République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université Mouloud Mammeri de Tizi-Ouzou

Faculté de : Génie électrique et d'informatique
Département : Informatique

Mémoire de Master

Spécialité : Systèmes Informatiques

Sujet : Prise en compte des liens dans la recherche d'information dans les documents XML

Réalisé par :
Mr. SADI Samy

Proposé et dirigé par :
Mme. AMIROUCHE Fatiha

Soutenu le 1er juillet 2012 devant le jury composé de:

Mme. AMIROUCHE Fatiha	Directrice de mémoire
Mme. FELLAG Samia	Présidente du Jury
Mr. HAMMACHE Arezki	Membre du Jury
Mr. SADOU Samir	Membre du Jury

Dédicaces

À mes très chers parents.

À mes deux sœurs que j'adore.

À mes amis.

Remerciements

Je tiens tout d'abord à témoigner ma profonde gratitude et mes remerciements les plus sincères à Mme. Fatiha Amirouche pour avoir dirigé mon travail, pour son soutien et pour tout le temps qu'elle a consacré au bon déroulement de ce travail.

Mes remerciements vont aussi à Mme. Samia Fellag pour l'aide qu'elle m'a apporté tout au long de ce travail.

Enfin, je remercie également les membres du jury qui ont bien voulu juger mon travail.

Résumé

Ce mémoire s'inscrit dans le cadre de la recherche d'information structurée. Plus précisément, nous nous intéressons à la prise en compte des liens dans la recherche d'information structurée. Ceci permet non seulement de sélectionner des documents pertinents inaccessibles par une recherche classique, mais aussi de réordonner les résultats retournés par une première recherche afin d'améliorer la qualité de la réponse apportée à l'utilisateur.

Dans ce mémoire, nous présentons un modèle de RI pour la prise en compte des liens qui propage les scores des documents vers leurs voisins (désignés par des liens sortants et entrants), tout en tenant compte du texte ancre des liens et de la balise titre des documents. Les évaluations de notre modèle sur les collections INEX 2006 et INEX 2009 ont montré des améliorations notables de la qualité de la réponse apportée à l'utilisateur. De plus, notre modèle a obtenu les meilleurs résultats parmi les modèles pour la prise en compte des liens qui ont été testés (dont HITS, SALSA et l'activation propagée).

Mots clés: recherche d'information, recherche d'information structurée, XML, prise en compte des liens, texte ancre du lien, balise titre, propagation de pertinence, INEX.

Abstract

This thesis is made within the context of information retrieval in structured documents. Specifically, we will consider link analysis techniques. This does not only allow to select relevant documents that are inaccessible by a traditional research, but also allows to reorder the results returned by a first research to improve the quality of the response returned to the user.

In this thesis, we present a new IR model for link analysis that propagates the scores of documents to their neighbors (designated by the outgoing and the incoming links), while in the meantime, the anchor text of links and the title tag of these documents are also used. Evaluations have shown significant improvements in the quality of retrieval results using our model in both INEX 2006 and INEX 2009 collections. Furthermore, our model showed the best results among the tested models for link analysis (including HITS, SALSA and spreading activation).

Key-words: information retrieval, information retrieval in structured documents, XML, link analysis, link anchor text, title tag, relevance propagation, INEX.

Table des matières

Table des tableaux et des figures

Introduction générale

1 Contexte du travail

Notre travail s'inscrit dans le cadre de la recherche d'information (RI). La RI est l'ensemble des méthodes et mécanismes qui permettent la création et l'utilisation d'une base d'information. Une base d'information est un système documentaire permettant d'exploiter une collection de documents.

La mise en œuvre de la recherche d'information est effectuée grâce à un système de recherche d'information (SRI). C'est un ensemble logiciel qui possède trois fonctions fondamentales: représenter le contenu des documents grâce à un processus d'indexation, représenter le besoin en informations de l'utilisateur et comparer ces deux représentations grâce à un processus d'appariement. À la suite de cette comparaison, le SRI retourne à l'utilisateur une liste de documents qui est généralement ordonnée de façon à ce que les documents les plus susceptibles d'intéresser l'utilisateur apparaissent en première position.

Aux fondements de la RI, vers la moitié du vingtième siècle, le processus de recherche consistait principalement à exploiter le contenu textuel des documents. Cependant, après l'apparition du SGML, la naissance du HTML et du WEB et avec la généralisation de l'XML, la RI a vite évolué pour tenir compte de la structure en plus de l'information textuelle des documents. L'information qu'apporte la structure a grandement amélioré la qualité de réponse des SRI. Elle permet de mieux définir la granularité de l'information, de définir de nouveaux langages de requêtes pour mieux exprimer le besoin en informations de l'utilisateur et aussi de lier les documents entre eux grâce à des liens de citations et de références. Dans le cadre de notre travail, nous nous focalisons sur ce dernier élément, à savoir : la prise en compte des liens dans la recherche d'information dans les documents XML.

Plusieurs travaux ont été entrepris pour tenir compte de l'information qu'apportent les liens dans la recherche d'information. Ces travaux proposent d'améliorer la réponse apportée à l'utilisateur en sélectionnant des documents qui n'auraient pas pu être retrouvés par une recherche classique (sans prise en compte des liens) et en réordonnant les documents restitués par une recherche classique afin de présenter en premier les documents qui répondent le mieux à la requête utilisateur.

2 Problématique

Le développement du web et l'accroissement du volume de données stockées sous format numérique posent de nouveaux défis dans le domaine de la RI. En effet, il faut prévoir de nouveaux systèmes capables d'indexer et d'effectuer des recherches dans de tels volumes d'information. De plus, on doit garantir à l'utilisateur de pouvoir retrouver l'information qu'il recherche avec un minimum d'efforts.

Pour ce faire, obtenir une liste, la plus exhaustive possible, des sources répondant à une requête est nécessaire, mais insuffisant dès lors que le nombre de réponses dépasse la centaine. Il est alors nécessaire de pouvoir classer ces réponses de façon à présenter à l'utilisateur en premier lieu les réponses qui sont les plus susceptibles de l'intéresser.

Le classement dans la RI classique se base uniquement sur le score de pertinence de chaque réponse par rapport à la requête. En revanche dans la RI structurée, d'autres facteurs peuvent être pris en compte, en particulier les liens de citation et de référence.

La problématique qui se pose alors est: « *comment exploiter l'information des liens pour améliorer la réponse apportée à l'utilisateur?* ».

Plusieurs travaux ont été entrepris dans ce sens et ont montré leur intérêt. Ces travaux exploitent pour la plupart les liens entrants et sortants pour réordonner les résultats obtenus suite à une requête. Dans ce mémoire, nous proposons une nouvelle approche pour répondre à cette problématique, qui consiste à considérer en plus des liens entrants et sortants, le texte ancre des liens et le titre des documents.

3 Contribution

Notre contribution dans le cadre de la recherche d'information se situe à plusieurs niveaux:

1. Réalisation d'un état de l'art sur la recherche d'information classique et structurée, on y présente notamment les travaux relatifs à la prise en compte des liens dans la recherche d'information;
2. Proposition d'un nouveau modèle pour la prise en compte des liens dans la recherche d'information qui tient compte, en plus des liens entrants et sortants, du texte ancre des liens et de la balise titre des documents;
3. Implémentation de ce modèle sur la plateforme de RI Terrier 3.5;
4. Évaluation de ce modèle sur les collections INEX 2006 et INEX 2009, et comparaison des résultats obtenus avec ceux obtenus en utilisant d'autres modèles de la RI pour la prise en compte des liens.

4 Organisation du mémoire

Ce mémoire est organisé en quatre chapitres et deux annexes.

Dans le premier chapitre, nous introduisons la recherche d'information et le processus de recherche d'information. On y présente notamment les différents modèles de la RI et les principaux critères d'évaluation d'un SRI (système de recherche d'information).

Dans le deuxième chapitre, nous présentons la recherche d'information structurée. Nous commençons par présenter le format XML qui permet de représenter l'information structurelle conjointement avec l'information textuelle dans un même document. Ensuite, nous identifions les problématiques qu'engendre la prise en compte de la structure dans le processus de RI. Les solutions à ces problématiques, qui consistent à modifier l'indexation et à définir de nouveaux langages de requêtes et modèles de RI, sont également décrites dans ce chapitre. Pour finir, nous présentons les compagnes d'évaluation INEX qui sont entreprises afin d'évaluer les SRIS (système de recherche d'information structurée).

Dans le troisième chapitre, nous présentons notre modèle pour la prise en compte des liens dans la recherche d'information. On décrit la procédure d'indexation et la fonction de correspondance qui tient compte des titres des documents, des liens et du texte ancre des liens.

Dans le quatrième chapitre, on présente notre implémentation et les résultats de l'évaluation de notre modèle sur les collections INEX 2006 et INEX 2009. En outre, nous comparons notre modèle avec les modèles de RI les plus populaires pour la prise en compte des liens.

Pour finir, nous présentons dans la première annexe les technologies de la famille XML et dans la deuxième, la plateforme de RI Terrier 3.5.

Chapitre 1: État de l'art de la recherche d'information

1 Introduction

Le stockage et l'accès à l'information a toujours suscité un grand intérêt pour les informaticiens, notamment avec la croissance considérable du volume documentaire stocké sous format numérique du fait de l'avènement explosif de l'internet et d'autres services de l'information.

La recherche d'information (RI) [82] [38] [111], est l'ensemble des techniques qui permettent de gérer ces documents. Ceci implique l'acquisition, l'organisation, la recherche et la sélection de l'information.

Un système de recherche d'information (SRI) [85], est un ensemble d'outils (programmes informatiques) permettant de sélectionner dans une collection de documents ceux qui sont susceptibles de répondre au besoin en informations d'un utilisateur. Ce besoin peut être exprimé de manière active sous forme d'une requête, on parle alors de recherche active, ou bien de manière passive, en se basant sur le profil utilisateur, on parle alors d'un *système de filtrage d'information* [100].

Dans le cadre de ce mémoire, nous nous intéressons à la recherche active de l'information. Ce chapitre a pour but de présenter les principaux concepts de la RI, les principaux modèles de recherche ainsi que les approches d'évaluation des SRI. Nous commençons tout d'abord par donner quelques définitions, puis nous décrivons les étapes d'un processus de RI. Nous passons ensuite en revue les différents modèles de recherche. Enfin, nous présentons les plus importants critères d'évaluation d'un SRI.

2 Concepts de base de la recherche d'information

Pour interroger[1] une base documentaire, l'utilisateur exprime son besoin en informations sous forme d'une requête qu'il transmet au SRI. Le rôle du SRI, est de sélectionner et de retourner à l'utilisateur les documents jugés pertinents à la requête parmi tous les documents de la base documentaire.

La problématique majeure du SRI, est de retrouver dans un temps raisonnable, quelques dizaines, centaines ou milliers de documents pertinents parmi des collections qui peuvent contenir des millions de documents. Pour ce faire, la requête, ainsi que tous les documents de la collection sont au préalable représentés sous une forme intermédiaire, propre au SRI, définie par le modèle de représentation. La façon dont ces représentations sont ensuite utilisées pour retrouver les documents pertinents est, quant à elle, définie par le modèle de recherche.

Dans ce qui suit, nous définissons les éléments clés de la recherche d'information que nous venons d'évoquer.

2.1 Le document

Le terme « *document* », du latin *« Documentum »*, a connu plusieurs définitions au cours de l'histoire. Le sens principal de ce terme a été « *enseignement* » jusqu'au dix-huitième siècle où il devint « *preuve ».* Une autre définition lui a été attribuée au dix-neuvième puis au vingtième siècle[2] : *« Document : Toute base de connaissance, fixée matériellement, susceptible d'être utilisée*

[1] Effectuer une recherche.

[2] L'Institut International de Coopération Intellectuelle (de la Société des Nations) et l'Union Française des Organismes de Documentation, 1937.

pour consultation, étude ou preuve. Exemples : manuscrits, imprimés, représentations graphiques ou figurées, objets de collections, etc. ». De façon générale, le document est une trace d'activité humaine véhiculant une information pouvant être lue et interprétée par une personne non nécessairement à l'origine de cette trace.

Avec l'avènement de l'informatique, le document a quitté son support matériel natif pour une représentation binaire dans les mémoires des ordinateurs. Suite de quoi, les SRI sont devenus une nécessité pour gérer ces documents.

Dans le cadre d'un SRI, le document représente l'élément objectif [87]. Il s'agit, dans l'optique de ce mémoire, d'un document textuel numérique qui peut être caractérisé selon trois vues:

- la vue représentation qui présente la mise en forme du texte (entêtes, paragraphes, alignements…);
- la vue logique qui présente la structure logique d'un document, elle porte des informations sur la structure;
- la vue contenu qui se focalise sur le sens ou la sémantique d'un document le plus souvent par un ensemble de mots.

En général, la vue contenu est celle qui représente le plus d'intérêt pour l'utilisateur [6].

2.2 La collection de documents

La collection de documents (ou base documentaire) est constituée d'un ensemble de documents. Elle contient la totalité des informations exploitables et accessibles par l'utilisateur.

La gestion de la collection de documents est assurée par le SRI. En général, et pour une gestion optimale, le SRI représente l'ensemble des documents de la collection sous une représentation intermédiaire.

2.3 La requête

La requête exprime le besoin en informations de l'utilisateur. C'est elle qui initie le processus de recherche d'information.

La requête doit contenir des éléments clés suffisamment discriminants afin d'exprimer au mieux le besoin en informations de l'utilisateur. La requête est écrite dans un langage d'interrogation compréhensible par le SRI parmi les langages d'interrogation, les plus utilisés sont:

- *le langage booléen*: la requête est un ensemble de mots clés (termes de recherche) séparés par des opérateurs logiques (*non*, *ou*, *et*);
- *le langage naturel*: la requête est exprimée avec un ensemble de mots clés dans un langage libre. *SMART* [83], *SPIRIT* [49], *OKAPI* [78] et *Mercure* [10] sont des systèmes interrogeables en langage naturel;
- *le langage graphique*: la requête est construite à partir d'une liste de mots clés connus. Pour ce faire, une vue d'ensemble représentant le contenu, notamment sémantique, des documents est offerte à l'utilisateur pour l'assister à formuler sa requête. *PROTEUS* [95] et *NEURODOC* [62] sont interrogeables en langage graphique.

2.4 La pertinence

La pertinence est une notion cruciale de la RI, à tel point que toutes les évaluations des SRI s'articulent autour de cette notion. Elle a été sujette à de nombreuses études au cours du développement de la RI et plusieurs définitions lui ont été attribuées:

- La correspondance entre un document et une requête ;
- Une mesure d'informativité du document à la requête;
- Le degré de relation (chevauchement, relativité, …) entre le document et la requête;
- Le degré de la surprise qu'apporte un document, qui a un rapport avec le besoin de l'utilisateur.

Ces définitions restent vagues et ne tiennent pas compte de certains aspects de la pertinence. Par exemple, le jugement utilisateur, qui est un facteur important pour définir la pertinence, n'est pas présent dans ces définitions. Un document peut être jugé pertinent par un utilisateur et non pertinent par un autre utilisateur, et cela pour une même requête. De ce fait, la pertinence n'est pas une relation isolée entre un document et une requête.

En pratique, il est souvent difficile de connaitre le jugement utilisateur. Pour cette raison, on a introduit les notions de *pertinence utilisateur* et de *pertinence système*. La pertinence utilisateur [42] [87] [70] est une notion subjective qui dépend du jugement utilisateur sur les documents restitués pour sa requête. La pertinence système [17], elle, est déterministe, objective, ne tenant pas compte du jugement utilisateur. Elle est, contrairement à la pertinence utilisateur, plus facilement évaluée par le SRI.

2.5 Le modèle de représentation

Avant d'être utilisés par le SRI, les documents et les requêtes sont traduits depuis une forme brute vers une forme structurée plus facilement exploitable. Le modèle de représentation définit comment sont représentés ces éléments une fois traduits.

2.6 Le modèle de recherche d'information

C'est le modèle noyau d'un SRI. Il comprend la fonction de décision fondamentale qui permet d'associer à une requête, l'ensemble des documents pertinents à restituer [99].

3 Le processus de recherche d'information

Le processus de recherche d'information est mis en œuvre par un SRI afin de répondre à une requête utilisateur qui se décompose, suivant le modèle de recherche implémenté par le SRI, en deux ou trois sous-processus: le processus d'indexation, le processus d'appariement et éventuellement le processus de reformulation de la requête.

La figure 1 illustre le déroulement d'un processus de recherche d'information.

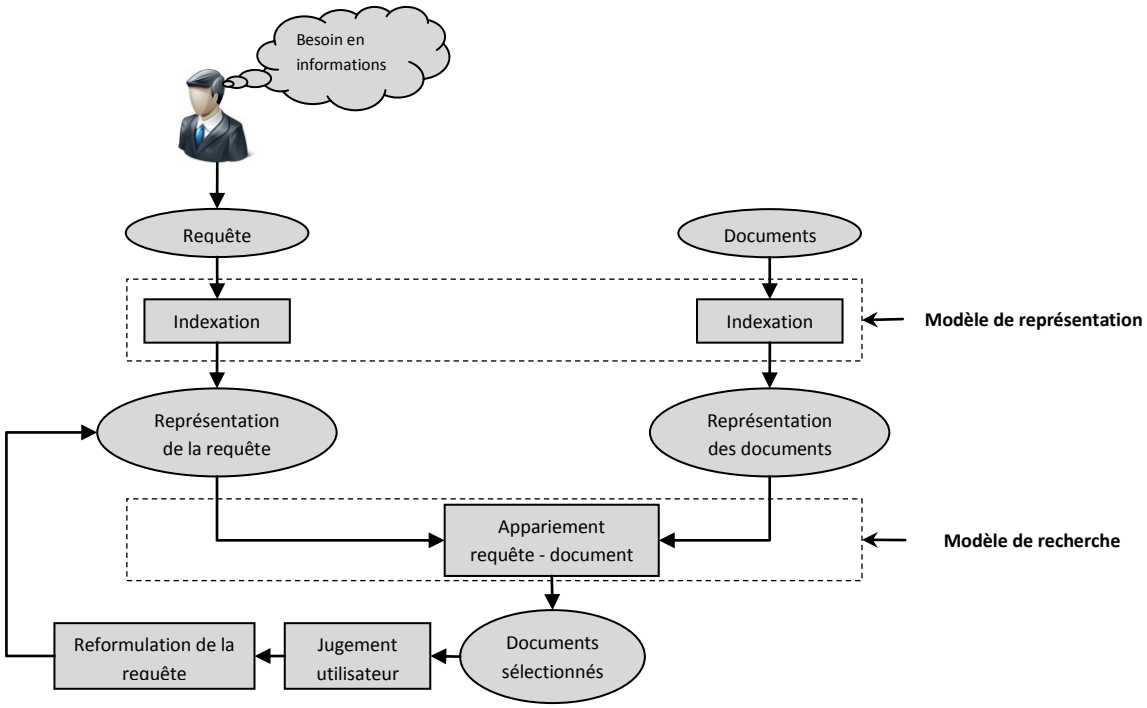


Figure 1: Processus de recherche d'information

Dans ce qui suit, nous présentons chacun de ces processus.

3.1 Le processus d'indexation

Le processus d'indexation consiste à créer une représentation interne de la requête utilisateur et de tous les documents de la collection de façon à refléter aussi fidèlement que possible leur contenu. Cette représentation consiste en un ensemble de descripteurs [83] [103], composés de mots et de groupes de mots souvent associés à des poids, qui forment *le langage d'indexation*. Ces descripteurs (ou termes d'index) représentent le résultat du processus d'indexation et sont rangés dans une structure appelée *dictionnaire* ou encore *index*.

De la qualité de l'indexation dépend, en partie, la qualité de la réponse du SRI. Ainsi, pour avoir une réponse satisfaisante du système, l'indexation doit être efficace. Plusieurs techniques d'indexation peuvent être utilisées: *l'indexation manuelle, l'indexation automatique* et *l'indexation semi-automatique*.

3.1.1 L'indexation manuelle

Lors de l'indexation manuelle, la liste des descripteurs pour chaque document est choisie par un documentaliste. Ce choix s'appuie souvent sur une liste de termes prédéfinie pour accroitre la qualité de l'indexation en prescrivant les termes les plus porteurs de sens tout en proscrivant les termes ambigus.

L'avantage de cette technique d'indexation est de pouvoir associer des termes à un document tout en tenant compte de son contexte, ce qui est difficilement réalisable par une indexation automatique.

Cependant, le fait de recourir à un opérateur humain rend l'indexation très subjective car elle devient dépendante du jugement et du domaine d'expertise de cet opérateur. Ce qui, en général, rend le système moins efficace car rien ne garantit que le documentaliste qui effectue l'indexation et l'utilisateur qui effectue une recherche utiliseront les mêmes termes pour décrire les mêmes

concepts (il a été démontré [35] que la probabilité que deux individus utilisent le même terme pour décrire un même concept est inférieur à *20%*). De plus, l'indexation manuelle est lente et coûteuse. Tout ceci implique que l'indexation manuelle est inadaptée, surtout pour les bases documentaires de grandes tailles.

3.1.2 L'indexation automatique

L'indexation automatique est la technique d'indexation la plus utilisée lors d'un processus de RI. Elle offre l'avantage d'être totalement automatisée et donc plus adaptée pour les bases documentaires de grandes tailles.

Le SRI choisit les termes les plus représentatifs dans chaque document et leur associe des poids. Ce traitement est réalisé en plusieurs étapes: *l'analyse lexicale*, *l'élimination des mots vides*, *la lemmatisation* et *la pondération*.

3.1.2.1 L'analyse lexicale

La première étape de l'indexation automatique consiste à extraire du document un certain nombre d'unités lexicales (termes). Une unité lexicale est définie comme étant une suite de caractères délimitée par des séparateurs (blancs, virgules …).

3.1.2.2 L'élimination des mots vides

Cette étape permet de réduire la taille de l'index, et de ce fait, le temps de réponse du système. Elle consiste à éliminer mots non significatifs, comme les articles, les prépositions et les conjonctions. Ce sont des mots qui ne sont pas discriminatoires lors d'une recherche, car ils ne traitent pas du sujet du document et sont présents en grand nombre dans les documents de la collection.

Deux techniques sont principalement utilisées pour l'élimination des mots vides:

- L'utilisation d'une liste prédéfinie de mots vides, aussi appelée *anti-dictionnaire* ou *stoplist* [29];
- L'utilisation de mesures statistiques sur la collection de documents pour déterminer les mots les plus fréquents ou les mots rares, qui sont « *probablement* » des mots vides.

3.1.2.3 La lemmatisation

La lemmatisation consiste à extraire la forme canonique (racine) de chacun des termes issus des étapes précédentes. L'objectif étant d'éliminer les variations morphologiques des mots (nombre, genre …).

