaître.

1.1 Récupération des pages webs

Nous devons récupérer les pages webs correspondant au terme à traiter. Pour cela, nous avons décidé, pour commencer, d'interroger un moteur de recherche : Google. Nous avons choisi Google car il référence un nombre de pages important, ses résultats sont pertinents (et non "sponsorisés"). Notre algorithme peut être appliqué à n'importe quel moteur de recherche, en effet si nous le souhaitons, nous pouvons arrêter d'utiliser Google, et nous pencher sur un autre moteur. De plus, si le besoin s'en fait sentir nous pouvons utiliser des encyclopédies en ligne (comme celle de club-internet <http://www.club-internet.fr/encyclopedia>) ou bien appliquer nos algorithmes à d'autres sources textuelles (revue, roman, journaux ...).

Puis nous devons extraire les adresses des pages résultats. Une fois ces pages récupérées, nous les stockons dans un dossier, correspondant à la recherche.

Mais une page web "brute" ne nous est pas utile, de nombreuses informations, notamment de mise en page, doivent être modifiées (car elles contiennent du bruit). De plus certains caractères sous forme d'entités HTML (tel que *Éeacute* ; représentant *é*) se doivent d'être remplacés.

Nous avons donc mis au point un algorithme, capable de réaliser cela.

Voici le résultat d'une telle transformation pour <http://www.google.fr> :

les . représentent les balises séparantes (celles qui servent à autre chose que la modification de la casse) et tout ce qui sépare deux mots (comme les . , ; ! ...).

. France . Web . Images . Groupes . Répertoire . Recherche avancée . Préférences . Outils linguistiques . Rechercher dans . Web . Pages francophones . Pages . France . Publicité . Google Toolbar . À propos de Google . Google . com English . 2003 Google . Nombre de pages Web recensées par Google . 3 . 083 . 324 . 652 .

1.2 Pré-traitement

1.2.1 Page francophone

Une fois les pages transformées en texte utile, nous devons supprimer celles en langues étrangères - principalement en anglais. Nous testons donc si la page contient divers mots tels que *the*, *and*, *where*, *we* ..., et si elle comporte plus de 10 ou plus de 5 % de ces mots, nous considérons que la page n'est pas en français, et donc nous ne la traitons pas.

Voici pour différents termes, la différence du nombre de pages avant et après traitement :

FIG. 1.1 – Nombre de pages

	Avant traitement	Après traitement
ange	50	30
barrage	98	51
elfe	96	63
paradist	49	41
rose	100	58
...

1.2.2 Lemmatisation

Ensuite, dans les textes, nous identifions les différentes “entités textuelles”. Par exemple, est-ce que des mots composés tels que *pomme de terre* sont présents ?

Puis nous supprimons les mots “vides de sens” présents dans notre anti-dictionnaire, comme *pour*, *par*, *le*, *que* ... ainsi que les mots liés à l'Internet-même : *html*, *pages*, *lien*, *accueil*, *menu* ...

De la liste résultante, nous ne gardons que les noms, les adjectifs et les mots inconnus. (Au début, nous pensions garder aussi les verbes, mais après différents tests, nous avons vu que cela ajoutait du bruit à nos résultats.)

Enfin nous lemmatisons les mots restants, c'est à dire, que dans la mesure du possible, les adjectifs sont mis sous forme singulier, masculin et les noms au singulier.

Voici un exemple de lemmatisation : **Texte de départ** : *Pour faire une bonne soupe de légumes, il nous faut une pomme de terre, 2 carottes, 500g de courge et 2 poireaux*
Identification des entités textuelles : *Pour, faire, une, bonne, soupe, de, légume, il, nous, faut, une, pomme de terre, 2, carottes, 500g, de, courge, et, 2, poireaux*
Lemmatisation et Suppression des mots vides : *bonne, soupe, légume, pomme de terre, carotte, 500g, courge, poireau*

1.3 Pondération

Une fois notre texte lemmatisé, il est transformé en vecteur de mots de la forme suivante (*mot, poids*) où *poids* correspond, dans un premier temps, au nombre de fois où apparaît le mot dans la page.

Ensuite, nous appliquons sur chaque vecteur, notre fonction de pondération dont voici la formule :

$$Poids(mot) = hit(mot) / \sum hit(mot \text{ de la liste}) \quad (1.1)$$

Une fois nos mots pondérés, nous supprimons les mots ayant une pertinence plus faible que leur utilisation normale ; ceux ne respectant pas la formule ci-dessous.

$$Pertinence(mot) > hitGoogle(mot) / 60000000 \quad (1.2)$$

Le nombre de hit d'un mot sur Google correspond au nombre de pages où apparaît ce mot et non le nombre de fois total, en effet un même mot peut apparaître plusieurs fois dans la même page. En effet, si le mot est moins utilisé dans notre page que sur le web, cela signifie qu'il n'est pas déterminant pour notre mot cible. (60 millions correspond au nombre estimé de pages francophones <http://docs.abondance.com/question10.html>)

1.4 Dictionnaire / Anti-dictionnaire

Dans l'étape précédente, nous avons ressenti le besoin d'avoir à notre disposition un dictionnaire, ainsi qu'un anti-dictionnaire, afin d'obtenir des informations sur un terme donné (mot "vide" ?, poids ...). Un dictionnaire est un outil nous permettant de savoir si un mot existe et d'en connaître son poids. L'anti-dictionnaire quand à lui regroupe tous les mots vides, ainsi que les mots liés au web (*accueil, html, lien ...* il nous permet donc de nettoyer nos textes avant de les traiter.

Il est impensable de garder ces (anti-)dictionnaires sous forme de fichiers textes, et de parcourir chaque fois que nécessaire les fichiers. Cela provoquerait beaucoup trop d'accès disque.

Plusieurs solutions pour gérer ces outils sont possibles :

- arbre à lettres

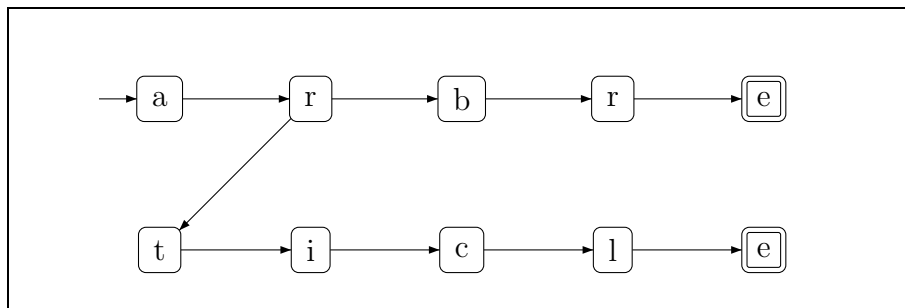
- automate
- table de hachage

1.4.1 Arbre à lettres

La structure en arbre à lettres présente de nombreux avantages. Tester si un mot est présent (et en obtenir les informations) se fait en $O(n)$ où n correspond au nombre de lettres du mot. De plus, nous pouvons trouver aisément les mots de racine commune.