L'algorithme de Porter [76] est l'algorithme le plus utilisé pour la lemmatisation des mots de la langue anglaise. En français, il n'existe pas d'algorithmes fiables pour la lemmatisation vu, par exemple, la complexité des formes prises par les verbes du troisième groupe (le verbe *être* peut prendre les formes: *est, sommes, fûmes* …). De ce fait, on a souvent recours à un dictionnaire.

3.1.2.4 La pondération

La pondération permet d'associer à chaque terme t_i un poids représentant son importance dans un document d_j. En général, ce poids est mesuré sur base de considérations statistiques en utilisant les deux mesures suivantes:

- *La fréquence locale (tf):* représente la fréquence du terme t_i dans le document d_j. La fréquence d'un terme représente le nombre d'occurrences de ce terme dans un document. Il existe plusieurs variantes pour calculer *tf*, notamment par l'application du

logarithme sur la fréquence du terme afin d'atténuer les écarts entre les fréquences (de l'ensemble des termes).

- *La fréquence globale (idf):* elle vise à donner une importance plus élevée aux termes qui apparaissent peu dans la collection de documents. Ainsi, un terme considéré comme rare dans la collection de documents aura un pouvoir discriminant plus élevé qu'un terme plus répandu [86]. Pour ce faire, on associe à chaque terme une mesure égale au logarithme de l'inverse de la proportion des documents qui contiennent le terme :

$$idf(t_i) = \log \frac{|D|}{|\{d_j : t_i \in d_j\}|}$$

Avec:

t_i : le terme dont on cherche la fréquence;

$|\{d_j : t_i \in d_j\}|$: le nombre de documents où le terme t_i apparait;

$|D|$: le nombre total de documents dans la collection;

$idf(t_i)$: la fréquence globale du terme t_i.

Ces deux mesures (*tf* et *idf*) sont combinées en *tf* × *idf* pour fournir le poids du terme t_i. Ce produit représente une bonne approximation du poids du terme, particulièrement si les documents de la collection sont de taille (longueur) homogène. Cependant, si la collection contient des documents très hétérogènes en termes de longueur, les documents les plus longs se voient favorisés (car plus un document est long, plus un terme a de chances d'apparaitre). Pour remédier à cela, un facteur de normalisation peut être utilisé [96] [79].

3.1.3 L'indexation semi-automatique

L'indexation semi-automatique est une alternative qui profite des deux techniques d'indexation précédentes. La liste des descripteurs est construite d'abord de façon automatique, ensuite un expert documentaliste rajoute, manuellement, les termes qu'il juge représentatifs pour chaque document.

3.2 Le processus d'appariement

Le processus d'appariement ou de comparaison consiste à mettre en correspondance la représentation de la requête avec les représentations des documents. Un degré de pertinence ou de similarité, noté *« RSV »* (*Retrieval Status Value*), entre la requête et chaque document de la collection est ainsi calculé. En se basant sur ce degré, les documents qui sont jugés similaires (par le SRI) à la requête sont, par la suite, retournés à l'utilisateur. Ceux-ci sont généralement classés par ordre de pertinence.

Plusieurs modèles existent pour réaliser l'appariement. Les principaux modèles sont décrits par la suite.

3.3 Le processus de reformulation de la requête

En plus des deux processus de base décrits précédemment, un SRI peut intégrer un processus de reformulation de la requête. Ce processus a pour but d'améliorer la performance du système en offrant un mécanisme de raffinement de la requête utilisateur. Les techniques utilisées durant ce processus se classent en deux catégories [8] : *les méthodes locales* et *les méthodes globales*.

3.3.1 Les méthodes locales

Elles s'appuient sur la technique dite de réinjection de pertinence (*relevance feedback*). En se basant sur la requête initiale, une liste de documents sera présentée à l'utilisateur, qui va ensuite

choisir les documents qu'il juge pertinents. La requête initiale sera ensuite enrichie avec les termes des documents ainsi choisis.

3.3.2 Les méthodes globales

La requête est réajustée en se basant sur des ressources externes telles que les bases lexicales, les thésaurus et les ontologies. Ainsi, des termes, non nécessairement présents dans la requête initiale, sont suggérés à l'utilisateur pour raffiner sa requête.

4 Modèles de recherche d'information

Le modèle de recherche d'information repose sur les représentations de la requête et des documents issues de l'indexation. Il permet de leur donner une interprétation afin de déterminer les documents qui sont similaires à la requête.

Au cours du développement de la RI, plusieurs modèles ont été proposés. Dans ce qui suit, nous présentons les principaux modèles utilisés.

4.1 Le modèle booléen

4.1.1 Le modèle booléen de base

Le modèle booléen [21] est le premier modèle utilisé en RI. Il se base sur la théorie des ensembles et l'algèbre de Boole. Les documents y sont représentés par l'ensemble des termes qu'ils contiennent. Et la requête y est représentée sous forme d'une équation logique contenant des termes reliés par des connecteurs logiques (*et, ou, non*).

Formellement:

- Un terme est une requête;
- Si **q** est une requête alors **NON q** est une requête;
- Si $\mathbf{q_1}$ et $\mathbf{q_2}$ sont des requêtes alors $\mathbf{q_1}$ **ET** $\mathbf{q_2}$ est une requête;
- Si $\mathbf{q_1}$ et $\mathbf{q_2}$ sont des requêtes alors $\mathbf{q_1}$ **OU** $\mathbf{q_2}$ est une requête.

La fonction d'appariement *RSV(d, q)* qui détermine si un document *d* répond à une requête *q* est calculée comme suit:

- RSV(d, t) = 1 si le terme $t \in d$, 0 sinon;
- RSV(d, **NON** q_i) = 1 - RSV(d, q_i);
- RSV(d, q_i **ET** q_j) = RSV(d, q_i) × RSV(d, q_j);
- RSV(d, q_i **OU** q_j) = RSV(d, q_i) + RSV(d, q_j) - RSV(d, q_i) × RSV(d, q_j);

q_i et q_j étant des requêtes quelconques.

C'est donc une fonction assez simple caractérisée par un mode d'appariement exact (*1* ou *0*). Ceci fait du modèle booléen un modèle facile à implémenter tout en étant tout à fait fonctionnel [30]. Par ailleurs, c'est aussi un modèle très expressif qui offre à l'utilisateur la possibilité d'effectuer une sélection exhaustive et non ambigüe.

Cependant, ce modèle nécessite de l'utilisateur la connaissance du langage booléen, qui est assez complexe. De plus, l'utilisateur doit pouvoir exprimer très précisément son besoin en informations, sans quoi, la réponse du système risque d'être insatisfaisante [20]. Par ailleurs, l'inconvénient majeur de ce modèle est de ne pas pouvoir classer les documents par ordre de

pertinence. Cet inconvénient se fait d'autant plus ressentir si la requête retourne un grand nombre de documents.

4.1.2 Le modèle booléen étendu

Le modèle booléen étendu [84] à été introduit pour compléter le modèle booléen de base en tenant compte du poids des termes issus de l'indexation.

La requête reste la même que celle du modèle booléen de base. Par contre, la fonction d'appariement utilise le poids des termes, qui sont préalablement normalisés pour être compris entre *0* et *1*, afin de calculer un score représentant le degré de similarité entre un document et une requête. Pour des requêtes à un ou deux termes, elle est définie comme suit:

- RSV(d, t_i) = a_i;
- RSV(d, **NON** t_i) = 1 - RSV(d, t_i);
- RSV(d, t_1 **ET** t_2) = $1 - \sqrt{\frac{(1-RSV(d,t_1))^2+(1-RSV(d,t_2))^2}{2}}$;
- RSV(d, t_1 **OU** t_2) = $\sqrt{\frac{RSV(d,t_1)^2 + RSV(d,t_2)^2}{2}}$;

  t_i étant un terme quelconque et a_i son poids associé dans le document d.

Les requêtes complexes contenant plusieurs termes sont, elles, traitées récursivement en appliquant les règles précédentes. Néanmoins, le modèle *p-norm* [97] propose une méthode permettant de calculer le score de similarité d'un document par rapport à la disjonction et la conjonction de plusieurs termes. Cette méthode se base sur les *p-distances* et est définie comme suit:

- RSV(d, t_1 **ET** t_2 **ET** … **ET** t_m) = $\sqrt[p]{\frac{a_1^p + a_2^p + \cdots + a_m^p}{m}}$;
- RSV(d, t_1 **OU** t_2 **OU** … **OU** t_m) = $1 - \sqrt[p]{\frac{(1-a_1)^p + (1-a_2)^p + \cdots + (1-a_m)^p}{m}}$;

  t_1, t_2 … t_m étant des termes quelconques et a_1, a_2 … a_m leurs poids associés dans le document d. Le paramètre entier p ($p \geq 1$) est indiqué au moment de la formulation de la requête.

L'avantage principal du modèle booléen étendu par rapport au modèle booléen de base est de pouvoir classer les documents suivant leur pertinence. Cependant, tout comme le modèle booléen de base, les requêtes restent complexes et difficilement formulées par l'utilisateur.

4.1.3 Le modèle des ensembles flous

Le modèle des ensembles flous [112] est une extension au modèle booléen de base. Il se base sur la théorie des ensembles flous où l'appartenance d'un élément à un ensemble est considérée comme probable et non certaine.

La requête y est décrite dans le langage booléen. Mais contrairement au modèle booléen de base, les termes se voient associés des poids (normalisés entre 0 et 1). Ceux-ci indiquent, dans le cadre de la théorie des ensembles flous, le degré d'appartenance d'un terme à un document. Ils sont utilisés par la fonction d'appariement pour calculer le degré de similarité d'un document à une requête de la façon suivante:

- RSV(d, t_i) = a_i ;

- RSV(d, **NON** t_i) = 1 - RSV(d, t_i);
- RSV(d, t_i **ET** t_j) = min(RSV(d, t_i), RSV(d, t_j));
- RSV(d, t_i **OU** t_j) = max(RSV(d, t_i), RSV(d, t_j));

  t_i et t_j étant des termes quelconques, et a_i le poids associé à t_i.

L'avantage principal de ce modèle est d'offrir la possibilité de classer les documents par ordre de pertinence. Cependant, il présente certains inconvénients. Notamment le fait de donner un score de similarité nul à la tautologie **t OU (NON t)** et un score non nul à l'antilogie **t ET (NON t)** (t étant un terme quelconque). Ceci fait qu'un système utilisant ce modèle peut retourner des résultats inattendus pour certaines requêtes.

4.2 Le modèle vectoriel

4.2.1 Le modèle vectoriel de base

Le modèle vectoriel [83] [85] est le modèle qui a le plus inspiré la recherche d'information après le modèle booléen. C'est un modèle qui utilise une représentation géométrique en considérant les termes d'indexation comme les dimensions d'un espace d'information multidimensionnel. La requête, qui est exprimée en langage naturel, et les documents sont alors représentés avec des vecteurs. La pertinence d'un document par rapport à une requête est relative aux postions de leurs vecteurs respectifs dans cet espace. La figure 2 illustre de façon graphique la représentation de deux documents ($\overrightarrow{d_1}$ et $\overrightarrow{d_2}$) et d'une requête ($\vec{q}$) dans un espace associé à deux termes.

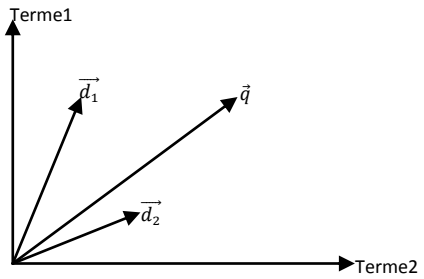


Figure 2: Exemple d'une représentation du modèle vectoriel

Dans un espace d'information à m termes $T = \{t_1, \dots, t_m\}$, chaque composante w_i d'un vecteur $\vec{v} = (w_1, \dots, w_m)$ désigne le poids d'indexation du terme t_i dans le document ou la requête représenté(e) par ce vecteur. Ces composantes sont utilisées par la fonction d'appariement pour mesurer la similarité d'un vecteur document $\vec{d} = (w_{d,1} \dots w_{d,m})$ à un vecteur requête $\vec{q} = (w_{q,1} \dots w_{q,m})$. Les mesures de similarité les plus utilisées sont [114]:

- Le produit interne (ou produit scalaire):

$$RSV(\vec{d}, \vec{q}) = \sum_{i=1}^{m} w_{q,i} \cdot w_{d,i} \; ;$$

- Le cosinus de l'angle entre les deux vecteurs:

$$RSV(\vec{d}, \vec{q}) = \cos(\vec{d}, \vec{q}) = \frac{\sum_{i=1}^{m} w_{q,i} \cdot w_{d,i}}{\sqrt{\sum_{i=1}^{m} {w_{q,i}}^2 \cdot \sum_{i=1}^{m} {w_{d,i}}^2}} \; ;$$

- La mesure de similarité de Dice (rapport sur la moyenne arithmétique des normes):

$$RSV(\vec{d}, \vec{q}) = \text{dice}(\vec{d}, \vec{q}) = \frac{2 \cdot \sum_{i=1}^{m} w_{q,i} \cdot w_{d,i}}{\sum_{i=1}^{m} {w_{q,i}}^2 + \sum_{i=1}^{m} {w_{d,i}}^2} \; ;$$

- La mesure de similarité de Jaccard (rapport de l'intersection sur l'union):

$$RSV(\vec{d}, \vec{q}) = \text{jaccard}(\vec{d}, \vec{q}) = \frac{\sum_{i=1}^{m} w_{q,i} \cdot w_{d,i}}{\sum_{i=1}^{m} {w_{q,i}}^2 + \sum_{i=1}^{m} {w_{d,i}}^2 - \sum_{i=1}^{m} w_{q,i} \cdot w_{d,i}} \; ;$$

- Le coefficient de superposition (*overlap*):

$$RSV(\vec{d},\vec{q}) = \text{overlap}(\vec{d},\vec{q}) = \frac{\sum_{i=1}^{m} w_{q,i} \cdot w_{d,i}}{\min(\sum_{i=1}^{m} {w_{q,i}}^2, \sum_{i=1}^{m} {w_{d,i}}^2)}.$$

En plus de ces formules, il existe un grand nombre d'autres formules qui peuvent être utilisées pour calculer le degré de similarité d'un document à une requête [71] [115]. Le modèle vectoriel ne définit pas la meilleure formule, et le choix de la formule à implémenter est laissé au programmeur. Toutefois, les méthodes les plus usuelles sont le calcul du produit scalaire et le calcul du cosinus. Le calcul du produit scalaire présente l'avantage d'être la méthode la plus rapide, mais retourne une valeur de similarité non normalisée. Quant au calcul du cosinus, c'est une méthode plus lente mais qui retourne une valeur de similarité normalisée (comprise entre *0* et *1*).

L'avantage du modèle vectoriel est de pouvoir classer les documents par ordre de pertinence par rapport à une requête. Il offre aussi la possibilité de limiter le nombre de documents retournés à l'utilisateur en ignorant les documents avec un degré de similarité inférieur à un certain seuil. En outre, les SRI se basant sur ce modèle présentent en général des résultats plus satisfaisants que ceux qui se basent sur le modèle booléen [102].

Néanmoins, ce modèle présente un inconvénient non négligeable, car il considère que les termes d'indexation forment une base, et donc qu'ils sont indépendants. Or, ceci est rarement le cas.

4.2.2 Le modèle vectoriel généralisé

Le modèle vectoriel généralisé [109] [108] a été proposé pour corriger les lacunes du modèle vectoriel de base, en considérant qu'il peut exister des dépendances entre les termes d'indexation. De ce fait, on ne peut plus représenter les documents et les requêtes dans un même espace que celui du modèle vectoriel de base où les axes représentent les termes d'indexation, car ceux-ci sont orthogonaux et impliquent l'existence d'une indépendance entre les termes.

Pour résoudre ce problème, un nouvel espace est défini où les axes ne représentent plus les termes d'indexation, mais un ensemble de descripteurs virtuels, appelés aussi *min-terms*, construit à partir des termes d'indexation en tenant compte de leurs cooccurrences dans les documents. Un document est alors représenté dans cet espace par un vecteur $\vec{d}$ défini comme suit:

$$\vec{d} = \sum_{i=1}^{m} w_{d,i} \cdot \overrightarrow{x_i}$$

Où:

- m est le nombre de descripteurs virtuels présents dans cet espace;
- $\overrightarrow{x_i}$ est un vecteur de base, associé au i^{ème} descripteur virtuel;
- $w_{d,i}$ est le degré de pertinence dans le document $\vec{d}$ du descripteur virtuel défini par $\overrightarrow{x_i}$.

Le vecteur associé à la requête est défini de la même manière, et le degré de similarité entre une requête et un document est calculé en appliquant les mêmes formules que celles utilisées dans le modèle vectoriel de base mais en utilisant les nouveaux vecteurs requête et document ainsi définis. De ce fait, le calcul du degré de similarité est plus couteux dans ce modèle, comparé au modèle vectoriel de base, mais il introduit la notion de dépendance entre les termes.

4.2.3 Le modèle Latent Semantic Indexing (LSI)

Le modèle *Latent Semantic Indexing (LSI)* [34] [24], ou modèle d'analyse sémantique latente, est une variante du modèle vectoriel de base qui a été introduite pour prendre en compte la sémantique des termes d'indexation. Ceci est fait en se basant sur la matrice d'occurrence représentant la collection de documents.

Dans une collection de n documents $D = \{d_1, \dots, d_n\}$, contenant les m termes d'indexation $T = \{t_1, \dots, t_m\}$, la matrice d'occurrence $F = (w_{i,j})_{i=1\dots m, j=1\dots n}$ est une matrice de dimension $m \times n$ où chaque élément $w_{i,j}$ représente le poids associé au terme t_i dans le document d_j:

$$F = \begin{pmatrix} w_{1,1} & \cdots & w_{1,n} \\ \vdots & \ddots & \vdots \\ w_{m,1} & \cdots & w_{m,n} \end{pmatrix} = \begin{pmatrix} \overrightarrow{d_1} & \cdots & \overrightarrow{d_n} \end{pmatrix}$$

L'idée du modèle LSI est de réduire la dimensionnalité de cette matrice tout en gardant le plus d'informations possibles. Pour ce faire, la matrice d'occurrence est décomposée en valeurs singulières pour donner lieu à une nouvelle matrice Σ de dimension $r \times r$ avec $r = \min(m, n)$:

$$F = U \cdot \Sigma \cdot V^t$$

Où:

- F est la matrice d'occurrence;
- U et V sont des matrices unitaires de dimension $m \times r$ et $n \times$ r respectivement, contenant un ensemble de vecteurs orthonormés ($U \cdot U^{\mathrm{t}} = 1$ et $V \cdot V^{\mathrm{t}} = 1$);
- V^{t} est la matrice transposée de la matrice V, de dimension $r \times n$;
- Σ est une matrice diagonale de dimension $r \times$ r contentant les valeurs singulières de F.

La matrice Σ est ensuite ordonnée de façon décroissante et réduite en se basant sur un certain coefficient $k <$ r, en considérant que les k valeurs les plus grandes de Σ suffisent pour représenter l'information contenue dans F:

$$F = U \cdot \Sigma \cdot V^t \approx \tilde{U} \cdot \tilde{\Sigma} \cdot \tilde{V}^t = \tilde{F}$$

Où:

- $\tilde{\Sigma}$ est la nouvelle matrice créée en se basant sur le coefficient k;
- $\tilde{U}$ et $\tilde{V}$ sont les matrices d'ordre $m \times k$ et $n \times$ k respectivement, construites à partir de U et de V;
- $\tilde{F}$ est la meilleure approximation de rang k de la matrice F au sens des moindres carrés.

Ces matrices sont ensuite utilisées pour définir la collection de documents par une nouvelle matrice F' de dimension $n \times k$:

$$F' = \tilde{V} \cdot \tilde{\Sigma} = \begin{pmatrix} w'_{1,1} & \cdots & w'_{1,k} \\ \vdots & \ddots & \vdots \\ w'_{n,1} & \cdots & w'_{n,k} \end{pmatrix} = \begin{pmatrix} \overrightarrow{d'_1} \\ \vdots \\ \overrightarrow{d'_n} \end{pmatrix}$$

Ainsi, une nouvelle représentation des documents de la collection est crée. Celle-ci, représente un document avec un nouveau vecteur $\overrightarrow{d_i'}$ de dimension $k \leq m$ avec : $\overrightarrow{d_i'} = (w'_{i,1}, \dots, w'_{i,k})$. Concrètement, cela signifie qu'un document dépend désormais d'un nombre plus limité de termes, car les termes identiques sémantiquement parlant, sont considérés comme une seule entité. La requête est aussi transformée en un nouveau vecteur de dimension $k \leq m$ comme suit:

$$\overrightarrow{q'} = \vec{q} \cdot \tilde{U} \cdot \tilde{\Sigma}^{-1}$$

Où $\vec{q}$ est le vecteur de la requête tel que défini dans le modèle vectoriel de base, et $\tilde{\Sigma}^{-1}$ la matrice inverse de $\tilde{\Sigma}$.

Le vecteur de la requête et du document étant définis, il ne reste plus qu'à définir la fonction d'appariement calculant le degré de similarité entre ces deux vecteurs. Celle-ci, tout comme dans le modèle vectoriel de base, consiste en un ensemble de formules (produit interne, cosinus, mesure de Dice …) appliquées sur les nouveaux vecteurs de la requête et du document.

L'avantage principal du modèle LSI est de prendre en compte la sémantique lors de l'appariement. Ceci en fait un modèle très utilisé pour la recherche d'information, mais aussi pour le filtrage d'information ou encore la recherche documentaire multilingue [25] [28]. Par ailleurs, ce modèle montre de meilleures performances que le modèle vectoriel de base pour les bases documentaires de petites et moyennes tailles.

Cependant, la mise en œuvre de ce modèle nécessite des traitements plus coûteux que le modèle vectoriel de base. De plus, la réduction de l'espace d'information, pose le problème du choix de la valeur k. En effet, il n'y a pas de moyen théorique général permettant de trouver la valeur optimale de k. Si on choisit une valeur trop petite, l'approximation sera mauvaise. Et si on choisit une valeur trop grande, la décomposition en valeurs singulières perd de son intérêt. C'est pourquoi, cette valeur est fixée empiriquement (les valeurs proches de 300 semblent donner de bons résultats). En outre, les dimensions de l'espace de représentation final sont difficilement interprétables, et représentent une combinaison linéaire de termes d'indexation (non pas des termes d'indexation comme c'est le cas dans le modèle vectoriel de base).

4.3 Le modèle probabiliste

4.3.1 Le modèle probabiliste de base

Le modèle probabiliste [67] [77] repose sur le principe de classement probabiliste (*Probability Ranking Principle*) qui stipule qu'un système retournant les documents dans l'ordre décroissant de leur probabilité de pertinence par rapport à la requête, opère de manière optimale. Ainsi, le système doit estimer de manière aussi précise que possible la probabilité de pertinence d'un document par rapport à une requête exprimée en langage naturel.

Cette probabilité est notée $P_q(pert|d_i)$, et exprime la probabilité que le document d_i fournisse des informations pertinentes pour la requête q. C'est une probabilité conditionnelle qui n'est pas directement calculable, et est d'abord décomposée en utilisant le théorème de Bayes:

$$P_q(pert|d_i) = \frac{P_q(d_i|pert) \cdot P(pert)}{P_q(d_i|pert) \cdot P(pert) + P_q(d_i|npert) \cdot P(npert)}$$

Où $P(d_i|pert)$ (respectivement $P(d_i|npert)$) indique la probabilité que d_i fasse partie de l'ensemble des documents pertinents (respectivement non pertinents) par rapport à la requête q. Quant à $P(pert)$ et $P(npert)$, elles indiquent respectivement la probabilité qu'un document quelconque soit pertinent ou non pertinent. Comme ces deux dernières probabilités sont fixes, on se limite aux deux valeurs $P_q(d_i|pert)$ et $P_q(d_i|npert)$ pour calculer le score de pertinence d'un document à une requête. Celles-ci sont calculées en se basant sur l'indexation binaire des documents où on considère que l'information la plus utile est la présence ou non d'un terme dans un document. De plus, on considère que les évènements liés à la présence ou l'absence de termes d'indexation sont mutuellement indépendants. De par ces hypothèses, la fonction d'appariement est alors définie comme suit:

$$RSV(d_i, q) = \log \frac{P_q(d_i|pert)}{P_q(d_i|npert)} = \sum_{j=1}^{m} x_{i,j} \cdot \left(\log \frac{P(t_j|pert)}{1 - P(t_j|pert)} + \log \frac{P(t_j|npert)}{1 - P(t_j|npert)} \right)$$

Où:

- t_j représente un terme d'indexation, tel que $t_j \in q$, et $q = \{t_1, \dots, t_m\}$;
- m représente le nombre total de termes d'indexation présents dans la requête q;
- $x_{i,j} \in \{0,1\}$ indique si le terme t_j est présent (1) ou non (0) dans le document d_i;
- $P_q(t_j|pert)$ indique la probabilité d'apparition du terme t_j sachant que le document appartient à l'ensemble des documents pertinents. Comme l'ensemble des documents pertinents n'est pas connu d'avance, une valeur fixe lui est généralement attribuée;
- $P_q(t_j|npert)$ indique la probabilité d'apparition du terme t_j sachant que le document appartient à l'ensemble des documents non pertinents. Cette probabilité correspond à df_j/n, où n indique le nombre de documents de la collection, et df_j le nombre de documents indexés par le terme t_j.

Le modèle probabiliste de base est un modèle tout à fait fonctionnel qui a démontré son efficacité en recherche d'information. Cependant il présente l'inconvénient de considérer les termes d'indexation comme étant indépendants les uns des autres, ce qui n'est généralement pas le cas. En outre, c'est un modèle qui génère des calculs de probabilités relativement complexes par rapport à d'autres modèles.

4.3.2 Les modèles de langue

Les modèles de langue [75] [11] utilisent une approche différente de celles des modèles présentés jusqu'ici. Dans cette approche, la pertinence d'un document par rapport à une requête est calculée en se basant sur la probabilité que la requête puisse être générée à partir du document.

On crée un modèle statistique M_d pour chaque document d. La correspondance du document d à une requête q est déterminée par la probabilité $P(q|M_d)$ que la requête q puisse être générée à partir du modèle M_d.

Pour une requête q contenant les m termes $\{t_1, \dots, t_m\}$, cette probabilité est calculée comme suit:

$$RSV(d, q) = P(q = (t_1, \dots, t_m)|M_d) = \prod_{j=1}^{m} P(t_j|d)$$

Avec:

$$P(t_j|d) = \frac{tf(t_j, d)}{\sum_{t \in d} tf(t, d)}$$

Où $tf(t_j, d)$ indique la fréquence d'occurrence du terme t_j dans d.

L'inconvénient de cette méthode est que si un seul terme t_j de la requête n'apparait pas dans le document d, alors la probabilité $P(q|M_d)$ sera nulle. Pour remédier à cela, on a recours à des techniques de lissage, qui consistent à affecter des valeurs non nulles aux termes qui n'apparaissent pas dans un document.

4.4 Le modèle connexionniste

4.4.1 Le modèle connexionniste de base

Le modèle connexionniste [59] [10] [5] est un modèle qui met à profit la théorie des réseaux de neurones dans la recherche d'information.

Un réseau de neurones s'inspire du fonctionnement des neurones d'un cerveau humain pour réaliser une tâche donnée. Un neurone peut être apparenté à une fonction, où les valeurs d'entrées correspondent aux *dendrites* du neurone biologique, et la valeur de retour (sortie) de la fonction correspond à l'*axone* du neurone biologique. Cette fonction est en général paramétrée avec des poids qui déterminent en partie la sortie de la fonction. Une représentation graphique d'un neurone est montrée dans la figure 3.

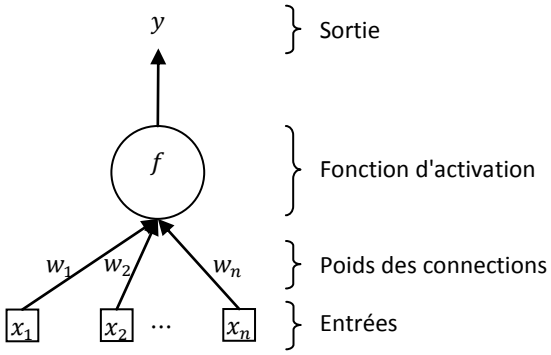


Figure 3: Représentation d'un neurone

L'intérêt des neurones réside dans les propriétés qui résultent de leur association en réseaux. Ils sont généralement organisés en couches, où on distingue une couche d'entrée, un certain nombre de couches cachées et une couche de sortie. Par ailleurs, la mise en œuvre d'un réseau de neurones passe automatiquement par une phase d'apprentissage, où les poids des connections entre les neurones sont ajustés de telle sorte qu'il retourne les résultats attendus.

Les systèmes de recherche d'information basés sur l'approche connexionniste utilisent les fondements des réseaux de neurones tant pour la modélisation des unités textuelles que pour la mise en œuvre du processus de recherche d'information. Les représentations utilisées reposent généralement sur la modélisation par un réseau de neurones à trois couches interconnectées: la couche requête, la couche termes et la couche documents. L'interrogation se fait par la propagation de signaux de la couche d'entrée vers la couche de sortie via la couche intermédiaire comme illustré dans la figure 4.

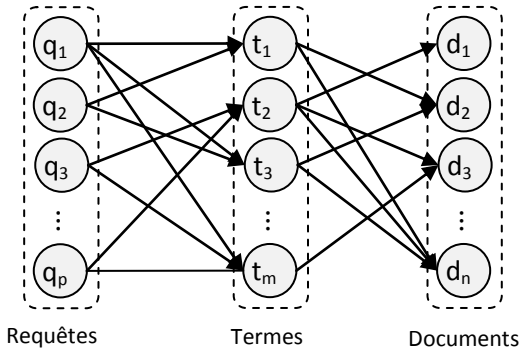


Figure 4: Un modèle de réseau de neurones pour la RI

Dans de tels systèmes, l'apprentissage se fait par le calcul d'erreur dans la couche documents, qui est suivi par la rétro propagation vers les autres couches du réseau afin d'ajuster les poids des connections.

5 Évaluation des systèmes de recherche d'information

L'évaluation constitue une étape importante dans la mise en œuvre d'un système de recherche d'information. Elle permet de mesurer la qualité du système en comparant les réponses du système avec les réponses idéales attendues par l'utilisateur. Ainsi, plus les réponses du système correspondent à celles attendues par l'utilisateur, meilleur est le système. Dans ce qui suit, nous définissons les notions de base nécessaires à l'évaluation d'un SRI, après quoi, nous présentons la compagne d'évaluation *TREC* qui est actuellement la plus utilisée.

5.1 Notions de bases

5.1.1 La précision et le rappel

Ce sont les mesures les plus utilisées pour l'évaluation des systèmes de recherche d'information. Elles sont calculées relativement à la réponse du système (voir figure 5), et représentent des critères de qualité lors de l'évaluation d'un SRI.

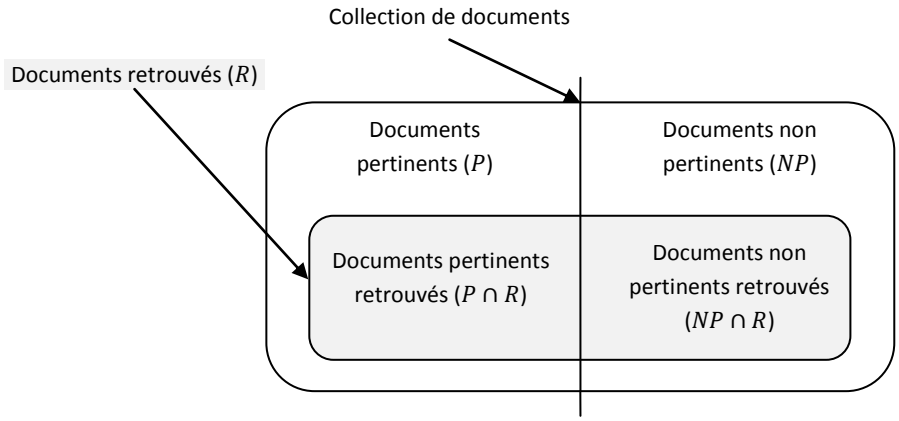


Figure 5: Répartition des documents d'une collection suite à une requête

5.1.1.1 Le rappel

Le rappel mesure la proportion de documents pertinents retrouvés parmi tous les documents pertinents de la collection. Il est calculé avec la formule suivante:

$$rappel = \frac{|P \cap R|}{|P|}$$

Où:

- $|P|$ représente le nombre de documents pertinents dans la collection de documents;
- $|P \cap R|$ représente le nombre de documents pertinents retournés par le système.

Le rappel mesure la capacité du SRI à sélectionner tous les documents pertinents. Un taux de rappel de *1*, signifie que tous les documents pertinents ont été restitués. Inversement, un taux de rappel de *0*, signifie qu'aucun document pertinent n'a été restitué par le système.

Le rappel permet aussi de définir le *silence documentaire* qui représente la proportion des documents pertinents non retrouvés par le système:

$$silence = 1 - rappel$$

5.1.1.2 La précision

La précision mesure la proportion de documents pertinents restitués parmi tous les documents restitués. Elle est calculée avec la formule suivante:

$$précision = \frac{|P \cap R|}{|R|}$$

Où:

- $|R|$ représente le nombre de documents retournés par le système;
- $|P \cap R|$ représente le nombre de documents pertinents retournés par le système.

La précision mesure la capacité du système à rejeter tous les documents non pertinents, et de ce fait, sa capacité à trouver exclusivement les documents pertinents. Un taux de précision de *1* signifie que seulement les documents pertinents ont été restitués. Tandis qu'un taux de précision de *0*, signifie qu'aucun des documents retournés n'est pertinent.

La précision permet aussi de définir le *bruit documentaire* qui mesure la proportion de documents non pertinents retournés par le système:

$$bruit = 1 - précision$$

5.1.2 La précision à x

La précision à x est la précision obtenue en considérant les x premiers résultats retournés par le SRI (en supposant que ceux-ci sont triés suivant leur pertinence).

5.1.3 La précision exacte ou la R-précision

La précision exacte ou la R-précision est la précision à x obtenue quand x est égal au nombre total de documents pertinents par rapport à la requête.

5.1.4 La précision moyenne

La précision moyenne est la moyenne arithmétique de précisions pour chaque document pertinent. La précision d'un document pertinent est la précision à x, où x représente la position de ce document dans la liste des documents retournés par le système. Elle est calculée avec la formule suivante:

$$précision\ moyenne = \sum_{x=1}^{n} \frac{p(x) \cdot pert(x)}{npert}$$

Où:

- n indique le nombre de documents retournés par le système;
- $p(x)$ indique la précision à x ;
- $pert(x)$ est une fonction qui vaut *1* si le document à la position x est pertinent, *0* sinon;
- $npert$ est le nombre total de documents pertinents retournés.

5.1.5 F-mesure

Le rappel et la précision sont les principales mesures utilisées pour l'évaluation d'un SRI. Mais il existe d'autres mesures qui ne sont pas moins significatives, et qui pour la plupart combinent la précision et le rappel. Comme la F-mesure (ou moyenne harmonique) qui est calculée comme suit :

$$F(x) = \frac{2 \cdot (précision\ \text{à}\ x) \cdot (rappel\ \text{à}\ x)}{(précision\ \text{à}\ x) + (rappel\ \text{à}\ x)}$$

5.1.6 La courbe rappel-précision

La plupart des approches d'évaluation des SRI se basent sur les deux mesures de rappel et de précision. En effet, plus ces deux valeurs sont proches de *1* pour un système donné, plus ce système est performant. Or, il existe une forte corrélation entre ces deux valeurs. Et, en pratique, plus l'une des deux valeurs augmente plus l'autre diminue. Ainsi en faisant varier l'une des valeurs on peut obtenir une courbe représentant la précision en fonction du rappel (ou inversement).

Il est possible, grâce à cette représentation, de comparer deux systèmes de recherche d'information. Particulièrement, si les deux courbes ne se croisent pas, on peut déduire que le système dont la courbe se trouve au dessus de l'autre, est plus performant (car il offre un taux de rappel et de précision plus élevé).

En outre, plus la courbe rappel-précision d'un système décroit tardivement, plus ce système est performant.

Un exemple d'une courbe rappel-précision est présenté dans la figure 6.

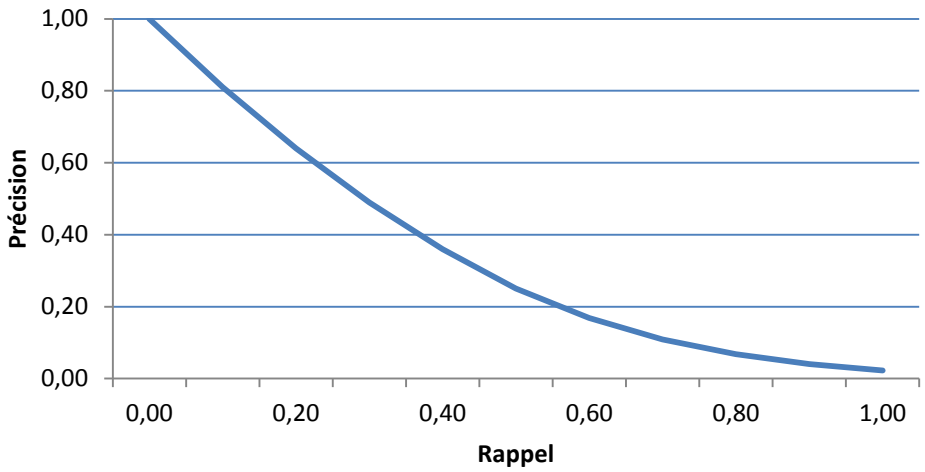


Figure 6: Courbe rappel-précision

5.2 Les campagnes d'évaluation et les collections de référence

L'évaluation est une étape importante dans l'élaboration d'un SRI. Elle permet de mesurer les caractéristiques du système et de fournir une base de comparaison entre différents modèles. Parmi ces caractéristiques on trouve notamment le temps de réponse du système et la qualité de réponse du système. La qualité d'un système est mesurée en comparant les réponses du système avec les réponses idéales que l'utilisateur espère recevoir. Plus les réponses du système correspondent à celles que l'utilisateur espère, mieux est le système.

Pour pouvoir réaliser une telle évaluation, on utilise des collections de tests fournies dans différentes compagnes d'évaluation. Une collection de tests est composée d'un ensemble de documents, d'un ensemble de requêtes et de la liste des documents pertinents pour chaque requête.

Les compagnes d'évaluation définissent aussi les critères qui doivent être utilisés pour l'évaluation, les plus utilisés sont le rappel et la précision.

Parmi les compagnes d'évaluation qui ont été entreprises au cours de l'histoire, on trouve la compagne de *Cranfield* [18] qui est considérée comme la première compagne d'évaluation, et les compagnes *TREC* qui constituent le modèle dominant et que nous développons par la suite.

5.2.1 Les compagnes TREC

Les compagnes TREC (*Text REtrieval Conferences*) [98] [43] constituent l'approche la plus utilisée dans le domaine de l'évaluation des systèmes de recherche d'information. C'est un projet qui a été lancé en 1992 par le NIST (*National Institue of Standards and Technology*) avec le soutien financier de l'IARPA (*Intelligence Advanced Research Projects Activity*). L'objectif de TREC est d'encourager les travaux dans le domaine de la recherche d'information, en fournissant l'infrastructure nécessaire (collections de tests) et en proposant des méthodologies d'évaluation des SRI.

La collection de tests qu'offre TREC comprend un ensemble de documents ainsi qu'un lot de thèmes (*topics*), chacun représentant un besoin en information. L'évaluation d'un SRI dans le cadre de la compagne d'évaluation TREC consiste à examiner les 1000 premiers résultats qu'il retourne pour chaque thème (requête). Une précision moyenne est ensuite calculée et représente une mesure de performance globale du système.

6 Conclusion

Dans ce chapitre, nous avons présenté les principales notions et les principaux concepts de la recherche d'information. Nous avons développé les étapes d'un processus de recherche d'information, soit le processus d'indexation, le processus d'appariement et le processus de reformulation de la requête. Après quoi, nous avons vu les principaux modèles de recherche d'information. Et enfin, nous avons présenté les principaux critères d'évaluation d'un SRI.

Jusqu'à présent, nous nous sommes intéressés à la recherche d'information en ne considérant que la vue contenu des documents. Dans le chapitre suivant, nous nous intéressons à une autre facette de la recherche d'information, en considérant en plus du contenu « plat » des documents, leur structure. Celle-ci est représentée dans un document grâce au format XML. La recherche d'information dans les documents XML est l'objet de notre étude. Nous la détaillons dans le chapitre suivant.

Chapitre 2: Recherche d'information et structure

1 Introduction

La recherche d'information n'a longtemps considéré que le contenu « *plat* » des documents, ignorant de ce fait, leurs structures. Cependant, avec l'essor du web et la démocratisation du XML (*eXtensible Markup Language*), la notion de structure se fait de plus en plus présente dans les documents numériques, et il est devenu primordial d'en tenir compte lors du processus de recherche d'information.

L'avantage qu'apporte l'intégration de la structure dans le processus de recherche d'information est multiple. Premièrement, l'utilisateur pourra mieux exprimer son besoin en informations grâce à des requêtes pouvant contenir, en plus des mots clés habituels, des contraintes structurelles qui permettrons, entre autres, de désambigüiser les termes de recherche. Deuxièmement, une réponse plus précise pourra être apportée à l'utilisateur en ne retournant que la partie jugée intéressante d'un document, chose qui n'est pas proposée par la recherche d'information classique qui considère le document comme un granule d'information indivisible. Et troisièmement, la qualité de réponse du système peut être grandement améliorée en exploitant l'information que véhicule la structure des documents et tout particulièrement les liens de citation et de référence.

Dans cette optique, plusieurs approches ont été proposées, et s'articulent, pour la plupart, autour du langage XML, qui est le langage de prédilection pour stocker des données structurées.

L'objectif de ce chapitre est de présenter en premier lieu le langage XML, en décrivant notamment la structure d'un document XML et la technologie *XLink* qui permet de créer des liens entre des documents XML. Ensuite, on s'intéresse à la problématique de la recherche d'information dans un document structuré. On présente, en particulier les approches de la RI structurée. Puis, on se penchera sur les différentes étapes du processus de recherche d'information structurée, soit le processus d'indexation, et le processus d'appariement. Pour finir, on présente les techniques d'évaluation des SRIS (système de recherche d'information structurée) à travers la présentation de la compagne d'évaluation INEX.

2 Présentation de XML

Le XML[1] acronyme de « *eXtensible Markup Language* », en français « *langage de balisage extensible* », est un langage informatique de balisage générique qui dérive du SGML (*Standard Generalized Markup Language*). Il a été développé par le « *XML Working Group* » formé sous les auspices du « *World Wide Web Consortium* » (W3C[2]) en 1996, qui en a fait une recommandation officielle dés février 1998[3].

XML a dés lors connu un grand succès auprès des développeurs. Car d'une part, il supporte un grand nombre d'applications, ce qui lui permet notamment d'être directement utilisable sur internet. Et d'autre part, il permet d'écrire des documents pouvant être facilement manipulés par des programmes informatiques, tout en restant lisibles par l'homme.

Dans ce qui suit nous présentons la structure d'un document XML, ensuite nous présentons la technologie *XLink* qui permet d'ajouter des liens dans un document XML et enfin nous introduisons la notion de parseur XML.

[1] http://www.w3.org/XML/
[2] http://www.w3.org/
[3] http://www.w3.org/TR/1998/REC-xml-19980210

2.1 Structure d'un document XML

Un document XML est un document semi-structuré sous forme textuelle. Par document semi-structuré, on entend un document dont la structure n'est pas aussi rigide, aussi régulière ou complète que la structure requise par les systèmes de gestion de bases de données traditionnels [1]. Concrètement, cela signifie que la structure de tels documents n'est pas pré-imposée, mais est déterminée par les besoins de conception.