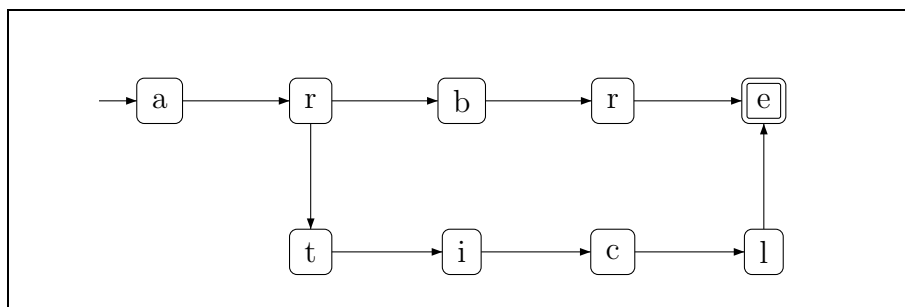
Mais cette représentation prend beaucoup de place en mémoire.

Voici un exemple d'arbre à lettres :



1.4.2 Automate

Les automates présentent les mêmes avantages que les arbres à lettres (compression des mots à racine commune) plus une compression des mots à suffixe commun. Ainsi nous pouvons connaître les mots ayant le même suffixe. Mais la place occupée en mémoire est encore trop importante et nous ne pouvons plus obtenir des informations relatives à un mot donné (du moins pas de manière facile) car la dernière lettre d'un mot, correspond à un état dans l'automate et cet état peut lui même correspondre à différentes lettres de différents mots, nous ne pouvons donc pas y stocker les informations relatives à un mot précis.



1.4.3 Table de hachage

Grâce à cette structure, la recherche de la présence d'un mot se fait en $O(1)$ pour un nombre important d'entrées. Cette structure présente l'inconvénient que nous ne pouvons pas connaître l'ensemble des mots ayant un même suffixe ou préfixe. Mais pour notre travail qui est la recherche de collocations et de cooccurrences, nous n'avons pas besoin de connaître de tels ensembles.

FIG. 1.2 – Dictionnaire (table de hachage)

magique	191000
magouille	9750
magouiller	716
magot	9850
magma	23300

1.4.4 Comparatif

Des tests ont été réalisés avec les différentes structures.

FIG. 1.3 – Tests réalisés sur un Athlon 1500+ - 512 DDRAM pour un fichier de 100 000 entrées

	Arbre à lettre	Table de hachage
Fichier -> Mémoire	6 sec.	3 sec.
Mémoire -> Fichier	3 sec.	1 sec.
Mémoire occupée	13.6%	4.8%
Recherche d'un mot	<1 sec.	<1 sec.

Conclusion

Pour conclure, nous pouvons dire que la structure en arbre à lettres est optimale pour traiter un dictionnaire, ainsi de nombreuses racines de mot se factorisent et permettent de gagner de la place en mémoire, mais il faut que le dictionnaire comporte un nombre très important d'entrées. Actuellement, vu la taille de nos dictionnaires (moins de 500000 entrées), la structure en table de hachage est la meilleure. C'est pourquoi nous l'avons choisi, tout en gardant à l'esprit que dans l'avenir, nous devrons certainement changer de structure.

1.5 Lemmatiseur

Notre lemmatiseur doit nous permettre de connaître rapidement si un mot est un lemme, savoir à quel(s) lemme(s) il est rattaché et de connaître sa nature grammaticale (*nom, verbe, adjectif, déterminant ...*). Comme nous pouvons le constater, les fonctionnalités ressemblent à celles des dictionnaires. Nous avons donc choisi une structure proche. En effet notre lemmatiseur comporte plusieurs tables de hachage. Chaque terme possède un identifiant unique.

La première contient tous les lemmes, une recherche dans cette table nous permet donc de savoir si un terme est un lemme ou non.

FIG. 1.4 – Table des lemmes

...
flemme
flemmer
avion
avionique
avionnette
avionneur
avion-cargo
avoir
...

La seconde fait correspondre les couples identifiant/mot.

FIG. 1.5 – Identifiant - mot

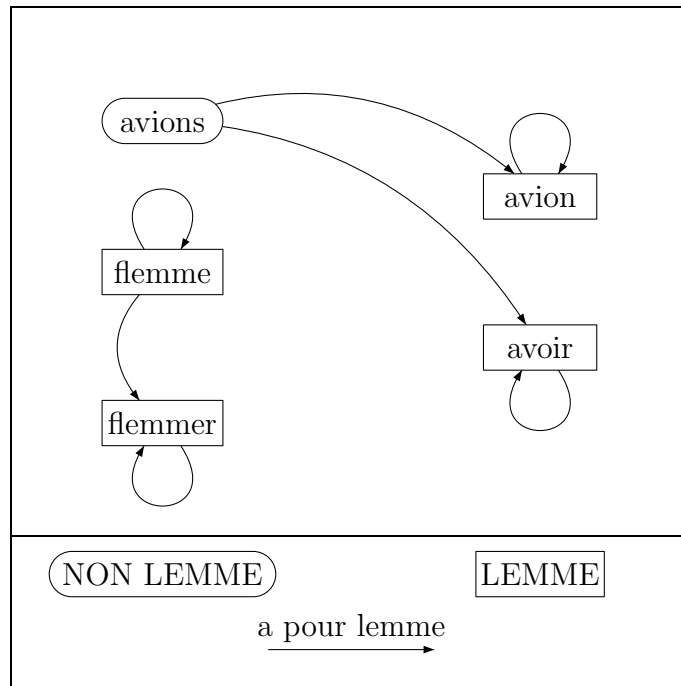
...	...
240959	flemme
240960	flemmer
4240	avion
42424	avionique
42429	avionnette
42431	avionneur
42432	avions
42438	avion-cargo
42445	avions-cargos
6	avoir
...	...

Enfin la dernière, permet de connaître la liste des lemmes auxquels est rattaché le terme passé en paramètre.

FIG. 1.6 – mot - liste des lemmes

...	...
flemme	((240960 . :VER) (240959 . :NOM))
flemmer	(240960 . :VER)
avion	((42420 . :NOM))
avionique	((42424 . :NOM))
avionneur	((42431 . :NOM))
avions	((6 . :VER) (42420 . :NOM))
avion-cargo	((42438 . :NOM))
avions-cargos	((42438 . :NOM))
avoir	((6 . :VER) (6 . :NOM))
...	...

Ainsi nous pouvons modéliser ceci :



Chapitre 2

Recherche de collocations

2.1 Objectif

Notre étude porte sur la recherche de signature lexicale d'un terme -à priori- inconnu. Dans une première étape, présentée ici, nous nous intéressons à la recherche de collocations pour un terme donné.

Qu'est ce qu'une collocation ?