Ci-dessous est présenté un document XML simple:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- Ceci est un document XML représentant la structure du chapitre
précédent -->
<chapitre auteur="Sadi Samy">
    <intitulé>État de l'art de la recherche d'information</intitulé>
    <plan>
        <titre>Introduction</titre>
        <titre>Concepts de base de la recherche d'information</titre>
        <titre>Le processus de recherche d'information</titre>
        <titre>Modèles de recherche d'information</titre>
        <titre>Évaluation des systèmes de recherche d'information</titre>
        <titre>Conclusion</titre>
    </plan>
    <révision numéro="2" date="01/03/2011" />
</chapitre>
```

Figure 7: Exemple d'un document XML

De par cet exemple, le document XML apparait comme un document texte lisible duquel on peut aisément déduire la teneur, qui s'agit là de la description du premier chapitre du présent mémoire. Il est ainsi possible, grâce aux différentes balises utilisées, de facilement identifier le titre du chapitre, son auteur ainsi que le plan adopté.

L'usage de balises est inhérent et indispensable à tout document XML, mais celles-ci ne forment pas l'intégralité de la syntaxe de XML. D'autres composants, peuvent aussi apparaitre dans un document XML. L'ensemble de ces composants est décrit dans la spécification XML[4] fournie par le W3C, qui décrit aussi les règles syntaxiques et lexicales que doit respecter un document XML pour être considéré comme « *bien formé* ».

Parmi les composants d'un document XML on trouve :

- *Le prologue* : il définit la version de XML (1.0 ou 1.1) et le jeu de caractères (encodage) utilisés dans le document. Dans la figure donnée en exemple (figure 7), le prologue se trouve à la première ligne;
- *Les instructions de traitement* : elles n'ont pas de rôle lié aux données ou à la structuration d'un document XML. Elles servent à donner à l'application qui utilise le document XML des informations supplémentaires telles que l'ajout d'un feuillet de style (voir annexe A);
- *La déclaration du type de document* : c'est une déclaration optionnelle qui sert à attacher une grammaire de type DTD (*Document Type Definition*) à un document XML;

[4] http://www.w3.org/TR/REC-xml/

- *Les éléments (ou nœuds)* : ils gèrent la structuration des données d'un document XML. On peut les qualifier de métadonnées, au sens où ils ne font pas partie réellement des données mais servent à en désigner la nature. Un élément est désigné par un nom et peut contenir du texte, d'autres éléments, un mélange de texte et d'éléments, ou bien être vide;
- *Les balises* : elles servent à désigner les constructions entre deux chevrons <…> dans un document XML. Les balises sont utilisées par paires (une balise ouvrante suivie par une balise fermante) pour délimiter le contenu des éléments;
- *Les attributs* : ce sont des doublets (nom, valeur) qualifiant un élément. Celui-ci peut alors avoir plusieurs attributs de nom différents;
- *Les éléments textes* : c'est le texte qui est associé à l'attribut, c'est-à-dire sa valeur, ou à l'élément, c'est-à-dire son contenu;
- *Les commentaires* : ils permettent d'ajouter des remarques ou des notes dans un document XML. Les commentaires sont destinés aux développeurs et ne sont pas porteurs d'informations quant à la structure ou au contenu d'un document XML.

La structure d'un document XML n'est pas imposée par un standard. Cependant, il est commun d'associer aux documents XML une grammaire exprimée à l'aide d'une DTD ou d'un schéma XML (voir annexe A) pour leur imposer une structure particulière. Dés lors, un document XML conforme à cette grammaire est dit *« valide »* et la validation d'un document XML consiste à vérifier qu'il est conforme à cette grammaire.

2.2 Les liens dans les documents XML: XLink

Un lien est une référence dans un document qui permet de définir une connexion entre un élément du document et un autre élément qui peut se trouver dans le même document ou dans un autre document. Il est commun d'utiliser l'appellation « *hypertexte* » pour désigner un document contenant des liens, et les appellations « *lien hypertexte* » et « *hyperlien* » pour désigner le lien. Dans un document XML, les liens sont insérés grâce à la technologie XLink.

XLink pour « *XML Linking language* », est une recommandation du W3C[5] depuis 2001 qui permet de créer des liens entre des documents XML ou des parties de documents XML grâce à *XPointer*[6]. Un lien XLink peut être ajouté à n'importe quel élément d'un document XML. Pour cela, un certain nombre d'attributs contenus dans l'espace de nom « *http://www.w3.org/1999/xlink* », sont mis à disposition des développeurs. Parmi ces attributs, on trouve notamment l'attribut « *type* » qui définit le type du lien, et l'attribut « *href* » qui définit l'URL[7] où pointe le lien.

XLink définit plusieurs types de liens dont le plus commun est le type « *simple* » qui permet de créer un lien simple reliant deux documents, où le document source est le document où est déclaré le lien, et le document destination est identifié par l'URL contenue dans l'attribut « *href* ». En outre, XLink offre le type « *extended* » pour gérer les liens plus complexes impliquant plusieurs documents XML.

[5] http://www.w3.org/TR/xlink11/

[6] XPointer est une spécification du W3C (http://www.w3.org/TR/xptr/) qui permet de désigner une partie d'un document XML grâce à une syntaxe XPath

[7] URL pour « *Uniform Resource Locator* », défini dans la RFC 3986: http://tools.ietf.org/html/rfc3986

Voici un exemple d'un document XML contenant des liens XLink:

```
<?xml version="1.0" encoding="UTF-8" ?>
<bibliothèque xmlns:xlink="http://www.w3.org/1999/xlink">
  <livres>
    <!-- le lien suivant pointe vers un document externe -->
    <livre
     catégorie="INFORMATIQUE"
     xlink:type="simple"
     xlink:href="http://www.amazon.com/Automatic-Information-Organization-
Retrieval-Gerald/dp/B000OFZ8HG">
      <titre lang="en">
         Automatic Information Organization and Retrieval
      </titre>
      <!-- le lien suivant pointe vers l'élément auteur dans ce document grâce à
XPointer -->
      <auteur xlink:type="simple" xlink:href="#xpointer(idAuteur('1741'))">
        Gerald Salton
      </auteur>
      <année>1968</année>
      <prix>34.79</prix>
    </livre>
  </livres>
  <auteurs>
    <auteur idAuteur="1741">
      <nom>Salton</nom>
      <prénom>Gerald</prénom>
    </auteur>
  </auteurs>
</bibliothèque>
```

Figure 8: Exemple d'un document XML contenant des liens XLink

2.3 Les parseurs XML

Un parseur (ou analyseur) XML est un module logiciel utilisé pour lire les documents XML et pour accéder à leur contenu et à leur structure. De plus, un parseur XML peut également permettre de vérifier la validité d'un document XML (conformément à une DTD ou à un schéma), on dit alors que le parseur est *« validant »*.

Selon l'approche utilisée pour traiter le document, deux types de parseurs sont à distinguer. Les parseurs utilisant une approche hiérarchique pour traiter le document, grâce à l'interface DOM. Et les parseurs utilisant une approche évènementielle pour traiter le document, grâce à l'interface SAX. Ces deux approches sont présentées succinctement dans ce qui suit.

2.3.1 DOM

DOM[8] (*Document Object Model*) est une spécification du W3C qui définit une interface de programmation d'applications (API) indépendante de toute plateforme et de tout langage, permettant d'accéder et de modifier de façon dynamique le contenu et la structure d'un document XML.

Techniquement, DOM construit l'arborescence de la structure d'un document XML et de ses éléments. Pour ce faire, le document est entièrement chargé en mémoire, ce qui fait que les programmes utilisant DOM ont généralement une empreinte mémoire volumineuse en cours de traitement. Toutefois, un programme utilisant DOM a, à tout instant, une vue globale du document XML traité. Ceci simplifie les traitements, et permet de générer un document XML à partir d'un arbre DOM présent en mémoire assez facilement.

[8] http://www.w3.org/DOM/

La figure suivante représente l'arbre associé au document XML donné dans la figure 7.

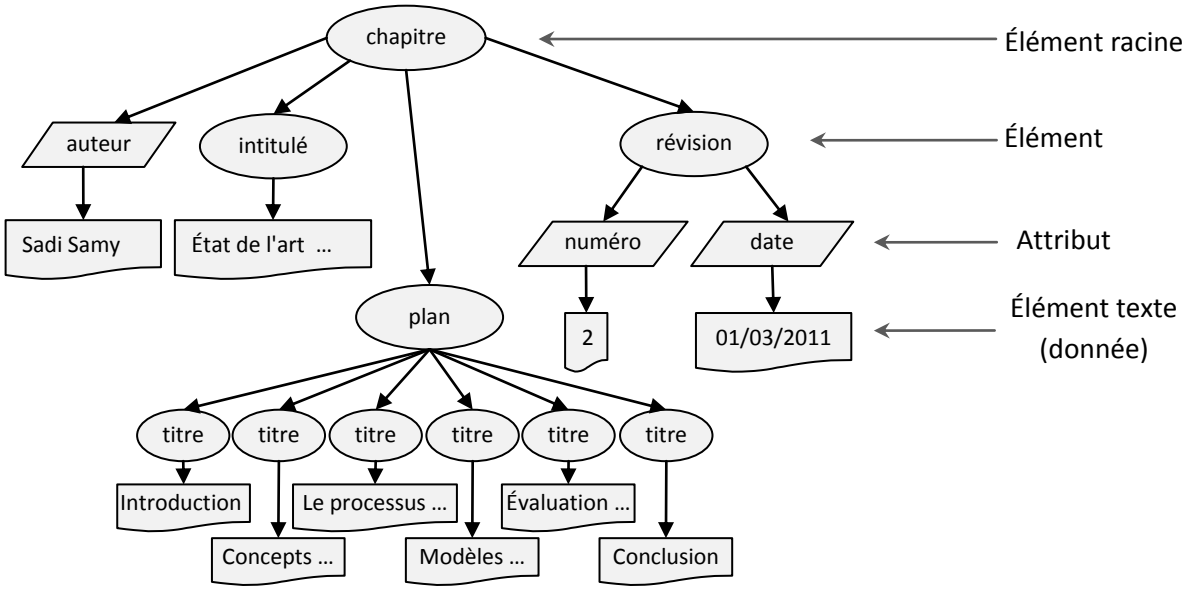


Figure 9: Exemple d'arbre DOM

2.3.2 SAX

SAX[9] (*Simple API for XML*) est une API (*Application Programming Interface*) développée par les membres du groupe de discussion *XML-dev* (*XML-dev mailing list*). Elle a été proposée comme une alternative à DOM, afin de proposer une solution moins gourmande en mémoire pour l'analyse des documents XML. Originellement spécifique à Java, SAX s'est par la suite généralisé à d'autres langages de programmation.

SAX se base sur un modèle évènementiel pour l'analyse d'un document XML. Cela signifie que, lors de l'analyse du document par le parseur, des évènements (signalant le début ou la fin du document, l'ouverture ou la fermeture d'un élément, du contenu texte …) sont générés et envoyés à l'application. Celle-ci peut alors soit les prendre en considération soit les ignorer. Par conséquent, l'application n'a pas de représentation globale du document comme c'est le cas avec DOM, mais a, en revanche, une empreinte mémoire plus faible et permet d'analyser un document plus rapidement surtout si celui-ci est d'une taille importante.

3 Problématiques de la RI structurée

La prise en compte de la structure dans le processus de recherche d'information impose de nouveaux défis. Le premier défi concerne le choix de l'unité d'information pertinente à retourner à l'utilisateur. Le second défi concerne l'expression du besoin en informations de l'utilisateur. Et le troisième défi concerne la prise en compte des liens de référence entre les documents. Ces trois points sont développés dans ce qui suit.

3.1 L'unité d'information pertinente

La recherche d'information structurée permet de mieux définir la notion « *d'unité d'information pertinente* » grâce à la prise en compte de la structure des documents. En effet, contrairement à la RI classique où le document est considéré comme un granule d'information indivisible, dans la RI

[9] http://www.saxproject.org/

structurée, tout élément d'un document XML (sous-arbre du document) peut être considéré comme une réponse potentielle à une requête.

Le choix de l'unité d'information pertinente revient à choisir le ou les éléments du document qui répondent le mieux à la requête de l'utilisateur [89]. Ceci revient à choisir la partie du document la plus *exhaustive* et la plus *spécifique* qui répond à la requête [15] [60]. On dit qu'une unité d'information est exhaustive, si elle contient toutes les informations requises par la requête, et on dit qu'elle est spécifique, si tout son contenu concerne la requête.

Cependant, si la requête ne contient que des mots clés, le choix de l'unité d'information à retourner à l'utilisateur peut alors être problématique. En effet, considérons une requête qui contient le texte « *la recherche d'information* », dans l'exemple donné dans la figure 7 deux éléments (*intitulé* et *titre*) répondent à cette requête. Le problème qui se pose alors est le choix de l'unité d'information à retourner à l'utilisateur. Faut-il regrouper les deux éléments comme une seule unité d'information? Ou alors faut-il retourner l'élément englobant ces deux éléments? Dans le premier cas, on garantit la spécificité de l'information, mais l'unité d'information peut alors paraitre confuse à l'utilisateur car celle-ci n'est pas une unité logique contigüe dans la mesure où des éléments adjacents aux éléments sélectionnés peuvent contenir une information complémentaire nécessaire à l'utilisateur. Dans le second cas, on garantit l'exhaustivité de l'information, mais l'unité d'information peut alors contenir des éléments qui ne concernent pas la requête.

3.2 L'expression du besoin en informations

Dans la recherche d'information structurée, l'utilisateur doit pouvoir formuler des requêtes pouvant contenir, en plus des mots clés habituels, des contraintes structurelles. Celles-ci seront utilisées conjointement avec les mots clés pour sélectionner les unités d'information à retourner à l'utilisateur.

Pour exprimer les contraintes structurelles, on a recours à des langages de requête plus évolués que ceux proposés dans la recherche d'information classique. Ceux-ci sont, pour la plupart, basés sur *XPath* et sont présentés plus loin dans ce chapitre.

3.3 La prise en compte des liens

Les systèmes de recherche d'information classique se basent uniquement sur le contenu textuel des documents. Ils considèrent le document comme une entité indépendante dans la collection de documents, et sa pertinence par rapport à une requête n'est identifiée que par son contenu. Or, en RI structurée, il existe généralement des liens de citations et de références entre les documents d'une collection, qui peuvent contribuer à améliorer la qualité des résultats retournés par le système de recherche d'information. En effet, un document cité par un nombre important de documents peut être considéré comme populaire, et devrait donc être plus pertinent qu'un document pas ou peu cité dans la collection.

4 Approches de RI dans les documents XML

Pour répondre aux problématiques citées précédemment, et permettre la recherche d'information dans des documents semi-structurés (documents XML), plusieurs approches ont été proposées, et se classent en deux principales catégories: *les approches orientées données* proposées par la communauté des bases de données (BD), et *les approches orientées documents* prises en

charge par la communauté de la recherche d'information (RI). Ces approches sont décrites dans ce qui suit.

4.1 Les approches orientées données

Les approches orientées données (*data-centric*) considèrent le document XML comme un ensemble de données typées et relativement homogènes. Elles s'intéressent davantage à la structure du document en considérant le document XML plus comme un support de stockage que comme un document textuel.

Ces approches utilisent les bases de données pour stocker toute l'information textuelle et structurelle des documents. C'est pourquoi la plupart des langages de requêtes qui y sont utilisés, sont des langages proches de la syntaxe de *SQL*. Ils permettent de faire des requêtes avec des contraintes très précises sur la structure, mais n'offrent guère de souplesse au niveau des contraintes sur le contenu textuel qui sont généralement traitées de manière booléenne (présent ou absent). Dés lors, il n'est plus possible de trier les résultats restitués à l'utilisateur suite à une requête.

Un exemple d'une requête portant sur la structure pour laquelle il convient d'utiliser ce type d'approches est la suivante « *trouver tout les étudiants âgés de 20 ans* ». Celle-ci ne requiert aucun triage des résultats, et un appariement exact de l'âge des étudiants avec l'âge souhaité suffit pour répondre au besoin en informations de l'utilisateur.

4.2 Les approches orientées documents

Les approches orientées documents (*document-centric*) considèrent les documents semi-structurés comme une collection de documents textes comportant des balises et des relations entre ces balises.

Ces approches se basent, pour la plupart, sur les modèles de la RI classique qui sont étendus pour intégrer la structure dans le processus de recherche d'information. Elles utilisent des langages de requêtes plus simples que ceux proposés dans les approches orientées données. En outre, l'appariement entre une requête et un document se fait, comme dans la RI classique, grâce au calcul d'un degré de pertinence. Ainsi, une plus grande souplesse est offerte à l'utilisateur car il n'est pas nécessaire qu'il y ait une correspondance exacte entre la requête et les résultats, et, de plus, ceux-ci peuvent être triés.

4.3 Récapitulatif

Le tableau suivant résume les propriétés clés de la recherche d'information classique, de la recherche d'information structurée utilisant les approches orientées données et de la recherche d'information structurée utilisant les approches orientées documents.

	RI classique	RIS: approches orientées documents	RIS: approches orientées données
Type de documents	documents textuels plat (non structurés)	documents semi-structurés	documents structurés
Contenu des documents	texte	texte + structure	données + structure
Unité d'information recherchée	document	élément	tuple
Besoin en informations	flou	flou	précis
Requête	mots clés	mots clés + contraintes structurelles	SQL et dérivés

Tableau 1: Récapitulatif de quelques caractéristiques de la RI structurée

Comme l'illustre ce tableau, on distingue des différences clés entre les deux types d'approches. Néanmoins, ces différences tendent de plus en plus à disparaitre. Ainsi, les solutions proposées par la communauté de RI peuvent être utilisées conjointement avec les solutions proposées par la communauté des BD.

Dans ce chapitre, nous nous focalisons sur les approches orientées documents qui sont proposées par la communauté RI.

5 Indexation des documents XML

L'indexation dans la RI structurée n'est pas aussi simple que pour la RI classique. En effet, vu la présence de la structure en plus du contenu textuel dans les documents traités, il est nécessaire d'indexer ces deux informations conjointement, en gardant les relations qui existent entre elles. Pour ce faire, l'indexation d'un document semi-structuré passe par deux étapes: l'indexation de l'information textuelle et l'indexation de l'information structurelle. Ces deux étapes sont décrites dans ce qui suit.

5.1 Indexation de l'information textuelle

L'information textuelle dans les documents XML se trouve dans les éléments textes (éléments feuilles). Il s'agit d'un ensemble de termes d'indexation qui sont pondérés afin de refléter leur importance.

L'indexation de l'information textuelle est étroitement liée à la structure du document car l'importance du texte est relative à son emplacement dans le document (c'est-à-dire l'élément dans lequel il se trouve). Deux problèmes apparaissent alors: le premier est relatif à la porté des termes et le second concerne la pondération des termes [88].

5.1.1 Porté des termes d'indexation

Les éléments textes qui contiennent l'information textuelle peuvent avoir un nombre quelconque d'éléments parents. Dés lors, de nouvelles questions se posent quant à la façon de lier cette information textuelle avec l'information structurelle que véhiculent ces éléments. Faut-il lier l'information textuelle avec tous les éléments parents où elle apparait? Ou alors faut-il la lier avec un seul de ces éléments?

Deux solutions ont été proposées pour répondre à ces questions: l'indexation avec les sous-arbres imbriqués et l'indexation avec la décomposition en unités disjointes. Elles sont décrites dans ce qui suit.

5.1.1.1 *Sous-arbres imbriqués*

On considère dans ce cas que le texte complet de chaque élément est une unité atomique [51] [68]. Les termes des éléments feuilles sont alors propagés dans l'arbre du document. Ainsi, chaque élément est associé avec tous les termes qui apparaissent dans l'un de ces descendants. De ce fait l'index contient des informations redondantes.

Le poids associé aux termes est généralement différent selon l'élément vers lequel il est propagé. Généralement, on calcule un poids initial pour chaque terme dans l'élément qui le contient. Ensuite, pour chaque élément parent, on attribue au terme un nouveau poids obtenu après réduction de son poids initial selon la distance entre cet élément et l'élément qui contient le terme [90].

5.1.1.2 *Unités disjointes*

Le document est décomposé en unités disjointes, de telle façon que le texte de chaque élément est l'union d'une ou de plusieurs parties disjointes [32] [36] [3] [81]. Les termes des éléments feuilles sont uniquement reliés à l'élément parent qui les contient. Ainsi, les éléments ne contenant pas de texte ne sont reliés à aucun terme (même s'ils contiennent des sous-éléments).

5.1.2 Pondération des termes

Les approches d'indexation dans la RI structurée extraient les termes d'indexation selon des processus similaires à ceux utilisés en recherche d'information classique. La pondération de ces termes doit cependant être vue sous un nouvel angle. Alors qu'en RI traditionnelle, le poids d'un terme cherche à rendre compte de son importance de manière locale au sein du document et de manière globale au sein de la collection, s'ajoute en RI structurée l'importance du terme au niveau de l'élément qui le contient [88].

De ce fait, les mesures classiques (IDF et TF) ont été adaptées pour la RI structurée. De plus, une autre mesure, la mesure IEF (*Inverse Element Frequency*) [37] [105] [39], a été proposée comme alternative à IDF qui est jugé peu efficace dans les documents XML, où le nombre de répétitions des termes est généralement assez réduit. La formule utilisée par IEF est la suivante:

$$ief(E, t_i) = \log \frac{|E|}{\left|\{e_j \in E : t_i \in e_j\}\right|}$$

Avec:

t_i : le terme dont on cherche la fréquence *ief*;

E : l'élément par rapport auquel est calculé *ief*;

$|E|$: le nombre total d'éléments que contient E;

$\left|\{e_j \in E : t_i \in e_j\}\right|$: le nombre d'éléments dans E où le terme t_i apparait;

$ief(t_i)$: la fréquence du terme t_i.

Certains auteurs [113], considèrent que le calcul du poids des termes est influencé par le contexte (unité d'indexation) dans lequel ils apparaissent. Ce calcul de poids s'inspire de la méthode *tf-idf* que nous avons appliquée aux éléments. Ainsi, les auteurs définissent le tf-itdf (*Term Frequency - Inverse Tag and Document Frequency*), qui permet de calculer la force discriminatoire d'un terme *t* pour un élément *e* relatif à un document *d*.

D'autres paramètres peuvent aussi être pris en considération dans la pondération des termes. Ainsi, outre le fait de tenir compte de l'élément dans lequel un terme apparait, on peut aussi tenir compte de son importance dans le contenu de l'élément, dans le contenu de ses descendants, dans

le contenu de ses voisins directs et dans le contenu des éléments auquels il est relié [56]. L'agrégation de ces paramètres peut être réalisée à l'aide de différents opérateurs (notamment avec l'operateur OWA [110]).

5.2 Indexation de l'information structurelle

L'indexation structurelle peut se faire selon des granularités variées [65], c'est à dire que toute l'information structurelle n'est pas forcément utilisée dans le processus d'indexation.

On distingue trois types d'approches pour l'indexation de l'information structurelle: l'indexation basée sur des champs, l'indexation basée sur des chemins et l'indexation basée sur des arbres. Ces approches sont décrites dans ce qui suit.

5.2.1 Indexation par les champs

C'est certainement la méthode d'indexation la plus simple pour l'indexation de l'information structurelle. Elle représente le document par un ensemble de balises et de contenu associé à ces balises [41]. La figure suivante illustre la procédure d'indexation par les champs d'un document XML.

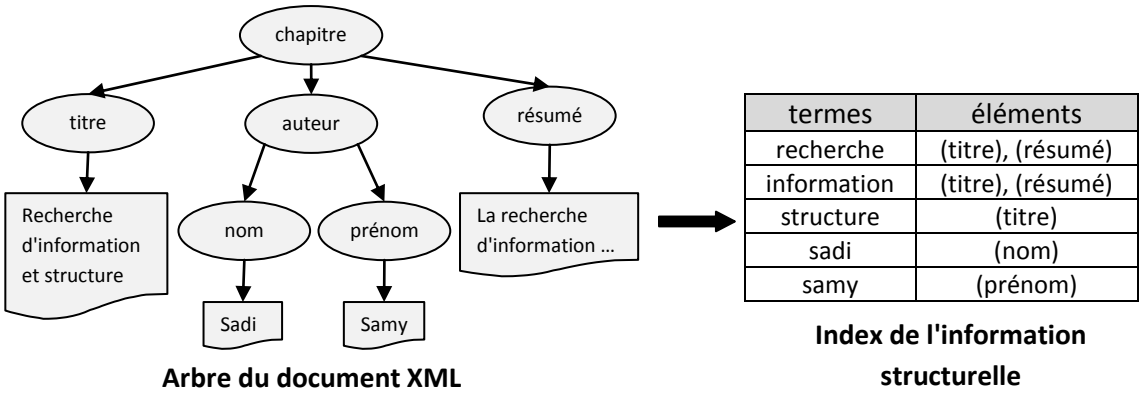


Figure 10: Exemple d'indexation basée sur les champs

5.2.2 Indexation par les chemins

L'indexation par les chemins [107] [46] [54], a été introduite afin de résoudre efficacement les requêtes de type *XPATH*. Elle associe à chaque terme d'indexation le chemin de l'élément où il se trouve (voir la figure 11). Dans ces approches, il est difficile de retrouver les relations de structure comme ancêtres-descendants entre les différents éléments de l'arbre du document. L'indexation basée sur les arbres, quant à elle, le permet.

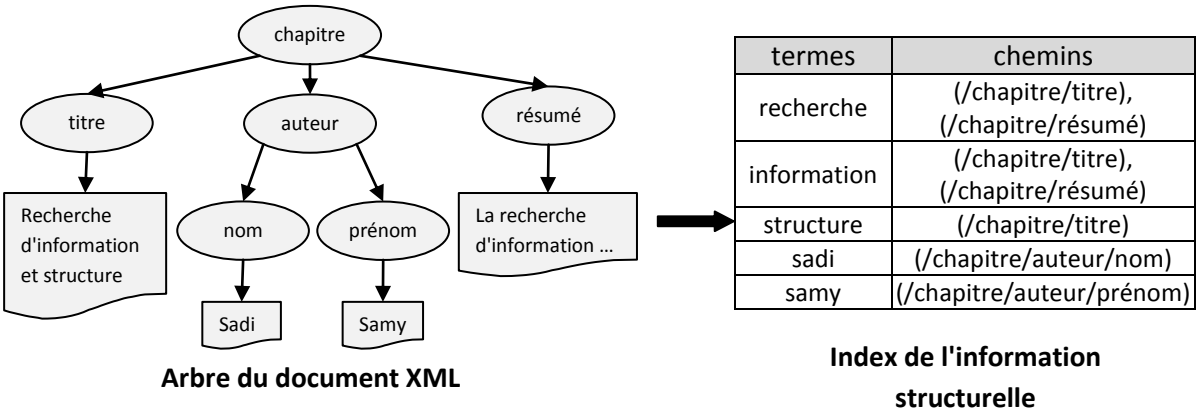


Figure 11: Exemple d'indexation basée sur les chemins

5.2.3 Indexation par les arbres

L'indexation par les arbres [47] [27] [94], ressemble beaucoup à l'indexation par les chemins, mais permet en plus, de facilement retrouver les relations ancêtres-descendants entre les différents éléments des documents indexés. Pour ce faire, un identifiant unique est associé à chaque élément de l'arbre du document XML. Celui-ci peut être attribué pour les éléments selon leur position dans l'arbre du document. La figure suivante illustre cette technique d'indexation.

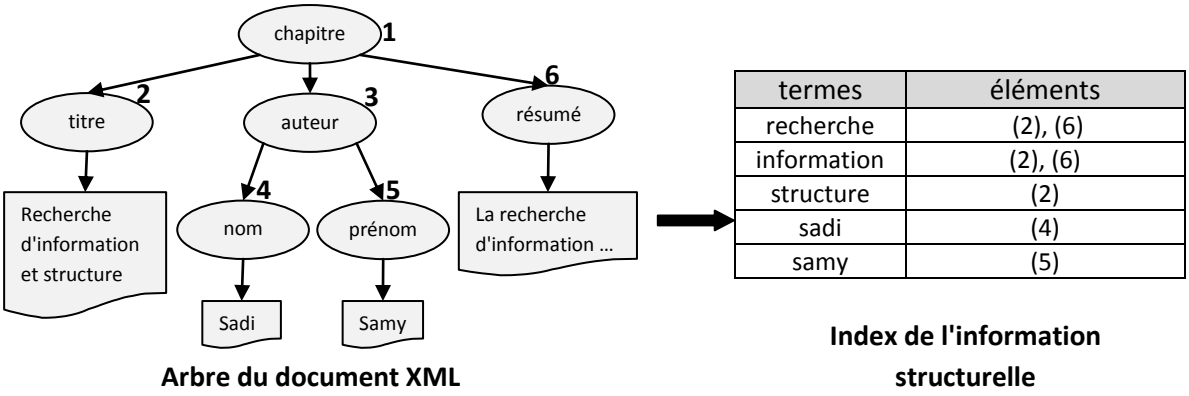


Figure 12: Exemple d'indexation basée sur les arbres

6 Langages de requêtes

Plusieurs langages de requêtes ont été développés afin de pouvoir interroger une collection de documents XML. Bien que ceux-ci différent par le niveau d'expressivité qu'ils offrent à l'utilisateur, ils permettent néanmoins d'intégrer la notion de structure dans la formulation du besoin en informations de l'utilisateur. La figure suivante montre les principaux langages de requêtes utilisés dans la RI structurée. Les langages les plus utilisés sont décrits par la suite.

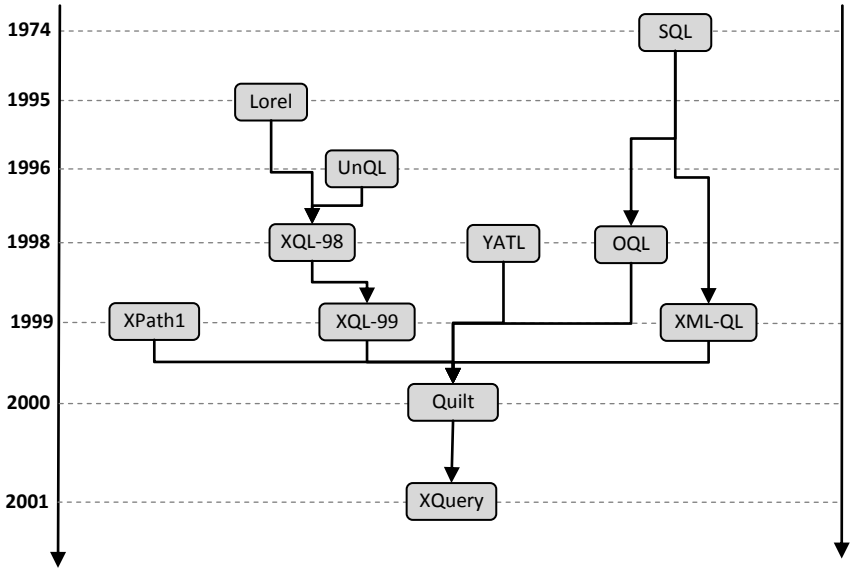


Figure 13: Historique des langages de requêtes sur des documents XML

6.1 XPath

XPath (pour *XML Path Language*) est une recommandation du W3C[10], qui définit un langage de requête (non XML) pour localiser un ou des éléments d'un document XML. Apparu en 1999, ce langage était destiné à fournir une syntaxe et une sémantique aux fonctions communes à *XPointer* et

[10] http://www.w3.org/TR/xpath/, http://www.w3.org/TR/xpath20/

XSL. Il a ensuite été adopté par les développeurs comme langage de requête simple d'emploi pour les documents XML.

Les requêtes XPath se basent sur la représentation sous forme d'arbre du document XML afin d'y sélectionner des éléments en se basant sur différents critères. Une requête XPath est exprimée sous forme d'un *chemin de localisation* lui-même constitué de *pas de localisations* séparés par des slashs « / ». Chaque pas de localisation est composé d'un *axe*, un *type d'élément* et des *prédicats*.

L'axe indique la direction dans laquelle se déplacer dans le document XML. Parmi les axes XPath on trouve:

- « *ancestor* » : tous les éléments ancêtres de l'élément courant (parent de l'élément, parent du parent de l'élément …);
- « *attribut* » : tous les attributs de l'élément courant. Le symbole « @ » peut être utilisé comme abréviation pour cet axe;
- « *child* » : tous les enfants de l'élément courant. Si l'axe n'est pas spécifié, *child* est considéré par défaut;
- « *descendant* » : tous les éléments descendants de l'élément en cours (fils de l'élément, fils du fils de l'élément…);
- « *descendant-or-self* » : tous les éléments descendants de l'élément en cours, plus l'élément en cours. La chaine de caractères « // » est une abréviation pour cet axe;
- « *parent* » : l'élément parent de l'élément courant. Utiliser deux points « .. » revient à utiliser cet axe;
- « *self* » : l'élément courant. Le point « . » est une abréviation pour cet axe.

Le test de l'élément permet de sélectionner des éléments suivant leur nom ou leur type. Il s'agit donc du nom d'un élément ou d'une expression plus générale tel que:

- « *text()* » : pour sélectionner les éléments textes;
- « *node()* » : pour sélectionner tout type d'élément;

Les prédicats sont écrits entre crochets (« [» et «] ») et permettent de filtrer les éléments sélectionnés par l'axe et le test de l'élément. Si la valeur du prédicat est numérique, elle indique la position de l'élément à sélectionner. Par exemple, « titre[5] » va sélectionner le cinquième titre dans l'élément courant. En revanche, si le prédicat n'est pas numérique, il est évalué comme une expression logique qui est testée avec chaque élément, lequel est rejeté s'il ne la satisfait pas. Par exemple, « chapitre[@titre='RI'] » va sélectionner tous les éléments « chapitre » dans l'élément courant, qui ont en plus un attribut « titre » contenant le texte « RI ».

6.2 XQuery

XQuery (pour *XML Query Langage*) est une recommandation du W3C[11], qui définit un langage de requête permettant non seulement d'extraire des informations d'un document XML, ou d'une collection de documents XML, mais également d'effectuer des calculs complexes à partir des informations extraites et de reconstruire de nouveaux documents ou fragments XML. Le « *XML Query working group* », qui a développé XQuery, s'est principalement inspiré de Quilt[12], qui à son tour, a emprunté ses caractéristiques à plusieurs autres langages de requêtes tels que XPath,

[11] http://www.w3.org/TR/xquery/
[12] http://www.almaden.ibm.com/cs/people/chamberlin/quilt.html

XQL [80], XML-QL [64] et OQL [13]. XQuery est apparu pour la première fois en 2001 mais il a fallu attendre 2007 pour qu'il devienne une recommandation officielle du W3C.

Côté syntaxe, XQuery offre deux manières pour formuler les requêtes. Une syntaxe « *naturelle* » non XML, dite aussi *FLWOR* (prononcé *flower*), dont le nom vient des cinq clauses principales qui la composent (*for*, *let*, *where*, *order by* et *return*). Et la syntaxe *XQueryX* dans laquelle une requête est un document XML. Généralement, on préfère la première syntaxe car elle est plus simple et plus lisible contrairement à XQueryX, qui est destinée à des manipulations formelles par des programmes (éventuellement eux-mêmes écrits en XQuery).

Voici en exemple un document XML, une requête XQuery et le résultat qu'elle renvoie.

Le document XML (livres.xml):