L'encyclopédie de club-internet (<http://www.club-internet.fr/encyclopedia>) donne pour définition *LING. Association syntagmatique entre deux ou trois unités linguistiques. Chien est souvent en collocation avec méchant ; une collocation stable peut se lexicaliser comme chien de garde, pomme de terre, pied de biche.*

Une collocation pour un terme donné est donc une expression où apparaît ce terme. Les collocations sont d'usage courant. En effet "passes moi ton *téléphone portable*, je dois passer un *coup de fil* etc ..."

Force est de constater que la plupart des collocations ont des modèles communs. Dans une première étape, nous avons listé, de manière non-exhaustive, des modèles de collocations : *Adjectif Nom*, *Nom de Nom*, *Nom de la Nom*, *Nom au Nom*, *Nom du Nom*, etc ...

Nous avons ainsi mis en évidence une trentaine de modèles. Nous avons décidé de ne pas prendre en compte les collocations où interviennent des verbes. En effet, des collocations parasites apparaissaient (ex. *Toto dort ... Toto enseigne ...*).

2.2 Notre algorithme

Notre recherche des collocations se découpe en plusieurs étapes :

Dans un premier temps, nous devons repérer dans notre corpus, les différents modèles où intervient le terme recherché (appelé *cible*). Une fois l'ensemble des collocations possibles (appelée *candidat*) trouvé, nous devons les "valider".

Algorithm 1: Est-ce une véritable collocation ?

Données: cible : chaîne ; candidat : chaîne

Résultat: Le candidat est valide : booléen

$reste \leftarrow \text{candidat} - \text{cible}$

si *reste* ne contient pas au moins une lettre ou 2 chiffres **alors**

retourner *Faux*

sinon

si *reste* = *cible* **alors**

retourner *Faux*

sinon

si *aPourType*(*reste*, “nom”) ou *aPourType*(*reste*, “adjectif”) **alors**

si *aPourType*(*reste*, “verbe”) **alors**

retourner *Faux*

sinon

si *appartientAntiDico*(*reste*) **alors**

retourner *Faux*

sinon

si *hitGoogle*(*candidat*) > α

 ou *hitGoogle*(*candidat*)/*hitGoogle*(*cible*) > β % **alors**

retourner *Vrai*

sinon

retourner *Faux*

fin

fin

fin

sinon

retourner *Faux*

fin

fin

fin

Cet algorithme élimine donc dans un premier temps, les mots trop courts (n’ayant pas au moins une lettre ou deux chiffres). Au départ, nous gardions aussi ceux contenant un seul chiffre. Cependant des collocations parasites comme *1 ange ... 2 démon ...* apparaissaient - c’était le plus souvent des titres de paragraphes ou de chapitres.

Dans un second temps, nous vérifions si le reste de la collocation (ie. la collocation privée de la cible) ne correspond pas à la cible elle-même, car sur la toile, de nombreux sites, répètent les mêmes mots pour être “mieux” référencés par les moteurs de recherche.

Après, nous testons si le reste de la collocation a, dans sa liste de types grammaticaux, l'état de nom ou d'adjectif. Dans le cas contraire, nous ne validons pas la collocation. Mais des mots comme *passe* sont à la fois des noms (*une passe de football*) et des verbes (*un ange passe*) ; et comme indiqué plus haut nous ne gardons pas de telles collocations.

Enfin, si le mot n'appartient pas à notre anti-dictionnaire (qui comporte environ 900 entrées), nous testons si la collocation a de la pertinence.

Si la collocation revient plus de α fois en terme de hit sur Google, nous la validons. Nous avons défini α à par une recherche empirique. En effet, nous avons pris plusieurs termes, comme *Tintin*, et nous avons regardé ce que nous obtenions avec comme borne 1000, puis 500, et enfin 750. Avec 1000, il nous manquait des collocations qui nous semblaient importantes comme *Tintin au pays des soviets*. Nous avons donc testé avec 500, mais nous nous retrouvions alors avec des collocations parasites pour différents termes. Enfin nous avons testé avec 750, et ainsi nous avons trouvé *albums de Tintin* qui nous semblait une collocation très importante pour *Tintin* avec comme poids 753, ce qui a conforté notre choix de 750.

Mais le test précédent n'est pas suffisant. En effet, nous omettons certaines collocations peu fréquentes sur Internet comme *beignets aux pommes* pour *beignets*, nous avons donc décidé, pour des cas similaires de tester si le poids de la collocation par rapport au poids de la cible dépasse un certain taux : β (nous avons choisi ici 2%), et de valider ainsi de nouvelles collocations.

Pour qu'une collocation soit validée il faut donc qu'elle respecte cette formule :

$$Poids(collocation) > \min(Poids(cible) * \beta \%, \alpha) \text{ avec } \alpha = 750 \text{ et } \beta = 2\% \quad (2.1)$$

Si la collocation est valide, nous regardons si nous ne pouvons pas l'améliorer. Nous cherchons si elle-même n'est pas un membre d'une collocation plus importante. Si nous trouvons une collocation plus longue, plusieurs cas se présentent : Soit, la nouvelle collocation a un poids supérieur à γ % du poids de l'ancienne collocation, alors nous ne validons que la nouvelle collocation. Soit, elle a un poids inférieur à γ % de l'ancienne et alors nous validons les deux collocations. (Nous avons fixé γ à 60 % par différents tests)

Prenons un exemple, cherchons les collocations de *Tintin* dans *Tintin au pays de l'or noir*. Si la recherche n'était pas récursive, nous trouverions uniquement *Tintin au pays*. Mais grâce à notre appel récursif, nous trouvons aussi *Tintin au pays de l'or noir* comme étant une collocation de *Tintin*, mais on ne trouve pas *Tintin au pays de l'or* car son poids (nombre de hit sur google) est de 987 et celui de *Tintin au pays de l'or noir* de 967.

Cet algorithme pourrait être amélioré, en ajoutant différentes heuristiques. En particulier, si un terme est à la fois un nom (ou un adjectif) et un verbe, alors nous regardons si devant celui-ci nous avons un déterminant pour conclure que c'est un nom. Ainsi nous

pourrions récupérer *passe de football* comme collocation dans une phrase telle que *une passe de football ...*

2.3 Nos résultats

Pour chaque terme, nous disposions d'un corpus dans lequel nous avons cherché leurs collocations.

En voici les résultats :

FIG. 2.1 – Collocations

cible	nombre de pages	collocations trouvées
ange	30	Ange noir, Saint-ange, Ange-gardien, Petite ange, Mi-ange, Peau d'ange
barrage	51	Match de barrage, amont du barrage, premier barrage, projet de barrage, Grand barrage, barrage hydroélectrique
chocolat	40	fabrication du chocolat, chocolat noir, Salon du chocolat, Musée du chocolat, chocolat blanc
elfe	63	Site de l'elfe noir, Elfe noir, site de l'elfe noir, elfe noir, Jeune elfe
enfer	31	Enfer vert, Radio enfer, Boucan d'enfer, Michel au val d'enfer
paradis	41	Vanessa paradis, vanessa paradis, Petit paradis, paradis-fiscal, paradis fiscaux, Bienvenue au paradis
rose	58	Côte de granit rose, côte de granit rose, Rose-croix, rose-croix, Ordre de la rose, Rose croix, Ste-rose, Rose des vents, Carnet rose, Red rose, carnet rose, Blanc rose, éléphant rose
mac	37	mac os, Mac os, Pro mac, Mac g4, Mac g3, Mac portables, mac g4
disney	205	Walt disney, personnages de disney, films disney, Walt disney world, Studios disney, Disney village ...

A la vue de ces résultats, nous pouvons faire différentes remarques et critiques.

Tout d'abord, différentes collocations comme *elfe noir* apparaissent à plusieurs reprises, mais avec des casses différentes. Au début, nous avons décidé de fusionner les termes

qui différaient seulement par leurs casses. Mais nous avons vu que le sens de certaines collocations est dépendant de leur casse. En voici un exemple : *la jolie rose ...*, on parle de la fleur ; *la jolie Rose ...*, la fille qui s'appelle Rose est jolie. Un autre exemple *Paris* et *paris*, on peut faire des *paris* à *Paris* mais pas faire des *Paris* à *paris*.

Nous avons pensé à une nouvelle heuristique (que nous n'avons eu le temps de mettre en pratique) pour éviter d'avoir des doublons (différent uniquement par la casse ou par un accent). Nous testons si le complément de la collocation appartient au dictionnaire avec la casse courante ainsi qu'entièrement en minuscule. Différents cas apparaissent, si une seule des deux casses du mot appartient au dictionnaire, nous conservons cette écriture. Si aucune des deux, ou les deux, casses n'est présente dans le dictionnaire, nous gardons le mot tel quel. De plus l'utilisation d'un correcteur orthographique serait la bienvenue pour, par exemple, les problèmes d'accentuation.

Ensuite, dans les collocations, nous trouvons *Red rose*, nous pouvons croire à un mauvais traitement des pages anglaises, mais après une recherche sur Google, nous nous apercevons que cela correspond à du thé (<http://www.lheure-du-the.com/content/teacup/history.html>).

Enfin, nous pouvons regretter le manque de certaines collocations comme *ange déchu*, *chocolat au lait*, ou bien *barrage routier*, ... mais après une vérification manuelle des pages traitées, nous remarquons que ces termes n'apparaissaient pas dans nos sous-corpus, il faudrait donc appliquer notre algorithme sur des sous-corpus plus importants pour les trouver.

Une fois les collocations validées, elles sont ajoutées au dictionnaire afin de l'enrichir, et d'affiner nos recherches futures de cooccurrences.

Chapitre 3

Recherche de cooccurrences

3.1 Objectif

Maintenant que nous avons trouvé les collocations d'un mot, nous devons en extraire ses cooccurrences.

D'après l'encyclopédie de club-internet (<http://www.club-internet.fr/encyclopedia>), une cooccurrence est l'*apparition simultanée, dans un énoncé, de deux ou plusieurs unités linguistiques données : relation existant entre elles*. . Trouver les cooccurrences d'un mot donné revient donc à trouver le(s) champ(s) sémantique(s) de ce mot.

Par exemple un champ sémantique pour *Tintin* pourrait être *Hergé, bd, belge, milou,*

Mais des mots peuvent avoir plusieurs champs lexicaux différents, nous devons donc - dans la mesure du possible - les différencier.

3.2 Principe

Notre recherche de cooccurrences comprends diverses étapes.

Tout d'abord, pour chaque texte de notre corpus, nous indentifions les différentes entités lexicales de notre texte (Nous recherchons la présence ou non de collocations appartenant à notre dictionnaire). Ensuite, nous lemmatisons notre liste d'entités lexicales, afin d'avoir le moins de redondance possible. Puis nous enlevons, grâce à notre anti-dictionnaire, les mots vides (ainsi que les mots liés au web).

Ainsi nous obtenons pour chaque page de notre corpus une liste de mots. Nous devons ensuite pondérer et épurer nos listes, fusionner celles qui sont voisines et enfin en extraire nos résultats.

3.3 Recherche des entités lexicales

Pour trouver les différentes collocations présentes dans notre texte, nous parcourons notre texte, et pour chaque mot nous testons si il forme avec ses successeurs une collocation connue de notre dictionnaire (Nous cherchons uniquement les collocations de taille inférieure à 5 mots en effet peu de collocations dépassent cette taille, et augmenter -ou supprimer- cette limite alourdit notre algorithme).

Algorithm 2: Identification des différentes entités lexicales

Données: Un texte et le dictionnaire

Résultat: Une liste d'entités lexicales

$mots[] \leftarrow Decoupe(Texte, " ")$

```
pour  $i$  de 1 à  $Taille(mots)$  faire
    si  $mots[i] \neq cible$  alors
         $candidat \leftarrow mots[i]$ 
        pour  $j$  de  $i+1$  à  $i+6$  faire
             $candidat \leftarrow mots[j]$ 
            si  $Existe(candidat, dictionnaire)$  alors
                ajouter( $candidat$ , résultat)
                 $i \leftarrow j$ 
            fin
        fin
    fin
fin
retourner résultat
```

3.4 Pré-traitement

Ensuite nos listes sont lemmatisées, afin de garder uniquement les noms, les adjectifs (au masculin singulier si possible) et les termes inconnus. Enfin nous supprimons les mots appartenant à notre anti-dictionnaire.

Lors de la lemmatisation, nous supprimons les doublons et nous affectons à chaque mot de nos listes un poids correspondant au nombre de fois où apparait le mot dans la page étudiée (ie. le nombre de doublons du mot en question).

Ensuite nous pondérons nos listes et supprimons les mots ayant une pertinence inférieure à la pertinence du mot sur le web francophone (voir formules 1.1 et 1.2).

Enfin nous supprimons les mots n'apparaissant pas dans au moins $\alpha\%$ des pages, afin que les mots parasites, mal orthographiés soient supprimés (α est fixé à 4 %).

A la fin de cette étape nous nous retrouvons donc avec différentes listes ayant cet aspect :

... *visite* (0.014285714), *montagne* (0.014285714), *loisirs* (0.014285714), *hiver* (0.042857144),
partenaire (0.014285714), *ski* (0.014285714), *neige* (0.028571429) ...

3.5 Différenciation des champs lexicaux

Pour différencier les différents champs lexicaux, nous avons tester différentes approches.

3.5.1 Graphe

Principe

Dans un premier temps, nous avons suivi l'idée de Véronis [2], qui est de créer un graphe ayant pour sommets les différents mots de nos listes.

Une arête entre 2 mots (mot A, mot B) existe si les 2 mots appartiennent à une même liste, et elle a pour poids le résultat de cette équation :

$$Poids(arete) = presence(motA + motB) / min(presence(motA), presence(motB)) \quad (3.1)$$

Cette équation correspond au maximum des probabilités conditionnelles d'avoir le mot A sachant qu'on a le mot B (et vice-versa).