```
<?xml version="1.0" encoding="UTF-8" ?>
<livres>
	<livre catégorie="INFORMATIQUE">
		<titre lang="en">Automatic Information Organization and Retrieval</titre>
		<auteur>Gerald Salton</auteur>
		<année>1968</année>
		<prix>34.79</prix>
	</livre>
	<livre catégorie="ENFANT">
		<titre lang="en">Harry Potter</titre>
		<auteur>J K. Rowling</auteur>
		<année>2005</année>
		<prix>29.99</prix>
	</livre>
	<livre catégorie="WEB">
		<titre lang="en">XQuery Kick Start</titre>
		<auteur>James McGovern</auteur>
		<auteur>Per Bothner</auteur>
		<auteur>Kurt Cagle</auteur>
		<auteur>James Linn</auteur>
		<auteur>Vaidyanathan Nagarajan</auteur>
		<année>2003</année>
		<prix>49.99</prix>
	</livre>
	<livre catégorie="ROMAN">
		<titre lang="fr">Germinal</titre>
		<auteur>Emile Zola</auteur>
		<année>1971</année>
		<prix>3.80</prix>
	</livre>
</livres>
```

La requête XQuery:

```
for $l in document("livres.xml")/livres
where $l/prix>30
return $l/titre
```

Le résultat:

```
<titre lang="en">Automatic Information Organization and Retrieval</titre>
<titre lang="en">XQuery Kick Start</titre>
```

7 Modèles de RI structurée

Les modèles de RI structurée sont utilisés tout comme pour la recherche d'information classique au niveau de l'appariement pour sélectionner et retourner les unités d'information pertinentes à l'utilisateur.

Les modèles de RI structurée peuvent être classés en deux catégories: les modèles de RI structurée sans prise en compte des liens, et les modèles de RI structurée pour la prise en compte des liens. Les modèles des deux catégories sont utilisés conjointement pour répondre à une requête utilisateur. Ainsi, un modèle de la première catégorie est d'abord utilisé pour sélectionner les premiers résultats répondant à une requête. Ensuite un modèle de la deuxième catégorie est utilisé pour réordonner les résultats obtenus par le premier modèle (principe du « *re-ranking* ») et éventuellement sélectionner d'autres résultats qui auraient été omis par le premier modèle.

7.1 Les modèle de RI structurée sans prise en compte des liens

Les modèles de recherche présentés dans cette section sont spécifiques aux approches orientées documents, puisqu'ils cherchent à attribuer des scores de pertinence aux éléments des documents XML. Les approches orientées données, pour leur part, ne se posent pas ce problème car le contenu des documents est traité de façon booléenne (présent/absent).

Les modèles présentés ci-dessous ont été adaptés depuis les modèles de RI classique pour tenir compte de l'information structurelle des documents. Ceci est fait par l'ajout de certains paramètres pour ajuster les formules classiques tel que: le nombre d'enfants d'un élément [33], son type [68], la fréquence de ce type d'éléments dans la collection [37] [92] ou alors l'importance d'un terme dans les autres éléments du même type [37].

7.1.1 Extension des modèles booléens

L'extension du modèle booléen [44] se base, pour la prise en compte de la structure, sur les *p-distances* et la propagation des scores depuis les feuilles de l'arbre jusqu'aux éléments. Ce modèle est principalement adapté pour les requêtes orientées contenu, et permet de définir l'unité d'information à renvoyer à l'utilisateur.

Chaque élément feuille (noté e_i) est alors représenté par un vecteur:

$$W(e_i) = (w_{i,1} \dots w_{i,n})$$

Où n représente le nombre de termes d'indexation dans la collection de documents, et $w_{i,j}$ représente le poids du terme t_j dans l'élément feuille e_i qui est donné par:

$$w_{i,j} = \frac{tf(t_j, e_i)}{\sum_{k=1}^{m}\sum_{l=1}^{n} tf(t_l, e_k)} \times \log\frac{p}{df(t_j)}$$

Où:

- m est le nombre d'éléments feuille dans le document;
- n est le nombre de termes d'indexation dans la collection de documents;
- p est le nombre de documents dans la collection de documents;
- $tf(t_j, e_i)$ représente la fréquence du terme t_j dans l'élément e_i;
- $df(t_j)$ représente le nombre de documents contenant le terme t_j.

Le score de similarité entre une requête q et un élément e est alors calculé de la manière suivante:

- Si e est un élément feuille:

$$rsv(e,q) = \frac{\mathrm{W(e)} \cdot \mathrm{q}}{|\mathrm{W(e)}| \times |\mathrm{q}|}$$

- Si e n'est pas un élément feuille, son score de similarité est calculé d'une manière récursive en partant des éléments feuilles:

$$rsv(e,q) = 1 - \sqrt[\frac{1}{r}]{\frac{1}{|enf(e)|} \times \sum_{e' \in enf(e)} \left(1 - rsv(e',q)\right)^r}$$

Où:

- r est généralement choisi avec une valeur égale à 2;
- $enf(e)$ représente l'ensemble des sous éléments de e;
- q représente le vecteur de la requête dont les composantes q_j (avec $1 \leq j \leq n$) sont égales à 1 si le terme t_j apparait dans la requête, et 0 sinon.

7.1.2 Extension des modèles vectoriels

Le modèle vectoriel a connu plusieurs adaptations [3] [23] [33] [68] [101] [104] [50]. Toutes ces adaptations calculent un score de similarité entre un élément et une requête qui sont alors représentés sous forme de vecteurs.

L'une des premières adaptations du modèle vectoriel [33], considère que chaque élément e est défini par un type T reflétant son emplacement dans l'arbre du document XML. La mesure de similarité d'un élément e à une requête $q = \{t_1 \dots t_k\}$ est calculée de la manière suivante:

$$rsv(e,q) = \alpha(T) \times cosm(q,e) + \sum_{i=1}^{|enf(e)|} \frac{cosm(q,e_i)}{\beta^{i-1}}$$

Où:

- $\alpha(T)$ est un facteur permettant de prendre en compte le type de l'élément;
- $|enf(e)|$ est le nombre d'éléments enfants de l'élément e;
- e_i est le $i^{éme}$ enfant de e;
- β est un paramètre qui permet d'assurer que le nombre d'enfants n'introduit pas un biais dans la formule;
- $cosm(q,e) = \sum_{i=1}^{k} \frac{w_{q,i} \times w_{e,i}}{|e|}$ où $|e|$ est le nombre de termes dans l'élément e. Et où $w_{q,i}$ et $w_{e,i}$ sont respectivement le poids du terme t_i dans la requête q et dans l'élément e.

Une autre adaptation utilisée dans le système *JuruXML* [68] propose d'indexer les éléments selon leur type (un index par type d'éléments) et d'appliquer ensuite le modèle vectoriel pour la pondération des éléments. Elle se base sur la décomposition de la requête et des éléments en un ensemble de conditions ($c_{q,i}$ et $c_{e,i}$). Une condition étant le chemin où apparait le terme. Ensuite, une correspondance entre les conditions est estimée de la façon suivante:

$$cr(c_{q,i}, c_{e,k}) = \frac{1 + |c_i|}{1 + |c_k|}$$

Où $|c_i|$ et $|c_k|$ sont respectivement le nombre de balises dans un contexte donné de la requête q et de l'élément e.

Le score de similarité entre une requête et un élément est alors calculé selon la formule suivante:

$$rsv(e,q) = \frac{\sum_{(t,c_{q,i})\in q}\sum_{(t,c_{e,i})\in e} w_{q,t} \times w_{e,t} \times cr(c_{q,i}, c_{e,i})}{|e| \times |q|}$$

Où:

- $w_{q,t}$ et $w_{e,t}$ représente respectivement, le poids du terme t dans la requête q et l'élément e;
- $|e|$ et $|q|$ sont les nombres de termes dans l'élément et la requête.

7.1.3 Extension des modèles probabilistes

Deux approches ont été principalement utilisées pour adapter les modèles probabilistes à la RI structurée. La première approche consiste à utiliser des probabilités conditionnelles de jointure sur les chemins du document, ainsi $P(d|t)$ devient $P(d|p\ contains\ t)$ où d représente un document ou une partie du document, t est un terme et p est un chemin dans l'arbre du document d. La deuxième permet d'étendre la logique propositionnelle à la logique des prédicats et prend en compte les problèmes liés à la structure [7].

Parmi les travaux qui ont adapté le modèle probabiliste pour la RI structurée, citons les travaux de *Fuhr & al.* [32] et *Gövert & al.* [36]. Ceux-ci ont proposé un modèle probabiliste inférentiel basé sur une méthode d'augmentation (multiplication par un facteur donné). Ce modèle a été implémenté sur le système *HyRex* et utilise le langage de requêtes *XIRQ*.

Dans ce modèle, les éléments sont considérés comme des unités disjointes. Les éléments feuilles ne sont pas indexés car ont une granularité trop fine. Ainsi donc, les termes associés aux éléments feuilles sont propagés dans l'arbre du document jusqu'à l'élément le plus proche.

Par ailleurs, il y eu d'autres travaux qui ont adapté le modèle de langue [2] [4] [106], et les réseaux bayésiens [74] [73] à la RI structurée.

7.2 Les modèle de RI pour la prise en compte des liens

Plusieurs modèles ont été développés pour l'exploitation des liens dans la recherche d'information. Historiquement, les premiers modèles ont été créés pour le Web. Ces modèles considèrent le document comme une unité d'information indivisible et s'intéressent uniquement aux liens document-document. Par la suite, d'autres modèles ont été proposés pour exploiter les liens élément-élément qui peuvent apparaitre dans les documents XML grâce à XLink.

Bien que les modèles des deux catégories peuvent être utilisés pour la prise en compte des liens dans la RI structurée. Les modèles de la deuxième catégorie sont mieux adaptés lorsque l'unité d'information pertinente recherchée par l'utilisateur est l'élément.

7.2.1 Modèles pour la prise en compte des liens dans le Web

Les modèles de cette catégorie se basent sur les liens document-document. Ce type de liens est le plus répandu sur le web.

Ces modèles se classent principalement en deux catégories: les modèles basés sur la propagation de popularité et les modèles basés sur la propagation de pertinence. Dans cette section, nous présentons les modèles les plus connus de ces deux catégories.

7.2.1.1 Modèles basés sur la propagation de popularité

Les modèles basés sur la propagation de popularité s'appuient sur l'hypothèse qu'un document référencé par un grand nombre de documents est un bon document. Ainsi donc, une valeur de popularité est calculée et assignée aux documents suivant le nombre de documents où ils sont cités (nombre de liens entrants) et le nombre de documents qu'ils citent (nombre de liens sortants).

La valeur de popularité d'un document est calculée soit en se basant sur les liens de tous les documents de la collection (InDegree et PageRank), soit en se basant sur les liens des documents jugés pertinents à la requête (Hits et Salsa). Dans le premier cas, la valeur de popularité est calculée une seule fois après l'indexation, et ce, pour tous les documents de la collection. Ainsi, il n'est plus nécessaire de la recalculer à chaque requête. Dans le deuxième cas, la valeur de popularité d'un document est calculée à chaque requête.

Plusieurs algorithmes ont été proposés pour les deux méthodes de calcul de popularité. Dans ce qui suit, nous présentons les plus connus.

1) InDegree

InDegree est surement le plus simple des algorithmes exploitant les liens. C'est aussi l'un des premiers algorithmes à être utilisé pour le web. Il se base sur une formule simple pour calculer la popularité d'un document:

$$InDegree(d) = \frac{|E(d)|}{n}$$

Où:

- d est le document dont on veut calculer la popularité;
- $E(d)$ est l'ensemble des documents contenant un lien vers le document d, et $|E(d)|$ la cardinalité de cet ensemble;
- n est le nombre total de documents dans la collection.

InDegree est un modèle très simple à mettre en œuvre. Cependant, c'est un modèle qui est peu efficace notamment si la collection de documents est issue du web, où le nombre de liens ne peut être tenu comme seul facteur pour déterminer la popularité d'une page. En effet, on peut intentionnellement créer des pages web fictives contenant des liens pointant vers une page web particulière afin d'augmenter sa popularité. De plus, InDegree ne tient pas compte de l'importance des liens, en considérant tous les liens issus de différents documents comme ayant une importance égale. Ce qui n'est pas le cas en pratique, car un document sera plus populaire s'il est cité par un document très pertinent que s'il est cité par un document moins pertinent.

2) PageRank

Le *PageRank* [12] est un algorithme qui permet de calculer la popularité d'un document en se basant non seulement sur le nombre de documents qui le citent, mais encore sur la popularité de ces documents. Ainsi, contrairement à InDegree présenté précédemment, le PageRank considère que les liens sont d'une importance variée qui dépend de la popularité du document où ils apparaissent.

La formule utilisée pour calculer le PageRank est la suivante:

$$PR(d) = (1 - c) + c \times \sum_{d_i \in E(d)} \frac{PR(d_i)}{|S(d_i)|}$$

Où:

- $PR(d)$ est le PageRank du document d;
- c est un facteur d'amortissement choisi entre 0 et 1;
- $E(d)$ est l'ensemble des documents contenant un lien vers le document d;
- d_i est un document qui contient un lien vers le document d;
- $S(d_i)$ est l'ensemble des documents cités par le document d_i;
- $|S(d_i)|$ est le nombre de documents cités par le document d_i. Autrement dit, le nombre de liens sortants du document d_i;
- $PR(d_i)$ est le PageRank du document d_i.

Pour illustrer ce que donnerait l'utilisation du PageRank sur une petite collection de document, la figure suivante est donnée en exemple.

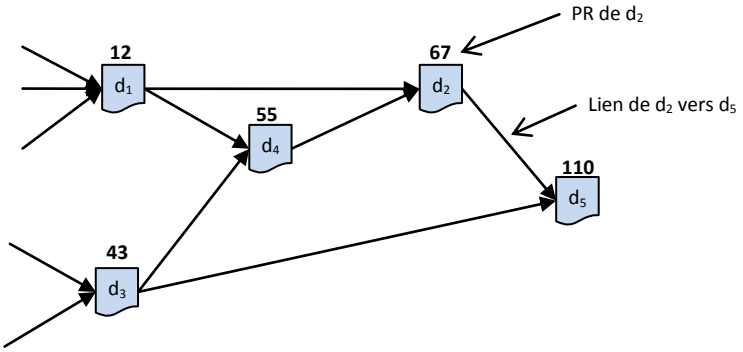


Figure 14: Exemple d'utilisation de l'algorithme de PageRank

La détermination de la valeur de PageRank pour un document donné n'est pas un calcul simple car il dépend de la connaissance du PageRank de tous les documents le citant, qui à leur tour dépendent de la connaissance du PageRank des documents les citant et ainsi de suite.

En pratique, la valeur du PageRank d'un document est calculée de façon itérative. Initialement, des valeurs de PageRank équiprobables sont assignées aux documents, par exemple $\frac{1}{n}$ où n représente le nombre de documents dans la collection. Ensuite, à chaque itération, les valeurs de PageRank sont propagées aux autres documents grâce à la formule donnée précédemment. Le nombre d'itérations dépend du nombre de documents dans la collection. Généralement, l'algorithme s'arrête quand les itérations ne produisent plus de modifications significatives du PageRank, ce qui se produit après une dizaine d'itérations.

L'avantage du PageRank est que le calcul des valeurs de popularité des documents n'est fait qu'une seule fois après l'indexation. Ainsi, on évite de refaire le calcul pour chaque requête au niveau de l'interrogation. Cependant, ceci signifie qu'un document populaire, le sera pour toutes les requêtes. Or, un document peut être populaire et servir de référence pour une requête portant sur un domaine particulier, et ne pas l'être pour une autre requête. Un autre problème lié à l'utilisation du PageRank se pose lorsque la collection de documents comporte des liens circulaires. Par lien circulaire, on entend tout lien qui pointe vers un document qui, lui-même, contient un lien qui pointe

vers le document qui contient le premier lien. Ce problème fait que le PageRank des documents concernés (contenant des liens circulaires) augmente à chaque itération, sans converger vers une valeur fixe. Une solution à ce problème a été proposée grâce à l'utilisation du principe du surfer aléatoire [14] (« *Random Walk* »). D'autres inconvénients peuvent aussi apparaitre lorsque le PageRank est utilisé pour le web, notamment les problèmes liés au spamming qui consiste à créer des pages fictives pointant vers une page particulière pour augmenter son PageRank.

Vu le succès que connait PageRank, particulièrement pour le web, plusieurs auteurs ont proposé des approches similaires. On citera notamment le PageRank modulaire [48] (« *Modular PageRank* »), le Page Rank calculé par blocs [53] (« *BlockRank* ») et le PageRank sensible à la thématique [45] (« *Topic sensitive PageRank* »).

3) Hits

HITS [57] (« *Hypertext Induced Topic Search* »), propose de calculer la popularité d'un document en se basant uniquement sur les liens des documents répondant à la requête.

Pour ce faire, HITS classe les documents restitués suite à une requête en deux classes: les documents pivots et les documents autorités. Les documents pivots sont des documents qui contiennent un grand nombre de liens vers d'autres documents, on dit aussi qu'ils ont un grand pouvoir *rayonnant*. Et les documents autorités sont des documents qui sont cités par un grand nombre de documents.

Il existe une relation circulaire entre le pouvoir rayonnant et le degré d'autorité des documents [16]: un document fait d'autant plus autorité qu'il est cité par des documents rayonnants et réciproquement, un document est d'autant plus rayonnant qu'il cite des documents autorités.

L'algorithme HITS propose une méthode itérative pour calculer le degré de rayonnement et d'autorité d'un document. Initialement, il associe à chaque document d un poids autorité x_d et un poids de rayonnement y_d initialisés à 1. Les poids sont ensuite mis à jour de la façon suivante: pour chaque document d, la valeur de son poids x_d est mise à jour par la somme des poids y_d de tous les documents d_i pointant vers le document d. D'une manière symétrique, le poids de rayonnement d'un document est mis à jour en fonction des poids autorités des documents vers lesquels ce document pointe. Les formules suivantes résument ces opérations:

$$x_d = \sum_{d_i \in E(d)} y_{d_i}$$

$$y_d = \sum_{d_i \in S(d)} x_{d_i}$$

Où:

- x_d et y_d sont respectivement le poids d'autorité et de rayonnement du document d;
- x_{d_i} et y_{d_i} sont respectivement le poids d'autorité et de rayonnement du document d_i;
- $E(d)$ est l'ensemble des documents contenant un lien vers le document d;
- $S(d)$ est l'ensemble des documents cités par le document d.

Les opérations de mise à jour sont répétées jusqu'à la convergence des valeurs des poids de rayonnement et des poids autorités. Une normalisation sur les poids est aussi réalisée, à chaque

itération, de sorte que la somme des poids de rayonnement, et la somme des poids autorités soient égales à 1.

Un des avantages de HITS est qu'il permet le classement des documents selon deux critères: le degré de rayonnement et le degré d'autorité qui est généralement retenu pour le classement final des documents. Côté inconvénients, HITS nécessite un temps de calcul supplémentaire au niveau de l'interrogation, car HITS dépend de la requête et les poids de rayonnement et d'autorité ne sont calculés qu'après la requête. Aussi, tout comme l'algorithme de PageRank, HITS est vulnérable aux techniques de spamming de liens lorsqu'il est utilisé pour le web.

Plusieurs travaux visant à améliorer les performances de HITS ont été proposés. On citera notamment, l'algorithme statistique PHITS [19] et l'algorithme TOPHITS [58].

4) Salsa

SALSA [63] (« *Stochastic Approach for Link-Structure Analysis* »), est un algorithme qui combine les techniques de HITS et de PageRank pour l'analyse des liens.

Tout comme pour HITS les documents répondant à une requête sont classés en deux types de documents: les documents pivots et les documents autorités. L'ensemble de ces documents forme alors un graphe biparti (une partie pour les documents pivots et une autre pour les documents autorités) où les documents sont les éléments du graphe et les liens représentent les arcs du graphe. SALSA utilise ensuite le principe de parcours aléatoire de PageRank [14] sur le graphe en alternant entre les cotés pivots et autorités du graphe.

L'algorithme commence par un document autorité. Ensuite, il choisit aléatoirement un lien entrant parmi tous les liens qui pointent le document autorité pour choisir un document pivot. Une fois le document pivot sélectionné, l'algorithme choisit un lien sortant aléatoirement pour sélectionner un nouveau document autorité. Ce parcours aléatoire se poursuit en alternant entre les documents pivots et autorités. Lors de ce parcours, le poids d'autorité x_d et de rayonnement $y_d$de chaque document d, sont mis à jour de la façon suivante:

$$x_d = \sum_{d_i \in E(d)} \frac{y_{d_i}}{|S(d_i)|}$$

$$y_d = \sum_{d_i \in S(d)} \frac{x_{d_i}}{|E(d_i)|}$$

Où:

- x_d et y_d sont respectivement le poids d'autorité et de rayonnement du document d;
- x_{d_i} et y_{d_i} sont respectivement le poids d'autorité et de rayonnement du document d_i;
- $E(d)$ est l'ensemble des documents contenant un lien vers le document d;
- $S(d)$ est l'ensemble des documents cités par le document d;
- $|E(d_i)|$ est le nombre de documents contenant un lien vers le document d_i;
- $|S(d_i)|$ est le nombre de documents cités par le document d_i.

Tout comme pour HITS, SALSA réalise un double classement des documents: selon le degré de rayonnement et selon le degré d'autorité. Ce dernier sera généralement retenu pour le classement final des documents. De plus, SALSA offre de meilleurs résultats quant au classement des documents comparé à HITS. Par ailleurs, SALSA est moins vulnérable au spamming de liens. Coté inconvénients,

on cite principalement la lenteur au niveau de l'interrogation qu'implique l'utilisation de SALSA car le classement des documents est réalisé après la requête.

7.2.1.2 Modèles basés sur la propagation de pertinence

Les modèles présentés dans la section précédente classent les documents restitués à l'utilisateur suivant la structure des liens dans la collection de documents. À aucun moment, la pertinence des documents n'est prise en compte. Les modèles basés sur la propagation de pertinence, se basent sur l'hypothèse que « *un document référencé par un grand nombre de documents pertinents à la requête utilisateur est un bon document* ». Ainsi, la pertinence des documents est améliorée suivant le nombre et la pertinence des documents voisins qu'ils possèdent (documents qu'ils citent ou qui les citent). Par ailleurs, les modèles basés sur la propagation de pertinence s'appliquent au moment de l'interrogation, car la pertinence d'un document ne peut être évaluée que par rapport à une requête. Dans ce qui suit, nous présentons un des premiers modèles de propagation de pertinence qui est l'activation propagée.

1) Activation propagée

L'activation propagée [22] [91] est un modèle qui se base, pour modifier la pertinence d'un document, sur la propagation d'une fraction du score de pertinence de ses documents voisins. Ainsi, pour chaque document d, une valeur de pertinence initiale (notée $RSV_0(d)$) est d'abord calculée grâce aux modèles de RI classiques ou structurés, ensuite cette valeur est modifiée en tenant compte de la valeur de pertinence des documents voisins selon la formule suivante:

$$RSV_k(d) = RSV_0(d) + \lambda \cdot \sum_{d_i \in E(d) \cup S(d)} RSV_{k-1}(d_i)$$

Où:

- $RSV_0(d)$ est la pertinence initiale du document d;
- λ est un facteur pour atténuer la propagation (généralement compris entre *0,1* et *0,2*);
- $E(d)$ est l'ensemble des documents contenant un lien vers le document d;
- $S(d)$ est l'ensemble des documents cités par le document d;
- d_i est un document voisin de d;
- $RSV_{k-1}(d_i)$ est la pertinence du document d_i calculée après $k-1$ itérations;
- $RSV_k(d)$ est la pertinence du document d calculée après k itérations.

Cette formule est appliquée itérativement jusqu'à stabilisation des valeurs de pertinences. Le score de pertinence final d'un document est une combinaison de son score reposant sur le contenu seul et de la propagation des fractions des scores de ces documents voisins.

D'autres travaux portant sur la propagation de pertinence ont été proposés, on citera notamment ceux de *Marchiori* [66], de *Frei & al* [31] et le modèle générique de propagation de pertinence proposé par *Shakery & al* [93].

7.2.2 Modèles pour la prise en compte des liens dans la RI structurée

Les modèles de cette section s'intéressent aux liens élément-élément dans les documents XML. Ce type de liens est ajouté grâce à XLink.

Contrairement aux modèles de la catégorie précédente, la popularité de deux éléments dans un même document XML peut être différente. De plus, ce n'est pas une liste de documents qui est réordonnée mais une liste d'éléments.

Peu de travaux ont été proposés dans ce sens, nous présentons quelque uns dans ce qui suit.

1) XRank

XRank [40] est le premier modèle qui propose d'intégrer la hiérarchie des éléments et les liens entre ces éléments dans des documents XML lors de l'appariement. Il permet de calculer le score d'un élément de façon assez similaire à PageRank, sauf que l'unité d'information pertinente considérée est l'élément.

Le score XRank d'un élément est calculé avec la formule suivante:

$$e(v) = \frac{1 - d_1 - d_2 - d_3}{N_d \times N_{de}(v)} + d_1 \sum_{(u,v)\in HE} \frac{e(u)}{N_h(u)} + d_2 \sum_{(u,v)\in CE} \frac{e(u)}{N_c(u)} + d_3 \sum_{(u,v)\in CE^{-1}} e(u)$$

Où:

- $e(v)$ est le score XRank de l'élément v;
- d_1, d_2 et d_3 sont des paramètres réels compris entre 0 et 1. Dans [40] ils ont été fixés respectivement à *0.35*, *0.25* et *0.25*;
- N_d est le nombre de documents dans la collection;
- $N_{de}(v)$ est le nombre d'éléments dans le document XML qui contient l'élément v;
- HE est l'ensemble des liens sortants entre les éléments. Si $(u, v) \in HE$, alors l'élément u contient un lien vers l'élément v;
- $e(u)$ est le score XRank de l'élément u;
- $N_h(u)$ est le nombre de liens sortants depuis l'élément u;
- CE est l'ensemble de liens hiérarchiques (ancêtre, descendant) entre les éléments. Si $(u, v) \in CE$, alors v est un élément contenu dans l'élément u (c'est-à-dire, u est l'ancêtre de l'élément v);
- $N_c(u)$ est le nombre d'éléments contenus dans l'élément u;
- CE^{-1} est l'ensemble construit à partir de CE qui contient les liens descendant-ancêtre entre les éléments. Si $(u, v) \in CE^{-1}$, alors u est un élément contenu dans l'élément v (c'est-à-dire, u est un descendant de l'élément v).

2) DocRank

DocRank [69] est une autre adaptation de PageRank aux collections de documents XML. Les auteurs proposent d'assigner un score à un élément en se basant sur le score de popularité du document qui le contient.

Le score de popularité du document est calculé au moment de l'indexation de la manière suivante:

$$DOCRANK(D) = \frac{1 - d}{NbrDocsCol\ l} + d \sum_{(i,D)\in links} DOCRANK(i)$$

Où:

- $DOCRANK(D)$ est le score de popularité (DocRank) du document D;
- d est un facteur d'amortissement;
- $NbrDocsColl$ est le nombre de documents dans la collection;

- $links$ représente l'ensemble des paires de liens (i, j) internes à la collection tel que le document i contient un lien vers le document j.

Une fois le DocRank de tous les documents de la collection calculé, le score d'un élément est calculé au moment de la requête avec la formule suivante:

$$NouvScoreDOCRANK(E_i) = \alpha \times ScoreInitial(E_i) + (1 - \alpha) \times DOCRANK(D)$$

Où:

- $NouvScoreDOCRANK(E_i)$ est le score de l'élément E_i;
- D est le document qui contient l'élément E_i ($E_i \in D$);
- $DOCRANK(D)$ est le score de popularité du document D;
- α est un paramètre qui permet de définir le degré de contribution des différents scores dans le score final;
- $ScoreInitial(E_i)$ représente le score initialement affecté par le système de recherche à l'élément E_i.

Les expérimentations effectuées dans [69] ont montré des améliorations de la qualité des résultats suite à l'utilisation de DocRank.

8 Évaluation des systèmes de RI structurée

L'évaluation des systèmes de RI structurée se base généralement, comme pour l'évaluation des systèmes de RI classique, sur les mesures de rappel et de précision. Toutefois, elle doit en plus pouvoir intégrer les notions d'exhaustivité et de spécificité, car l'unité d'information retournée à l'utilisateur ne s'agit plus de documents mais d'éléments extraits depuis ces documents.

Actuellement, INEX[13] (INiative for the Evaluation of XML retirieval) est considérée comme la seule compagne d'évaluation pour les systèmes de RI structurée. Elle a été tenue pour la première fois en 2002, et a lieu chaque année depuis. Lors de l'évaluation, elle fournit aux participants une collection de tests pour faire un classement de leurs systèmes.

Les collections de tests de INEX se composent de trois parties: un ensemble de documents XML, un ensemble de besoins en informations exprimés sous forme de requêtes (*topics*) et un ensemble de jugements de pertinence.

8.1 La collection de documents

La collection de documents offerte par la première compagne d'évaluation INEX contenait 12107 articles publiés de 1995 à 2002, elle avait une taille de 500 Mo. Celle-ci n'a pas cessé d'être enrichie depuis, pour atteindre en 2005 une taille de 750 Mo avec 16819 articles. Un article est alors composé en moyenne d'environ 1500 éléments.

À partir de 2006, la collection de tests commence à être enrichie par des articles issus de l'encyclopédie libre *Wikipédia.* Ainsi, en 2012, la taille de la collection de tests offerte par INEX a atteint la taille de 60 Go avec plus de deux millions et demi[14] d'articles. Cette nouvelle collection, est alors utilisée pour la plupart des taches prises en charge par INEX, incluant la tache principale *AdHoc*.

[13] https://inex.mmci.uni-saarland.de/
[14] http://www.mpi-inf.mpg.de/~gurajada/wikipedia-lod-2012/index.html

8.2 Les requêtes (topics)

Les requêtes sont créées par les différents participants et sont censées être représentatives des besoins en informations les plus communs des utilisateurs. Dans la compagne d'évaluation INEX on distingue deux principales catégories de requêtes:

- *Les CO (content only)*: ce sont des requêtes qui portent sur le contenu textuel des documents uniquement (sans contraintes structurelles). Elles sont généralement exprimées en langage naturel, avec des mots clés éventuellement enrichis d'opérateurs (le « + » pour indiquer qu'un mot est obligatoire, et le « - » pour indiquer que le mot ne doit pas apparaitre dans les résultats);
- *Les CAS (Content And Structure)*: ce sont des requêtes qui contiennent des contraintes structurelles.

Les *topics* sont soumis aux participants à l'évaluation sous forme de fichiers XML qui contiennent, entre autres, la définition simplifiée du *topic* dans l'élément « *title* » et sa description détaillé en langage naturel dans les éléments « *description* » et « *narrative* ».

8.3 Les jugements de pertinence

Les jugements de pertinence permettent d'évaluer les résultats retournés par les SRI participant à l'évaluation. INEX met à disposition des participants un système de jugement en ligne qui se base sur trois critères: le degré de pertinence, l'exhaustivité et la spécificité des résultats. Ces deux dernières sont mesurées, jusqu'en 2004 sur une échelle à 4 niveaux:

- pas exhaustif / pas spécifique;
- marginalement exhaustif / marginalement spécifique;
- assez exhaustif / assez spécifique;
- très exhaustif / très spécifique;

Après 2005, la spécificité est mesuré sur une plage de mesure réelle allant de 0 à 1, où 1 indique un élément totalement spécifique. L'exhaustivité, quant à elle, est toujours mesurée sur une échelle de 4 niveaux, mais avec une signification différente de la précédente: 2 pour une exhaustivité élevée, 1 pour une exhaustivité moyenne, 0 pour pas d'exhaustivité, et une valeur indéfinie si l'élément est trop petit.

Par ailleurs, la mesure de rappel et de précision ont été utilisée principalement jusqu'au 2004. Après quoi, d'autres mesures ont été définies pour permettre une évaluation plus appropriée des performances des systèmes de recherche en RI structurée [55] [61]. Depuis 2007, les mesures officielles sont basées sur l'interpolation de rappel/précision sur 101 niveaux [52].

9 Conclusion

Dans ce chapitre nous avons présenté la RI structurée, notamment dans des documents XML. Nous avons vu que la prise en compte de la structure dans le processus de recherche d'information engendrait de nouvelles problématiques au niveau de la granularité de l'information à retourner à l'utilisateur, au niveau l'expression du besoin en informations de l'utilisateur et au niveau de la prise en compte des liens de citations et de références. Ensuite, nous nous sommes intéressés aux processus d'indexation et d'appariement dans la RI structurée en présentant quelques langages de requêtes et les principaux modèles de recherche. Pour finir, nous avons présenté la compagne d'évaluation INEX, qui est la compagne de référence pour l'évaluation des systèmes de RI structurée.

Dans le chapitre suivant, nous présentons notre modèle basé sur la propagation des scores pour la prise en compte des liens dans la RI.

Chapitre 3: Un modèle de RI pour la prise en compte des liens

1 Introduction

Dans l'état de l'art nous avons vu que la plupart des modèles existants pour la prise en compte des liens dans la RI se limitent à l'exploitation du nombre de liens (HITS et SALSA) et à la propagation des scores des documents (Activation propagée). Nous proposons un nouveau modèle de RI semi-structurée qui exploite en plus, le texte ancre des liens et la balise titre des documents pour améliorer la qualité de réponse du système. Nous nous sommes inspirés pour cela d'une part de l'activation propagée, et d'autre part des travaux de *Fellag-Berchiche* [26].

Dans ce chapitre nous présentons en détail notre modèle en indiquant en premier lieu les dispositions à prendre au niveau de l'indexation, puis en présentant la fonction de correspondance qui permet de réordonner les résultats retournés par la première recherche.

2 Indexation

Pour mettre en œuvre notre modèle il est nécessaire d'indexer au préalable:

- Les termes contenus dans le titre des documents;
- Les liens sortants;
- Les liens entrants;
- Les termes contenus dans le texte ancre des liens.

Ces démarches permettent, à tout moment, de pouvoir retrouver pour un document particulier: son titre, les documents qu'il référence, les documents qui le référencent et également le texte ancre de chaque lien de référence. Ces informations sont utilisées par la suite dans la fonction de correspondance. Plusieurs intuitions motivent l'exploitation de ces informations et sont exposées plus loin dans ce chapitre.

Dans ce qui suit, nous présentons l'indexation des éléments précédemment cités.

2.1 Indexation des titres

L'indexation des titres est réalisée en même temps que l'indexation de l'information structurelle en utilisant indifféremment l'une des techniques d'indexation citées dans l'état de l'art (indexation par les champs, par les chemins ou par les arbres).

2.2 Indexation des liens sortants

L'indexation des liens sortants consiste à extraire depuis chaque document cible, l'ensemble des documents qu'il référence. On obtient comme résultat un nouvel index dont la structure est décrite dans la figure 15.

document	documents référencés
1	2
2	1, 3, 5
3	4
4	1, 2
5	3

Figure 15: Exemple d'un index des liens sortants

2.3 Indexation des liens entrants

L'indexation des liens entrants consiste à trouver pour chaque document l'ensemble des documents qui le référencent. Pour ce faire, on se base sur l'index des liens sortants qui doit être construit en premier lieu. Pour chaque document, il suffit donc de parcourir cet index et de retrouver les documents qui le référencent.

La figure 16 montre l'index des liens entrants construit à partir de l'index des liens sortants donné dans la figure 15.

document	documents qui le référencent
1	2, 4
2	1, 4
3	2, 5
4	3
5	2

Figure 16: Exemple d'un index des liens entrants

2.4 Indexation du texte ancre des liens

L'indexation du texte ancre des liens consiste à attacher pour chaque lien, l'ensemble des termes qu'il contient. Pour ce faire, on modifie la structure de l'index des liens entrants et sortants pour ajouter un pointeur vers un nouvel index contenant le texte ancre des liens. La figure 17 illustre ces propos.

document	documents référencés
1	2(**P1**)
2	1(**P2**), 3(**P3**), 5(**P4**)
3	4(**P5**)
4	1(**P6**), 2(**P7**)
5	3(**P8**)

Index des liens sortants

document	documents qui le référencent
1	2(**P2**), 4(**P6**)
2	1(**P1**), 4(**P7**)
3	2(**P3**), 5(**P8**)
4	3(**P5**)
5	2(**P4**)

Index des liens entrants

Pointeur	Texte ancre des liens
P1	Recherche d'information
P2	xml
P3	Propagation de score
P4	lemmatisation
P5	Activation propagée
P6	xlink
P7	indexation
P8	PageRank

Index du texte ancre des liens

Figure 17: Exemple d'un index des liens sortants et entrants avec leur texte ancre

3 Fonction de correspondance

La fonction de correspondance que nous proposons permet de reclasser les documents restitués après une première recherche basée sur un modèle de RI sans prise en compte des liens. C'est une fonction itérative où le score d'un document à l'itération k est calculé à partir de son score initial (fourni par la première recherche), du score de son titre et du score à l'itération $k-1$ des documents voisins qu'il référence et qui le référencent (liens entrants et sortants).

La fonction de correspondance que nous proposons est définie comme suit:

$$Score_k(d) = (1-\lambda)\cdot Score_0(d) + \lambda\cdot\left(\alpha\cdot ScoreT(d) + \beta\cdot ScoreLE_{k-1}(d) + \gamma\cdot ScoreLS_{k-1}(d)\right)$$

Où:

- $Score_k(d)$ est le score calculé pour le document d à l'itération k;
- $Score_0(d)$ est le score initial du document d, calculé lors de la première recherche, qui doit être normalisé entre 0 et 1;
- $ScoreT(d)$ est le score du titre du document d par rapport à la requête, celui-ci est défini par la suite;
- $ScoreLE_{k-1}(d)$ est le score des liens entrants au document d calculé en se basant sur les scores à l'itération $k-1$ des documents voisins, celui-ci est défini par la suite;
- $ScoreLS_{k-1}(d)$ est le score des liens sortants au document d calculé en se basant sur les scores à l'itération $k-1$ des documents voisins, celui-ci est défini par la suite;
- λ est un paramètre (compris entre 0 et 1) qui est utilisé pour atténuer la propagation des scores lors des itérations;
- α, β et γ sont des paramètres utilisés pour donner une importance particulière aux scores du titre, des liens entrants et des liens sortants. La somme de ces paramètres doit être égale à 1.

3.1 Le score du titre du document

Le titre du document est un élément caractéristique du contenu du document. Ainsi, un document qui contient les termes de la requête dans son titre devrait être plus pertinent qu'un document qui contient les termes de la requête autre part dans son contenu.

Dans le cadre de notre approche, le score du titre du document est calculé comme suit:

$$ScoreT(d) = \frac{\sum_{t\in titre(d)\cap q} tf(t)}{\sum_{t\in titre(d)} tf(t)}$$

Où:

- $ScoreT(d)$ est le score du titre du document d par rapport à la requête q. Il est compris entre 0 et 1;
- $titre(d)$ est l'ensemble des termes contenus dans le titre du document d;
- q est la requête utilisateur;
- $tf(t)$ est la fréquence du terme t.

3.2 Le score des liens entrants

L'intuition derrière l'exploitation des liens entrants est que « *un bon document est un document qui est référencé par un grand nombre de bons documents* ». Cette idée est exploitée dans notre modèle, en tenant compte en plus des scores des documents, du score du texte ancre des liens qui les relient. Ces deux scores sont combinés de telle sorte que la propagation du score d'un document vers un autre document est d'autant plus importante si le lien qui relie ces deux documents contient des termes de la requête. La formule qui permet de calculer ce score est la suivante:

$$ScoreLE_{k-1}(d) = \sum_{d_i\in E(d)} \frac{\left(1 + ScoreAncreLien(d_i, d)\right)\cdot Score_{k-1}(d_i)}{2\cdot |E(d)|}$$

Où:

- $ScoreLE_{k-1}(d)$ est le score des liens entrants au document d. Il est compris entre 0 et 1;
- $E(d)$ est l'ensemble des documents contenant un lien vers le document d;
- $|E(d)|$ est le nombre de documents contenant un lien vers le document d;
- d_i est un document qui contient un lien vers le document d;
- $Score_{k-1}(d_i)$ est le score du document d_i à l'itération $k-1$;
- $ScoreAncreLien(d_i, d)$ est le score du texte ancre du lien qui référence le document d dans d_i. Il est compris en 0 et 1;

Le $ScoreAncreLien(d_i, d)$ est calculé de la façon suivante:

$$ScoreAncreLien(d_i, d) = \frac{\sum_{t \in lien(d_i,d) \cap q} tf(t)}{\sum_{t \in lien(d_i,d)} tf(t)}$$

Où:

- $lien(d_i, d)$ est l'ensemble des termes contenus dans le texte ancre du lien qui référence le document d dans d_i;
- q est la requête utilisateur;
- $tf(t)$ est la fréquence du terme t.

3.3 Le score des liens sortants

L'idée derrière l'exploitation des liens sortants est que « *un document qui a de bonnes références est un bon document* ». Contrairement au calcul du score des liens entrants, on se limite aux scores des documents référencés, sans tenir compte du score du texte ancre des liens. La raison à cela est que ce dernier score est déjà inclus dans le calcul des scores des documents référencés.

La formule qui permet de calculer le score des liens sortants est la suivante:

$$ScoreLS_{k-1}(d) = \sum_{d_i \in S(d)} \frac{Score_{k-1}(d_i)}{|S(d)|}$$

Où:

- $ScoreLS_{k-1}(d)$ est le score des liens sortants depuis le document d. Il est compris en 0 et 1;
- $S(d)$ est l'ensemble des documents référencés par le document d;
- $|S(d)|$ est le nombre des documents référencés par le document d;
- d_i est un document qui est référencé par le document d;
- $Score_{k-1}(d_i)$ est le score du document d_i à l'itération $k-1$.

3.4 Algorithme de recherche

Les formules présentées précédemment sont utilisées pour trouver, pour chaque document, un nouveau score de pertinence qui sera utilisé pour reclasser les documents avant de les restituer à l'utilisateur. Dans ce qui suit nous résumons l'algorithme de recherche basé sur ces fonctions. On considère que $C = \{d_1 \dots d_n\}$ est la collection de documents et que q est la requête de l'utilisateur.

1. Construire le sous-ensemble E de C. Pour chaque document d de C faire:
   1.1. Calculer le $Score_0(d)$ en utilisant un modèle de RI sans prise en compte des liens;
   1.2. Si le $Score_0(d) \neq 0$, ajouter d à E.
2. Initialiser $k = 0$
3. Tant que les scores ne convergent pas, faire:
   3.1. Incrémenter k;
   3.2. Pour chaque document d de E faire:
      3.2.1.Calculer le $ScoreT(d)$;
      3.2.2.Calculer le $ScoreLE_{k-1}(d)$;
      3.2.3.Calculer le $ScoreLS_{k-1}(d)$;
      3.2.4.Calculer le $Score_k(d)$.
4. Ordonner les documents dans E suivant l'ordre décroissant de leurs scores: $Score_k(d)$;
5. Retourner à l'utilisateur les documents ainsi ordonnés;
6. Fin de l'algorithme.

Comme indiqué dans l'algorithme, le nombre d'itérations n'est pas un paramètre qui peut être fixé à l'avance pour toutes les requêtes. Il est déterminé dynamiquement au niveau de l'appariement de telle sorte que les scores des documents convergent, et qu'une nouvelle itération n'apporte plus de modifications significatives des scores, ce qui se produit en général après une quinzaine d'itérations.

4 Conclusion

Dans ce chapitre nous avons proposé un nouveau modèle de RI pour la prise en compte des liens qui intègre le titre des documents, les liens entrants, les liens sortants ainsi que le texte ancre des liens pour le calcul des scores.

Dans le chapitre suivant nous nous intéressons à l'implémentation et à la validation de notre modèle à travers les expérimentations menées sur les deux collections de tests INEX 2006 et INEX 2009.

Chapitre 4: Implémentation et expérimentations

1 Introduction

Dans le chapitre précédent nous avons présenté un nouveau modèle pour la prise en compte des liens dans la recherche d'information semi-structurée. Nous implémentons ce modèle sur la plateforme de RI *Terrier* (voir annexe B) afin d'effectuer les expérimentations qui s'imposent.

Dans ce chapitre nous décrivons tout d'abord l'extension que nous avons apportée à Terrier qui nous a permis d'implémenter notre modèle, mais aussi un grand nombre d'autres modèles dont HITS, SALSA et l'activation propagée qui nous serviront de base de comparaison lors des expérimentations. Ensuite, nous décrivons les collections de tests INEX 2006 et INEX 2009, et enfin, nous présentons les résultats de nos expérimentations sur ces collections.

2 Implémentation: un module de liens pour Terrier

Nous avons choisi pour l'implémentation de notre modèle la plateforme de RI Terrier. C'est une plateforme de recherche d'information open-source (libre) entièrement écrite en Java, qui a été développée par le département informatique de l'université de Glasgow en Écosse. Elle est utilisée avec succès pour la recherche ad-hoc, la recherche web et la recherche inter-langages dans des environnements centralisés et décentralisés. Cette plateforme présente plusieurs avantages, notamment le fait qu'elle soit libre et qu'elle ait été prévue pour être facilement extensible (voir annexe B pour une présentation plus complète de cette plateforme).

Une partie de notre travail a consisté à créer un module pour étendre Terrier afin de pouvoir tenir compte des liens dans la RI. Lors de sa réalisation, nous nous sommes imposés, entre autres, les objectifs suivants:

1. le module doit être fiable;
2. le module doit être rapide aussi bien au niveau de l'indexation des liens qu'au niveau de l'appariement;
3. pouvoir lui ajouter facilement de nouveaux modèles de RI;
4. le module doit être facilement utilisé avec Terrier, sans nécessairement utiliser une version modifiée de Terrier.

En tenant compte de ces objectifs, nous avons procédé à élaboration de ce module. Celui-ci étend Terrier principalement sur deux niveaux: au niveau de l'indexation et au niveau de l'appariement. Nous décrivons cela dans ce qui suit.

2.1 Extension au niveau de l'indexation

L'extension au niveau de l'indexation a été la partie la plus difficile lors de l'implémentation. En effet, Terrier n'indexe pas les liens des documents. Il a donc fallu créer de nouvelles classes pour extraire les liens depuis les documents XML, et aussi créer de nouvelles structures pour accueillir l'information ainsi extraite.

Dans un souci d'adaptabilité et d'extensibilité, les classes que nous avons ajoutées à Terrier pour l'extraction des liens, permettent non seulement l'extraction des liens XLink depuis des documents XML, mais aussi l'extraction des liens depuis des documents HTML. Le choix des liens à extraire, ainsi que d'autres réglages, peuvent être facilement opérés au niveau du fichier de configuration de Terrier (généralement nommé « *terrier.properties »,* voir annexe B).

Parmi les structures que nous avons ajoutées à Terrier :

- l'index des liens sortants (nommé *outlink.bf*);
- l'index des liens entrants (nommé *inlink.bf*);
- l'index du texte ancre des liens (nommé *linkterms.bf*).

Ces structures utilisent le même format que les autres structures de Terrier.

Au final, nous avons ajouté un total de 36 nouvelles classes pour gérer l'indexation des liens.

2.2 Extension au niveau de l'appariement

L'extension au niveau de l'appariement consiste à exploiter l'information contenue dans les liens que nous avons préalablement indexés pour implémenter différents modèles. Pour ce faire, une classe utilitaire qui permet de créer et de gérer facilement un graphe de liens a été créée. Cette classe permet d'accéder à l'index des liens de façon transparente et permet de créer des modèles de RI pour la prise en compte des liens en quelques lignes de code seulement.

Tout comme pour l'indexation, l'extension que nous avons apportée au niveau de l'appariement est facilement paramétrable. Il est ainsi possible de choisir et de paramétrer un modèle de RI pour la prise en compte des liens directement depuis le fichier de configuration de Terrier (généralement nommé « *terrier.properties »,* voir annexe B).

Au final, un total de 17 classes ont été ajoutées au niveau de l'appariement dont 9 sont des classes qui implémentent des modèles de RI pour la prise en compte des liens.

2.3 Synthèse

Après l'implémentation, le module obtenu répond bien aux objectifs cités précédemment.

Ainsi, au niveau de la fiabilité, plusieurs tests ont été réalisés avec succès: le module permet d'indexer les liens et de retrouver facilement à partir d'un document l'ensemble des liens qu'il contient.

Au niveau de la rapidité d'indexation, l'indexation des liens sortants (et du texte ancre de ces liens) étant faite au même moment que l'indexation du contenu des documents, elle s'avère peu couteuse en temps d'exécution. Pour ce qui est de l'indexation des liens entrants, elle est aussi réalisée en un temps raisonnable qui dépend du nombre de liens dans la collection. Généralement, le temps supplémentaire qu'engendre l'indexation des liens est négligeable par rapport au temps d'indexation du contenu des documents de la collection.

Au niveau de la rapidité d'exécution de l'appariement, celle-ci dépend grandement du modèle et de la taille de la collection utilisés. Par exemple, le temps d'exécution supplémentaire qu'engendre le modèle « *InDegree* » est négligeable pour les deux collections INEX 2006 et INEX 2009. Par contre, le temps d'exécution supplémentaire qu'engendre le modèle « *ActivationPropagee* » peut atteindre une seconde pour la collection INEX 2006 et une vingtaine de secondes pour la collection INEX 2009.

Enfin, le dernier objectif a lui aussi été atteint. En effet, il est possible d'utiliser le module pour la prise en compte des liens, simplement en modifiant le fichier de configuration de Terrier. Il n'est donc pas nécessaire de recompiler entièrement Terrier.

3 Expérimentations

Les expérimentations que nous avons menées ont été réalisées sur les deux collections INEX 2006 et INEX 2009. Le but derrière l'utilisation de ces deux collections est d'analyser l'impacte du nombre de documents dans la collection sur les résultats obtenus. Dans ce qui suit, nous commençons par présenter les collections de tests INEX 2006 et INEX 2009. Ensuite, nous décrivons le protocole d'évaluation utilisé lors des tests. Enfin, nous présentons les résultats de notre évaluation.

3.1 Les collections de tests

3.1.1 La collection INEX 2006

La collection de tests INEX 2006 a été construite à partir d'articles de l'encyclopédie libre *Wikipedia.* Elle contient prés de 660 000 documents, et offre un ensemble de 125 requêtes pour les évaluations. Le tableau suivant montre les principales caractéristiques de cette collection:

Taille de la collection	*4,6 Go*
Nombre de documents	*659 387*
Nombre de liens	*13 562 121*
Nombre moyen de liens/document	*21*
Nombre de topics	*125*

Figure 18: Caractéristiques de la collection INEX 2006

3.1.2 La collection INEX 2009

La collection de tests INEX 2009 a été construite à partir de la collection INEX 2006 qui a en plus été enrichie avec d'autres articles de l'encyclopédie libre *Wikipedia.* Elle contient plus de 2 666 000 documents, et offre un ensemble de 107 requêtes pour les évaluations. Le tableau suivant montre les principales caractéristiques de cette collection:

Taille de la collection	*50 Go*
Nombre de documents	*2 666 190*
Nombre de liens	*110 042 341*
Nombre moyen de liens/document	*41*
Nombre de topics	*107*

Figure 19: Caractéristiques de la collection INEX 2009

3.2 Protocole d'évaluation

Notre modèle est évalué et comparé avec d'autres modèles de RI pour la prise en compte des liens. Pour ce faire, nous nous basons sur plusieurs mesures:

- La précision moyenne: c'est la moyenne arithmétique des précisions pour chaque document pertinent. Elle décrit la performance globale du modèle;
- La R-Précision: c'est la précision à x obtenue quand x est égal au nombre total de documents pertinents par rapport à la requête. Elle décrit la capacité du modèle à retourner les documents pertinents en tête de liste;
- La précision interpolée au niveau de rappel 0.01 comme recommandé dans INEX 2006. Elle représente la précision obtenue quand 1% des documents pertinents sont retournés. C'est une mesure moins globale que la R-Précision, qui décrit la capacité du modèle à retourner les documents pertinents dés les premiers résultats.

Par ailleurs, le modèle de RI sans prise en compte des liens qui est utilisé pour retrouver les premiers résultats est le modèle InL2 décrit dans [72] qui est le modèle par défaut utilisé par Terrier.

3.3 Résultats

Les résultats présentés dans cette section ont été obtenus pour les modèles suivants:

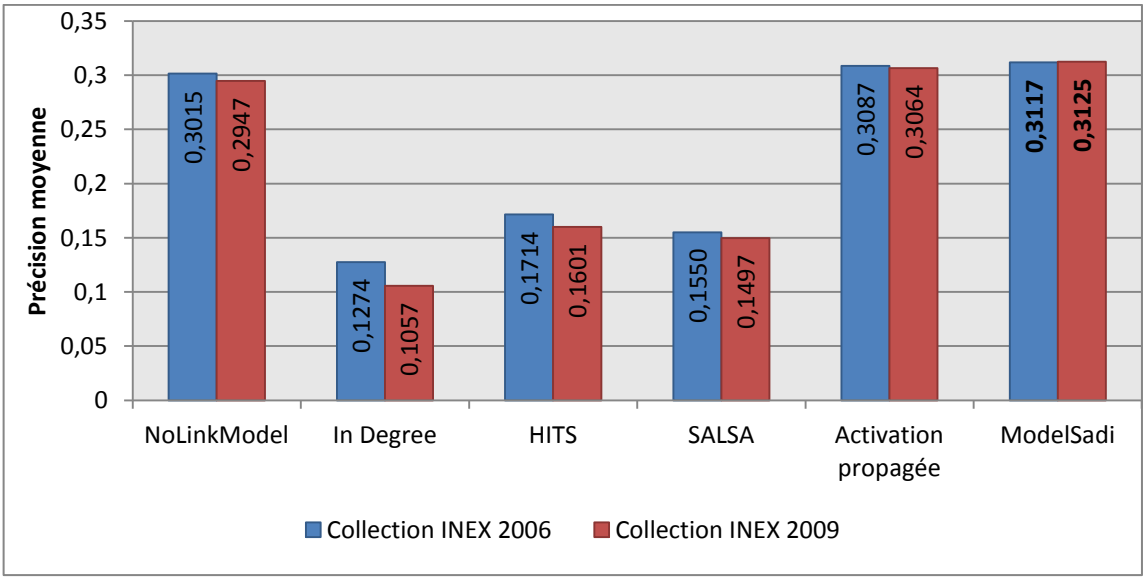
- NoLinkModel qui indique que les résultats ont été obtenus pour un modèle de RI sans prise en compte des liens;
- InDegree;
- HITS;
- SALSA;
- L'activation propagée, avec un facteur $\lambda = 0.005$ qui nous a permis d'obtenir les meilleurs résultats pour ce modèle;
- Notre modèle qui est désigné par *ModelSadi*.

Pour notre modèle nous avons effectué plusieurs tests pour le choix des paramètres λ, α, β et γ. Au final, nous avons fixé ces valeurs comme suit:

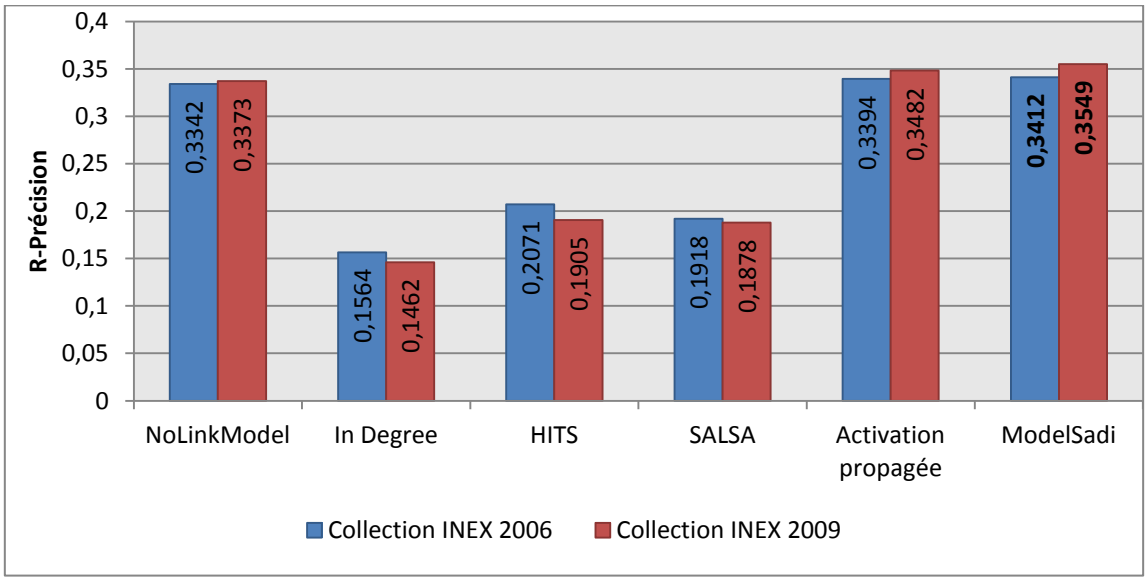
- $\lambda = 0{,}17$;
- $\alpha = 0{,}05$;
- $\beta = 0{,}35$;
- $\gamma = 0{,}60$.

Dans ce qui suit nous présentons les résultats des expérimentations obtenus sur chacun des modèles cités précédemment pour chaque collection de tests.

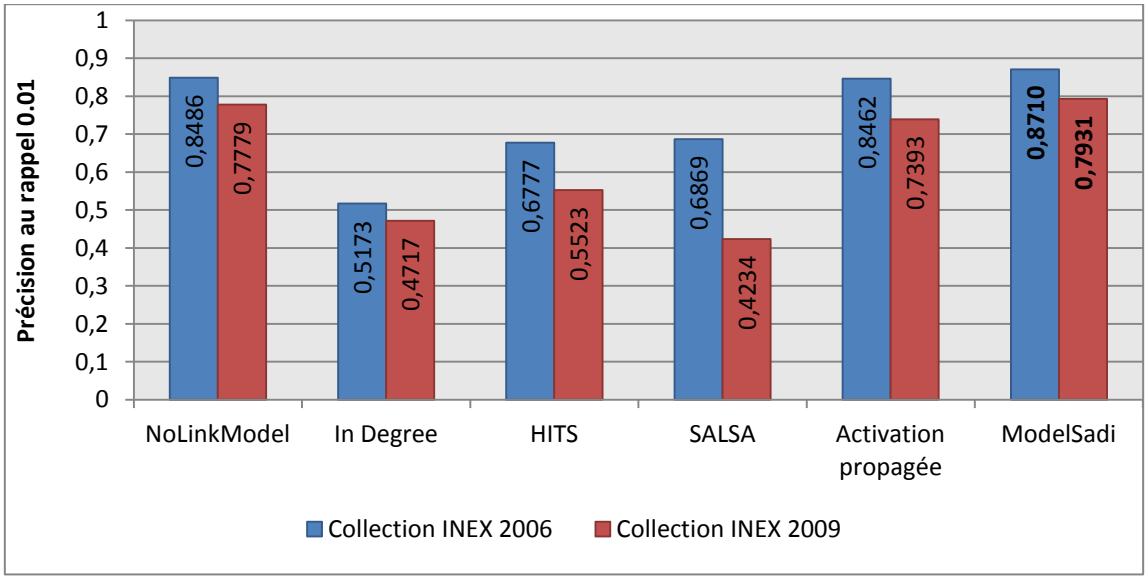
3.3.1 Précision moyenne



3.3.2 R-précision

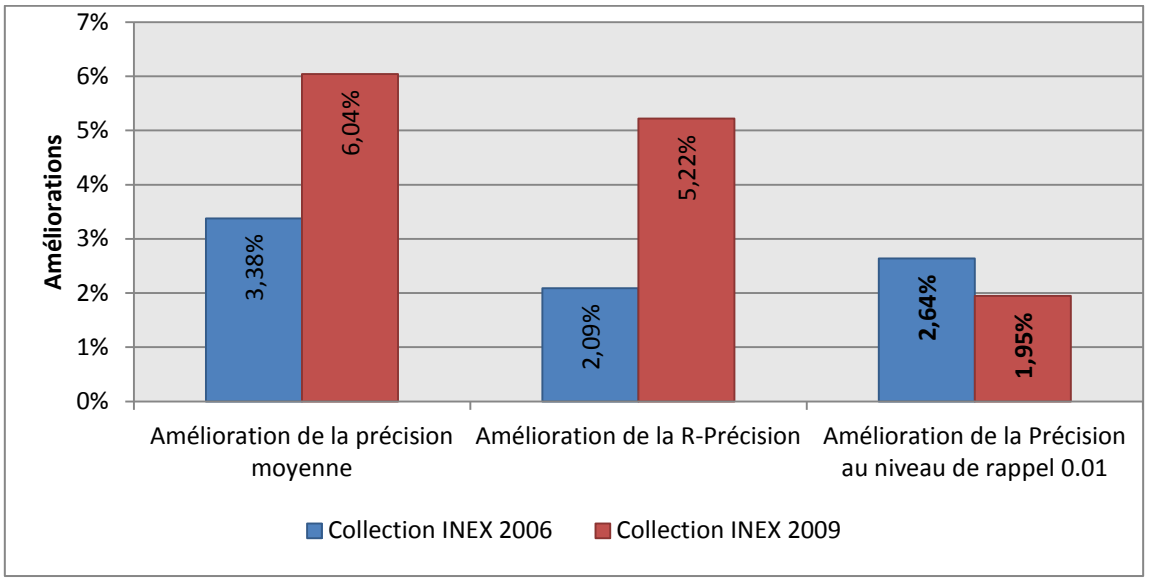


3.3.3 Précision au niveau de rappel 0.01



3.3.4 Synthèse

Les expérimentations que nous avons effectuées ont montré que notre modèle offre de meilleurs résultats par rapport aux autres modèles que nous avons testés. Ainsi, comparé au modèle sans prise en compte des liens, on obtient les taux d'amélioration suivants:



L'activation propagée offre aussi de bons résultats (bien que, de moins bonne qualité comparés à ceux obtenus avec notre modèle), et est le seul autre modèle à obtenir de meilleurs résultats que le modèle sans prise en compte des liens parmi les modèles testés.

Les autres modèles pour la prise en compte des liens à savoir: InDegree, Hits et SALSA ont obtenu des résultats moins bons que ceux obtenus par le modèle sans prise en compte des liens aussi bien au niveau de la précision moyenne, qu'au niveau de la R-Précision et de la précision interpolée au niveau de rappel 0.01.

4 Conclusion

Dans ce chapitre, nous avons d'abord décrit l'implémentation de notre modèle de RI pour la prise en compte des liens. Ensuite nous avons présenté les résultats d'expérimentation obtenus après application de ce modèle et d'autres modèles de RI sur les collections INEX 2006 et INEX 2009. Ces résultats nous ont permis de déterminer le meilleur modèle parmi les modèles testés, qui n'est autre que le modèle que nous avons proposé.

Conclusion et perspectives

1 Conclusion

Dans ce mémoire, nous avons abordé la problématique de la prise en compte des liens dans la recherche d'information semi-structurée (dans les documents XML). Nous avons commencé par introduire la recherche d'information, puis nous avons dressé un état de l'art de la recherche d'information semi-structurée. Nous avons ensuite proposé et décrit notre modèle pour la prise en compte des liens dans la RI semi-structurée, modèle qui combine les avantages de certains modèles existants. Nous avons finalement évalué ce modèle sur les collections INEX 2006 et INEX 2009.

Le modèle que nous avons proposé se démarque des autres modèles de l'état de l'art par le fait qu'il intègre le texte ancre des liens et le titre des documents dans sa fonction de correspondance.

Les expérimentations effectuées sur les collections INEX 2006 et INEX 2009 ont montré que notre modèle donne de meilleurs résultats comparé aux autres modèles de RI pour la prise en compte des liens, qui pour la plupart, n'apportaient pas de gain de précision par rapport aux modèles sans prise en compte des liens (à l'exception de l'activation propagée et de notre modèle proposé).

2 Perspectives

Nous envisageons plusieurs suites à nos travaux

- La première perspective serait d'améliorer la granularité de l'information considérée par notre modèle. En effet, principalement à cause du choix de la plateforme d'implémentation (Terrier 3.5), notre modèle considère le document comme une unité d'information indivisible. Or, dans les documents XML, comme présenté dans le 2éme chapitre, l'unité d'information pertinente est l'élément (nœud). C'est pourquoi, il serait plus intéressant de considérer les liens élément-élément, à l'image de [26] et [69]. La plateforme de RI XFirm[1] pourra être utilisée à la place de Terrier pour ce faire;
- Une deuxième perspective consiste à modifier notre modèle de telle sorte à considérer tous les documents de la collection lors du ré-ordonnancement des résultats, à l'image de [26]. Ainsi, il est possible de retrouver d'autres documents qui auraient été omis par la première recherche. Cependant, cette recherche est très couteuse et nécessite plusieurs secondes, voir plusieurs minutes par requête selon la taille de la collection. Pour indication, le temps moyen d'exécution d'une requête dans ce cas de figure, est d'environ une minute sur la collection INEX 2006;
- Une autre perspective consiste à intégrer la sémantique lors du calcul du score du texte ancre des liens. En effet, un lien contenant par exemple le texte « *automobile* » devrait être considéré comme pertinent pour une requête contenant le terme « *voiture* ». Plusieurs travaux de RI classique ont été entrepris dans ce sens dont beaucoup se basent sur le dictionnaire WordNet[2] [9]. Leur adaptation à la RI semi-structurée serait d'un bénéfice certain.

[1] http://www.irit.fr/~Karen.Pinel-Sauvagnat/xfirm.html
[2] http://wordnet.princeton.edu

Annexes

A Les technologies de la famille XML

Le succès de XML et sa vulgarisation en tant que technologie phare dans divers domaines d'application, ne sont pas seulement dus à sa simplicité, sa flexibilité et sa portabilité mais aussi aux nombreuses technologies construites autour de lui. On citera notamment les espaces de noms, les DTD, les schémas et bien d'autres technologies (voir la figure 20). Dans cette annexe, nous présentons certaines de ces technologies.

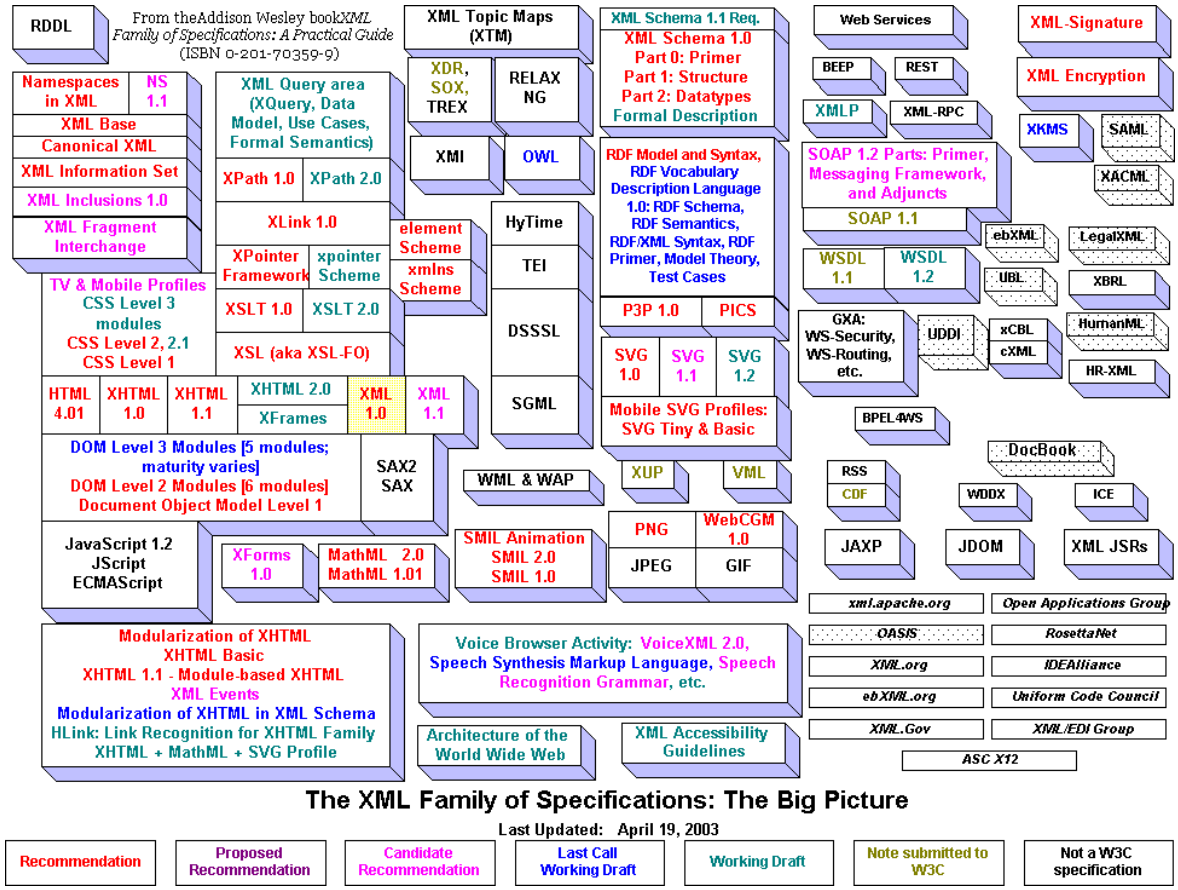


Figure 20: Les technologies de la famille XML

A.1 Les espaces de noms

Les espaces de noms (*namespaces* en anglais) est une recommandation du W3C[1] qui permet de résoudre les conflits de noms dans les documents XML. Les conflits de noms peuvent apparaitre si on utilise dans un même document XML, des noms d'éléments ou d'attributs issus de plusieurs vocabulaires différents. Ainsi, par exemple un élément dénommé « *note* » peut, selon le vocabulaire, s'agir de la note d'un élève, d'une note de musique, d'un rappel écrit … Le fait d'associer un espace de noms (nommé « *musique* » par exemple) à la balise « *note* » permet de résoudre cette ambigüité.

Dans un document XML, les espaces de noms sont associés aux éléments grâce à l'attribut « *xmlns* » (pour *XML NameSpace*) qui aura pour valeur une URI (*Uniform Resource Identifier*) qui identifiera de manière unique l'espace de noms (évitant ainsi tout conflit de noms pour les espaces de noms). Cet attribut peut également avoir la forme suivante : « *xmlns:préfixe* ».

[1] http://www.w3.org/TR/REC-xml-names/

Dans le premier cas, l'espace de noms s'applique à l'élément où se situe sa déclaration et tout son contenu. Dans le second cas, l'espace de noms s'applique seulement aux éléments préfixés par l'espace de noms.

L'exemple ci-dessous illustre l'utilisation des espaces de noms.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<commande xmlns:produit="http://mon-entreprise.com/XML/produit"
xmlns:client="http://mon-entreprise.com/XML/client">
      <produit:numero>p7325</produit:numero>
      <produit:nom>Alienware M18x</produit:nom>
      <produit:quantite>1</produit:quantite>
      <produit:prix>1999,00</produit:prix>
      <client:numero>c16212</client:numero>
      <client:nom>Dupont, François</client:nom>
</commande>
```