Au départ, nous avons pensé calculer ces probabilités conditionnelles sur Internet mais pour chaque arête et pour chaque sommet nous devions envoyer une requête à Google. Or notre graphe comporte plusieurs milliers de sommets et d'arêtes. Cette solution n'est donc pas exploitable.

Nous avons donc décidé d'utiliser uniquement notre corpus, et calculer le poids de nos arêtes en effectuant des calculs sur nos listes.

Une fois notre graphe pondéré, nous devons en extraire les composantes de fortes densités, mais actuellement la plupart des algorithmes capables de réaliser un tel partitionnement sont NP-difficiles et donc inexploitable. De plus d'après cet algorithme, le graphe obtenu n'est pas orienté, donc si le mot A est lié au mot B alors celui-ci est lui-même lié au premier. Or, par exemple, pour *marteau* il est interessant d'avoir *outil* mais pas forcément l'inverse.

Algorithme

Nous avons donc décider d'utiliser l'algorithme suivant :

Algorithm 3: Identification des différents champs lexicaux

Données: Graphe(X, E), cible, taux

Résultat: ensembles de mots

tant que possible faire

candidat \leftarrow *PlusProche(cible)*

Liste.ajout(candidat)

tant que possible faire

Liste.ajout(Voisins(Liste, TAU X))

fin

si *Taille(Liste)* > 5 **alors**

resultat.ajout(Liste)

 supprimer du graphe les mots de Liste

fin

fin

Cet algorithme regarde le terme le plus proche de la cible (arête de poids maximal), puis les voisins proches de ce terme, puis les voisins des voisins etc ... Si la liste de termes contient 5 termes ou plus, nous supprimons ces mots du graphe et nous recommençons. (Si la liste contient moins de 5 éléments, nous passons directement au terme suivant).

Ainsi, à la fin de l'algorithme, nous avons défini les différents champs lexicaux du mot *cible*.

Résultats

cendrillon	barrage	quimboiseur
conte (0.066245867)	lac (0.029568234)	best (0.007843138)
fée (0.065264866)	barrage (0.028365412)	chrétien (0.007843138)
nom (0.021668479)	construction (0.027613785)	pères (0.003921569)
Perrault (0.020654563)	longueur (0.023722177)	disciple (0.003921569)
cendrillon (0.019317413)	eau (0.020153038)	verset (0.003921569)
vie (0.01308328)	mètres (0.014136381)	difficile (0.003921569)
	retenue (0.012146007)	chant (0.003921569)
	ville (0.010492401)	foi (0.003921569)
	rivière (0.009866371)	ligne (0.003921569)
	aval (0.009550673)	bible (0.003921569)
	béton (0.008613556)	étymologie (0.003921569)
	hauteur (0.007563527)	SOS (0.003921569)
	année (0.0074701235)	jésus (0.003921569)
	eaux (0.006443997)	prière (0.003921569)
	ouvrage (0.0061222725)	réflexion (0.003921569)
		sorcellerie (0.0014005603)
		projet (5.628254E-4)
		sorte (4.6685344E-4)
		irrationnel (4.6685344E-4)
		mondes (4.6685344E-4)
		Antilles (4.6685344E-4)
		astrologie (4.6685344E-4)
		événements (4.6685344E-4)
		Amérique (4.6685344E-4)
		époque (4.6685344E-4)

3.5.2 Liste

Principe

Nous partons de l'heuristique suivante : dans une page donnée, issue de la recherche du mot cible, il n'existe qu'un seul champ lexical pour le mot cible.

A partir de cette heuristique, nous déduisons que pour séparer les différents champs lexicaux du mot cible, il suffit de fusionner les listes voisines.

Algorithme

Voici donc notre algorithme :

Algorithm 4: Identification des différents champs lexicaux

Données: Ensemble des listes, α , β

Résultat: ensembles de listes

tant que possible faire

 | Fusionner 2 listes semblables à plus de β %

fin

pour chaque Liste de l'ensemble faire

 | **si** $Taille(Liste) < \alpha$ **alors**

 | Supprimer la liste

 | **sinon**

 | Trier la liste

 | **fin**

fin

Après une recherche empirique, nous avons déterminé les différentes valeurs. Ainsi, nous avons affecté au α la valeur 5 et pour β 50%.

A la fin de cet algorithme, les pages ayant un champ lexical semblable sont fusionnées, et nous faisons ressortir les différents champs lexicaux du terme étudié.

Résultats de la recherche de cooccurrences

cendrillon	barrage	quimboiseur
Perrault (0.13636364)	mètres (0.09677419)	sorcellerie (0.06)
cendrillon (0.13636364)	photo (0.083333336)	Buffy (0.02)
enfant (0.05)	eau (0.047244094)	Angel (0.02)
Disney (0.048780486)	lac (0.031746034)	rural (0.02)
livres (0.045454547)	nom (0.023809524)	Berry (0.02)
images (0.045454547)	réservoir (0.023809524)	Wallonie (0.02)
contes (0.045454547)	vallée (0.023809524)	ville (0.02)
musical (0.03508772)	ville (0.023622047)	irrationnel (0.02)
chanson (0.033333335)	construction (0.02173913)	astrologie (0.02)
conte (0.023255814)	hauteur (0.02173913)	cartomancie (0.02)
vol (0.018867925)	fondation (0.02173913)	Antilles (0.02)
nom (0.018181818)	amont (0.02173913)	sorcier (0.02)
belle (0.016129032)	retenue (0.02173913)	dérivé (0.02)
compte (0.015384615)	centrale (0.021276595)	rite (0.02)
atelier (0.014492754)	visite (0.020833334)	vaudou (0.02)
personnage (0.012048192)	ouvrages (0.017857144)	anthropologue (0.02)
ami (0.011111111)	technique (0.01724138)	formes (0.02)
vie (0.0068593826)	aménagement (0.015873017)	Afrique (0.02)
fée (0.006802721)	million (0.015151516)	Amérique (0.02)
ville (0.006802721)	eaux (0.014492754)	Océanie (0.02)
	hydroélectrique (0.013333334)	Shakespeare (0.02)
	barrage (0.010869565)	Walt (0.02)
	rivière (0.010869565)	Disney (0.02)
	ouvrage (0.010869565)	Goethe (0.02)
	année (0.010869565)	sorcière (0.02)
	terre (0.010869565)	halloween (0.02)
	longueur (0.010869565)	balais (0.02)
	aval (0.010869565)	...
	prise (0.010869565)	tan (0.014423077)
	béton (0.010204081)	créole (0.009615385)
		fé (0.009615385)
		...
		wélélé (0.0048076925)
		charivari (0.0048076925)
		moucheron (0.0048076925)
		zouk (0.0048076925)
		...

3.5.3 Conclusion

Nous pouvons voir que les résultats des deux méthodes sont assez voisins. Mais avec le graphe, les résultats sont moins nombreux. De plus avec cette méthode, une condition forte entre en considération : un mot ne peut appartenir à plus d'un champ lexical d'un autre mot, mais cette condition nous semble trop forte en effet si nous recherchons les collocations de 007, deux champs lexicaux existent celui de *James Bond* et celui des *casinos* or dans ces deux thèmes nous aimerions avoir *smoking* ou bien *jeu*.

De plus, la recherche par la méthode du graphe a une complexité plus élevée que la méthode de fusion des listes. C'est pourquoi nous avons décidé de garder uniquement la méthode de fusion.

Comme nous pouvons le voir, la pondération utilisé, n'est pas très parlante. Nous avons donc décidé, qu'une fois les listes établies, nous devons les repondérer de manière relative. Voici un exemple de re-pondération :

Avant re-pondération	Après re-pondération
noir (0.2991453)	noir (0.309)
elfe (0.15384616)	t elfe (0.159)
armure (0.07692308)	armure (0.079)
plates (0.06410257)	plates (0.066)
complète (0.055555556)	complète (0.057)
fléau (0.047008548)	fléau (0.048)
bouclier (0.034188036)	bouclier (0.035)
carreau (0.025641026)	carreau (0.026)
Epée (0.025641026)	Epée (0.026)
longue (0.021367522)	longue (0.022)
mailles (0.021367522)	mailles (0.022)
cotte (0.021367522)	cotte (0.022)
arbalète (0.012820513)	arbalète (0.013)
vitesse (0.012820513)	vitesse (0.013)
étourdissement (0.012820513)	étourdissement (0.013)
container (0.012820513)	container (0.013)
Ecu (0.008547009)	Ecu (0.0080)
fléchette (0.008547009)	fléchette (0.0080)
rondache (0.008547009)	rondache (0.0080)
XP (0.0042735045)	XP (0.0040)

Nous voyons qu'apparaissent, dans nos listes de cooccurrences résultats, des bouts de collocations. En effet pour *quimboiseur*, nous avons *Walt* et *Disney* et nous préfererions avoir uniquement *Walt Disney*.

Nous allons donc voir comment fusionner ces termes pour obtenir leur collocation.

Notre algorithme est fait pour tourner continuellement, en effet il faut initialiser le programme avec une liste de mots à chercher, puis quand les cooccurrences de ces mots sont trouvées, nous partons à la recherche des mots des listes de cooccurrences et ainsi de suite. Durant le déroulement de ce processus, les mots sont recherchés plusieurs fois cela permet d'affiner leurs champs sémantiques (en trouvant par exemple de nouvelles collocations en tant que cooccurrence) et d'en trouver de nouveaux si le mot évolue.

Chapitre 4

Algorithmes généraux et architecture

Lors de nos recherches de collocations et de cooccurrences, nous trouvons des mots inconnus, ainsi que des mots faisant partis de collocations inconnues.

Nous avons donc mis en place un algorithme destiné à tourner continuellement. Cet algorithme permet de trouver les signatures lexicales de nouveaux termes, et d'affiner celles de mots déjà connus.

Algorithm 5: Algorithme général

Données: Liste de termes inconnus

Résultat: une signature lexicale pour chaque terme

pour chaque *terme de la liste* **faire**

 Recuperation d'un corpus pour le terme

 Recherche des collocations du terme dans notre corpus

pour chaque *collocation* **faire**

si *collocation inconnue* **alors**

 ajout(Liste, collocation)

fin

fin

 Recherche des cooccurrences du terme dans notre corpus

pour chaque *cooccurrences* **faire**

si *cooccurrence inconnue* **alors**

 ajout(Liste, cooccurrence)

fin

fin

 ajout(Liste, terme)

fin

La liste de termes doit être implémentée sous forme d'un arbre de priorité, ainsi un mot demandé fréquemment sera cherché prochainement. De plus, à tout moment, nous devons pouvoir forcer la recherche immédiate d'un mot.

Grâce à cet algorithme, au fur et à mesure de nos recherches, notre dictionnaire augmente, et le nombre de collocation de même.

Ainsi si nous recherchons *livre de la jungle*, nous allons trouver certainement *Disney* et *Walt*, puis lors de la recherche des collocations de *Disney*, nous allons trouver *Walt Disney*, ainsi quand nous re-chercherons les cooccurrences de *livre de la jungle*, les deux termes *Walt* et *Disney*, appartenants à la liste de cooccurrences, seront fusionnés.

Chapitre 5

Résultats complémentaires

5.1 Amour

Collocations de *Amour*

Collocation	Poids	Collocations	Poids
amour de dieu	34000	France d'amour	3200
humour amour	13600	amour conjugal	2540
Mots d'amour	12100	Ivre d'amour	2520
mots d'amour	12100	Maladie d'amour	2480
Amour fou	10700	nouvel amour	2410
Déclaration d'amour	8600	Mariage amour	2230
déclaration d'amour	8600	question d'amour	2220
Amour impossible	7340	mot amour	2180
chanson d'amour	6180	Salon de l'amour	1900
chansons d'amour	5460	civilisation de l'amour	1710
chagrin d'amour	5030	Philtre d'amour	1560
Poèmes d'amour	4980	messages d'amour	1310
amour éternel	4970	Amour d'enfance	1290
Declarations d'amour	4130	puissance de l'amour	1130
Déclarations d'amour	4130	Histoire de l'amour	938
declaration-amour	4020	amour fidèle	903
Amour-humour	3260	Fol amour	803
Amour vrai	3230	Amour anniversaire	768

Cooccurrences de *Amour*

Cooccurrences	Poids	Cooccurrences	Poids	Cooccurrences	Poids
amour	0.153	bureau	0.13	larmes	0.129
dire	0.127	déclaration	0.089	coeur	0.129
monde	0.087	kamasutra	0.065	beauté	0.079
xxx	0.063	lit	0.065	vie	0.077
don	0.061	patron	0.065	citation	0.064
bonheur	0.038	année	0.065	tête	0.064
éternel	0.038	bonheur	0.06	aime	0.064
rêve	0.038	amour	0.044	seul	0.064
rencontres	0.03	pour toujours	0.042	lune	0.064
sentiment	0.03	amoureux	0.042	amour	0.064
joie	0.03	couple	0.042	virtuel	0.064
âme	0.03	compte	0.032	sentiment	0.064
tomber	0.03	tête	0.032	amitié	0.064
amoureux	0.03	interlocuteur	0.032		
ami	0.03	joli	0.032		
amies	0.03	acteur	0.032		
désir	0.03	belles	0.032		
belle	0.03	cuisse	0.032		
amitié	0.03	seul	0.027		
pour toujours	0.03				
vie	0.026				

5.2 Chaos

Collocations de *Chaos*

Collocations	Poids
théorie du chaos	3550
Théorie du chaos	3550
Theorie du chaos	3550
Chaos legion	3040
Kung-fu chaos	2680
Kung fu chaos	2680
Urban chaos	2470
Chaos in	1210
Chaos total	979
Chaos 2002	777
chaos déterministe	773

Cooccurrences de *Chaos*

Cooccurrences	Poids
structures	0.132
bifurcation	0.106
rythme	0.09
système	0.051
dynamique	0.04
complexité	0.038
analyse	0.035
Previous	0.035
noeud	0.026
session	0.026
phase	0.025
ordre	0.02
conservatifs	0.02
unité	0.02
méthode	0.02
chaos	0.02
sciences	0.012
programme	0.012
contenant	0.012
déterminisme	0.01
icône	0.01
cas	0.01
Daniel	0.01
Etude	0.01
perturbation	0.01
physique	0.01
mécanique	0.01
Jacques	0.01
HENRARD	0.01
LEMAÎTRE	0.01
azur	0.01
université	0.01
mathématique	0.01
linéaire	0.01
calcul	0.01

Cooccurrences	Poids
nature	0.199
fractales	0.199
France	0.199
documentaires	0.199
maths	0.199

Cooccurrences	Poids
dimension	0.222
espace	0.111
chaos	0.055
Previous	0.055
attracteur	0.055
transition	0.055
scénario	0.055
phase	0.055
cascade	0.055
sous-harmonique	0.055
intermittence	0.055
quasi-périodicité	0.055
oscillation	0.055
électronique	0.055

5.3 Chirurgie esthétique

Collocations de *Chirurgie esthétique*

aucune collocation trouvée

Cooccurrences de *Chirurgie esthétique*

Cooccurrences	Poids	Cooccurrences	Poids	Cooccurrences	Poids
articles	0.08	chirurgie	0.086	chirurgie esthétique	0.09
article	0.08	cicatrice	0.043	technique	0.06
chirurgie esthétique	0.08	cutané	0.043	reconstructive	0.06
différence	0.04	chirurgie esthétique	0.043	brûlure	0.03
clic	0.04	médical	0.021	microchirurgie	0.03
disponible	0.04	déontologie	0.021	vasculaire	0.03
lire	0.04	Champel	0.021	anomalie	0.03
préopératoire	0.04	clinique	0.021	cutané	0.03
consultation	0.04	plastie	0.021	tumeur	0.03
plastique	0.04	mammaire	0.021	lambeau	0.03
reconstructrice	0.04	augmentation	0.021	greffes	0.03
douleur	0.04	réduction	0.021	plastique	0.03
vaincue	0.04	Lipo-aspiration	0.021	année	0.03
cicatrisation	0.04	lipo-sculpture	0.021	programme	0.03
étranger	0.04	ultra-sons	0.021	nez	0.03
prothèse	0.04	rhinoplastie	0.021	paupière	0.03
mammaire	0.04	vieillesse	0.021	reconstructive	0.03
mammographie	0.04	lifting	0.021	oreille	0.03
lipoaspiration	0.04	rides	0.021	troubles	0.03
classique	0.04	filling	0.021	nerfs	0.03
ultra	0.04	correction	0.021	cranio-faciaux	0.03
		dermabrasion	0.021	malformation	0.03
		calvitie	0.021	congénital	0.03
		imperfection	0.021	cranio-faciales	0.03
		silhouette	0.021	augmentation	0.03
		satisfait	0.021	réduction	0.03
		peau	0.021	reconstruction	0.03
		superficiel	0.021	mammaire	0.03
		perspective	0.021	main	0.03
		diététique	0.021		

5.4 Dragon

Collocations de *Dragon*

Collocations	Poids
Dragon ball	49500
dragon ball	49500
Dragon rouge	10200
dragon rouge	10200
Dragon ball gt	6040
Black dragon	5540
Dragon noir	4120
Dragon quest	3410
Red dragon	2950
Hidden dragon	2560
Coeur de dragon	1850
Antre du dragon	1590
Dragon naturallyspeaking	1410
dragon dvd	1280
Saga dragon ball	1010
saga dragon ball	1010
chevalier du dragon	982
dragon balls	974
Sang de dragon	862
Oeil du dragon	827
Manga dragon	753

Cooccurrences de *Dragon*

Cooccurrences	Poids	Cooccurrences	Poids	Cooccurrences	Poids
vrai	0.125	ball	0.159	tête	0.13
Komodo	0.125	Sangoku	0.079	travail	0.086
lézard	0.125	Bulma	0.039	énorme	0.043
collerette	0.125	nommé	0.039	ciel	0.043
volant	0.125	filles	0.039	signe	0.043
dragon	0.125	rencontre	0.039	second	0.043
moyen	0.125	Dragon	0.039	enfantement	0.043
age	0.125	début	0.039	douleurs	0.043
		boule	0.039	enceinte	0.043
		grand-père	0.039	étoiles	0.043
		dispersés	0.039	pied	0.043
		monde	0.039	lune	0.043
		vouloir	0.039	enveloppe	0.043
		seul	0.039	rouge	0.043
		voeu	0.039	feu	0.043
		sacré	0.039	cornes	0.043
		DBZ	0.039	diadème	0.043
		dimension	0.039	arrêt	0.043
				enfant	0.043

5.5 Geek

Collocations de *Geek*

aucune collocation trouvée

Cooccurrences de *Geek*

Cooccurrences	Poids	Cooccurrences	Poids
Yahoo	0.142	note	0.15
catégorie	0.095	intéressant	0.1
copine	0.047	vrai	0.05
nerds	0.047	superglobaux	0.05
Geeks	0.047	tableau	0.05
communauté	0.047	patricia	0.05
cultures	0.047	utile	0.05
société	0.047	session	0.05
contenu	0.047	tuto	0.05
avancée	0.047	Unix	0.05
conseil	0.047	year	0.05
première	0.047	pweter	0.05
rencontre	0.047	articles	0.05
alimentation	0.047	MySQL	0.05
romantisme	0.047		
barre	0.047		
outil	0.047		
compagnon	0.047		

5.6 Google

Collocations de *Google*

Collocations	Poids
Google usenet	20400.0
google france	18000.0
Référencement google	5170.0
referencement google	5170.0
technologie google	2310.0
Positionnement google	1200.0
positionnement google	1200.0

Cooccurrences de *Google*

Cooccurrences	Poids
référencement	0.114
échange	0.081
positionnement	0.081
pagerank	0.081
partenariat	0.065
popularité	0.065
Référencement	0.049
dynamique	0.032
outil	0.032
Pagerank	0.032
rank	0.032
promotion	0.032
indice	0.032
booster	0.032
référéncé	0.016
populaire	0.016
échange-de-liens	0.016
Rank	0.016
Ranking	0.016
alltheweb	0.016
fast	0.016
exalead	0.016
altavista	0.016
echange-de-liens	0.016
optimisation	0.016
referencement	0.016
définition	0.016
inktomi	0.016

Cooccurrences	Poids
traceroute	0.238
Google	0.214
java	0.214
cyrille	0.071
morvan	0.071
free	0.071
yahoo	0.047
antibot	0.023
nombre	0.023
resultats	0.023

Cooccurrences	Poids
ligne	0.153
adresse	0.153
Google	0.076
rendez-vous	0.076
sommaire	0.076
navigation	0.076
Netgroupe	0.076
astuce	0.076
outil	0.076
linguistique	0.076
langue	0.076

5.7 Mythologie

Collocations de *Mythologie*

Collocations	Poids
Mythologie égyptienne	2480
mythologie égyptienne	2480
Mythologie égyptienne	2480
Mythologie nordique	2390
mythologie nordique	2390
Mythologie romaine	2170
mythologie romaine	2170
Mythologie greco-romaine	1260
Mythologie gréco-romaine	1260
Mythologie classique	1110
mythologie classique	1110
Mythologie celtique	993
Héros de la mythologie	867
Dictionnaire de la mythologie	794

Cooccurrences de *Mythologie*

Cooccurrences	Poids
mythologie	0.114
musée	0.102
nom	0.096
activité	0.081
terre	0.081
grecs	0.054
illustration	0.049
grecque	0.049
dieux	0.049
monde	0.049
planète	0.04
connue	0.04
Troie	0.032
guerre	0.032
couleur	0.024
consacré	0.016
romaine	0.016
héros	0.016
légendes	0.016
déesse	0.016
grec	0.016

Cooccurrences	Poids
nordique	0.236
légendes	0.218
mythologie	0.044
antique	0.044
encyclopédie	0.044
sculpture	0.044
peinture	0.044
grecque	0.044
images	0.044
introduction	0.044
moderne	0.044
dictionnaire	0.044
bonjour	0.033
dieux	0.033
héros	0.033

Cooccurrences	Poids
légendes	0.134
dieux	0.134
déesse	0.134
grecque	0.155
vivants	0.097
héros	0.067
grecques	0.067
nordique	0.067
créature	0.067
cerbère	0.067
cyclope	0.067

5.8 Napoléon

Collocations de *Napoléon*

Collocations	Poids
Napoléon iii	29200
Napoléon bonaparte	16800
Napoléon ier	8280
Napoléon 1er	5780
Empereur napoléon	4300
Napoléon empereur	2250
Joseph napoléon	2200
Hotel napoléon	1770
Frans napoléon	1480
Fondation napoléon	1360
époque napoléon	1120
fihs de napoléon	850

Cooccurrences de *Napoléon*

Cooccurrences	Poids	Cooccurrences	Poids	Cooccurrences	Poids
histoire	0.052	Espagne	0.015	vie	0.0080
Français	0.048	ligne	0.015	Empereur	0.0080
photo	0.046	for	0.015	table	0.0080
époque	0.035	edu	0.014	second	0.0080
Bonaparte	0.035	presse	0.014	année	0.0080
Napoleon	0.035	lettre	0.014	Victor	0.0070
tableau	0.029	maréchal	0.014	Ajaccio	0.0070
NAPOLEON	0.028	décembre	0.014	Joseph	0.0070
BONAPARTE	0.028	impérial	0.014	nom	0.0050
guerre	0.026	Sainte-Hélène	0.014	aigle	0.0050
France	0.026	roi	0.013	livre	0.0050
Europe	0.026	empereur	0.013	porte	0.0050
route	0.025	campagne	0.013	sacre	0.0050
invalides	0.022	Joséphine	0.011	1er	0.0040
français	0.022	Autriche	0.01	personnage	0.0040
révolution	0.021	Charles	0.01	Empereur	0.0040
franc	0.021	ouvrages	0.0090	peinture	0.0040
in	0.018	francs	0.0090	Italie	0.0040
grand	0.018	Ier	0.0080	napoléonien	0.0030
musée	0.017	empire	0.0080	batailles	0.0030
Louis	0.016	Marie-Louise	0.0080	corse	0.0030

5.9 Quidditch

Collocations de *Quidditch*

Collocations	Poids
match de quidditch	926

Cooccurrences de *Quidditch*

Cooccurrences	Poids
joue	0.142
sorcier	0.071
national	0.071
sport	0.071
Quidditch	0.071
balais	0.071
volant	0.071
batteur	0.071
cognars	0.071
poursuiveur	0.071
souafle	0.071
attrapeur	0.071
vif	0.071

Cooccurrences	Poids
Potter	0.058
Harry	0.058
hibou	0.058
cartes	0.058
astuce	0.058
PS-one	0.058
acteur	0.058
images	0.058
film	0.058
magie	0.058
truc	0.058
dessin	0.058
Pré-au-Lard	0.058
traduction	0.058
sortilège	0.058
Hist-Poudlard	0.058
rouge	0.058

Cooccurrences	Poids
gallions	0.222
nimbus	0.088
balai	0.066
éclair	0.044
acessoire	0.022
magasin	0.022
Quidditch	0.022
Assecoire	0.022
Saver	0.022
Screen	0.022
Scene	0.022
Lake	0.022
Free	0.022
Melanies-hogwarts	0.022
étoile	0.022
fillante	0.022
brossdur	0.022
de feu	0.022
boule	0.022
entretien	0.022
kit	0.022
gallion	0.022
feu	0.022
Equipement	0.022
genouilliere	0.022
gants	0.022
battes	0.022
ect	0.022
formulaire	0.022
nom	0.022

5.10 Tigre

Collocations de *Tigre*

Collocations	Poids
Tigre de tasmanie	1360
tigre de tasmanie	1360
tigre noir	1150
oeil du tigre	1120
Oeil du tigre	1120
Brigades du tigre	1110
brigades du tigre	1110
Tigre blanc	1080
tigre blanc	1080
Oeil de tigre	1080
tigre de papier	895
tigre du bengale	837
Tigre du bengale	837

Cooccurrences de *Tigre*

Cooccurrences	Poids	Cooccurrences	Poids
dents	0.053	Barjavel	0.013
faim	0.053	collection	0.013
Tasmanie	0.041	longueur	0.012
bébé	0.038	mètres	0.012
petit	0.033	Sibérie	0.012
fourrure	0.032	nom	0.012
original	0.032	eau	0.012
forêt	0.031	amoureux	0.012
tête	0.028	trouvé	0.011
Editeur	0.026	animaux	0.01
roman	0.026	tigre	0.0080
présente	0.026	dangereux	0.0080
blanc	0.025	mère	0.0080
queue	0.016	espèces	0.0080
félin	0.016	menacé	0.0080
année	0.016	tigres	0.0080
porte	0.016	Asie	0.0080
tigresse	0.016	peau	0.0080
lion	0.016	livres	0.0070
griffes	0.015	jeunesse	0.0070
noir	0.015	mesure	0.0070
jungle	0.015	seul	0.0070
vie	0.014	Bengale	0.0070
mâle	0.014	mammifère	0.0060
femelle	0.014	animal	0.0060
longévit��	0.014	pelage	0.0060
ray��	0.013	noires	0.0060
commentaire	0.013	taille	0.0060
		bleu	0.0060
		chasse	0.0050
		litt��rature	0.0030

Bibliographie

- [1] Olivier Ferret. Filtrage thématique d'un réseau de collocations, 2003.
- [2] Jean Véronis. Cartographie lexicale pour la recherche d'information, 2003.