Figure 21: Exemple d'utilisation des espaces de noms XML

Dans cet exemple, sans les espaces de noms, on ne pourrait pas distinguer le numéro et le nom du produit du numéro et du nom du client.

A.2 Les DTD

Une DTD[2] (*Document Type Définition*) est un document qui décrit une grammaire permettant de valider un document XML. Issues de l'univers SGML (*Standard Generalized Markup Language*), les DTD sont aussi utilisées dans XML.

Les DTD présentent l'avantage d'être simples au niveau de la syntaxe (utilisent une syntaxe non XML) mais présente l'inconvénient d'être pauvre en possibilités de contrôle surtout au niveau du typage de données. Par ailleurs, les espaces de noms sont difficilement gérables car il faut intégrer, dans la grammaire, les préfixes, ce qui est contraire à l'esprit des espaces de noms.

Une DTD peut être associée à un document XML selon deux façons. En l'ajoutant directement dans le document XML, la DTD est alors dite interne. Ou en indiquant une référence vers la DTD, qui se trouve dans un fichier externe, la DTD est alors dite externe. Cette dernière forme de DTD est généralement préférée car elle permet d'avoir des documents XML moins volumineux et simplifie les procédures de maintenance.

L'ajout de DTD se fait avant le premier élément et après le prologue dans un document XML. L'exemple suivant montre un fichier XML auquel est associée une DTD externe.

[2] http://www.w3.org/TR/REC-xml/#dt-doctype

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE réservation SYSTEM "reservation.dtd">

<réservation>
      <client numéroClient="8241">
            <nom>Dupont</nom>
            <prénom>François</prénom>
      </client>
      <chambre numéroChambre="73">
            <type>standard</type>
      </chambre>
      <payement>
            <date>2012-08-03</date>
            <durée>3</durée>
            <total>600</total>
            <type>cheque</type>
      </payement>
</réservation>
```

Figure 22: Fichier XML avec une DTD externe

Dans cet exemple, la deuxième ligne permet d'associer la DTD nommée « *reservation.dtd* » au document XML. Pour ce faire, elle définit trois attributs. Le premier est le nom de l'élément principal du document (qui est « *réservation* »). Le second spécifie quel est le type de la DTD, ici il s'agit de « *SYSTEM* » qui indique que la DTD se trouve dans un fichier externe. Enfin, le troisième indique l'URI pour accéder à la DTD externe.

Commençons à présent à analyser la syntaxe d'une DTD. Tout d'abord, il faut savoir que celle-ci est une syntaxe non XML à base d'instructions. Une instruction a la forme suivante: « <!INSTRUCTION …> ». On distingue principalement deux types d'instructions: la définition d'un élément et la définition d'un attribut.

La définition d'un élément a la forme suivante: « <!ELEMENT nom contenu> ». Le nom indique l'élément à considérer et le contenu indique le contenu autorisé pour cet élément. Celui-ci peut être vide (« *EMPTY* »), du texte (« *(#PCDATA)* »), d'autres sous-éléments en indiquant leur nom ou n'importe quel autre élément présent dans la DTD (« *ANY* »).

La définition d'un attribut a la forme suivante: « <!ATTLIST nom_element liste_attributs> ». La première valeur indique le nom de l'élément pour lequel s'applique cette instruction. La seconde est une liste d'attributs autorisés pour cet élément. Chaque attribut a alors la forme suivante: « nom type obligation valeur_par_defaut ». Le nom est le nom de l'attribut, le type indique le contenu qu'il peut avoir (« *CDATA* » pour du texte), l'obligation indique si l'attribut est obligatoire ou optionnel et la valeur par défaut est utilisée si l'attribut est optionnel et qu'il n'est pas spécifié.

Voici un exemple d'une DTD associée au document XML donné dans la figure 22.

```
<!ELEMENT réservation (client,chambre,payement)>
<!ELEMENT client (nom,prénom)>
<!ELEMENT nom (#CDATA)>
<!ELEMENT prénom (#CDATA)>
<!ELEMENT chambre (type)>
<!ELEMENT payement (date,durée,total,type)>
<!ELEMENT date (#CDATA)>
<!ELEMENT durée (#CDATA)>
<!ELEMENT total (#CDATA)>
<!ELEMENT type (#CDATA)>

<!ATTLIST client
      numéroClient CDATA #REQUIRED>
<!ATTLIST chambre
      numéroChambre CDATA #REQUIRED>
```

Figure 23: DTD associée au document XML donné dans la figure 22

A.3 Les schémas XML

Les schémas XML est une recommandation[3] du W3C depuis 2001, qui définit une grammaire de documents XML exprimée dans un formalisme XML. Tout comme les DTD, les schémas permettent de valider un document XML conformément à une grammaire donnée. Mais contrairement aux DTD, les schémas offrent plus de souplesse au niveau du typage de données et au niveau de la gestion des espaces de noms.

L'assignation d'un schéma à un document XML se fait à l'aide d'attributs supplémentaires insérés dans l'élément racine du document. Ces attributs doivent appartenir à l'espace de noms *http://www.w3.org/2001/XMLSchema-instance*. L'attribut qui indique l'emplacement du schéma est « *schemaLocation* » si le schéma prend en compte les espaces de noms, et est « *noNamespaceSchemaLocation* » si le schéma ne prend pas en compte les espaces de noms.

Au niveau du typage de données, les schémas XML offrent deux possibilités: les types simples et les types complexes.

Les types simples sont utilisés pour les attributs et les éléments sans attributs qui ont du texte comme contenu. Il existe plusieurs types simples de bases présents dans l'espace de noms par défaut des schémas. Notamment, le type « *string* » pour les chaines de caractères, « *int* » pour les entiers, « *date* » pour les dates… Les schémas offrent aussi la possibilité de construire ses propres types simples en appliquant des restrictions sur des types simples déjà existants (en définissant une énumération ou en imposant une longueur maximum, un pattern…). Ceci est fait grâce à l'élément « *simpleType* »;

Les types complexes sont utilisés pour tout les autres cas d'usages non gérables par les types simples. Ceci inclut la définition de types d'éléments contenant un ou plusieurs sous-éléments, des attributs, du contenu mixte (texte et éléments)… Les types complexes sont définis à l'aide de l'élément « *complexeType* », qui peut alors contenir les éléments: « *attribute* » pour indiquer la présence d'un attribut, « *séquence* » pour indiquer une suite d'éléments obligatoires, « *choice* » pour indiquer la présence d'un élément au choix parmi ceux listés, et « *all* » pour indiquer une suite d'éléments obligatoires mais dont l'ordre n'est pas important.

[3] http://www.w3.org/XML/Schema

Les éléments sont représentés dans les schémas par un élément « *element* » qui aura pour attribut le nom de l'élément et son type.

L'exemple suivant montre un document XML et le schéma qui lui est associé.

```xml
<?xml version="1.0" encoding="UTF-8" ?>

<réservation xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="reservation.xsd">
      <client numéroClient="8241">
            <nom>Dupont</nom>
            <prénom>François</prénom>
      </client>
      <chambre numéroChambre="73">
            <type>standard</type>
      </chambre>
      <payement>
            <date>2012-08-03</date>
            <durée>3</durée>
            <total>600</total>
            <type>cheque</type>
      </payement>
</réservation>
```

Figure 24: Fichier XML associé à un schéma XML

```xml
<?xml version='1.0' encoding='utf-8' ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="réservation" type="réservationType" />
  <xsd:complexType name="réservationType">
    <xsd:sequence>
      <xsd:element name="client" type="clientType" />
      <xsd:element name="chambre" type="chambreType" />
      <xsd:element name="payement" type="payementType" />
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="clientType">
    <xsd:sequence>
      <xsd:element name="nom" type="xsd:string" />
      <xsd:element name="prénom" type="xsd:string" />
    </xsd:sequence>
    <xsd:attribute name="numéroClient" type="xsd:positiveInteger">
  </xsd:complexType>
  <xsd:complexType name="chambreType">
    <xsd:sequence>
      <xsd:element name="type" type="xsd:string" />
    </xsd:sequence>
    <xsd:attribute name="numéroChambre" type="xsd:positiveInteger">
  </xsd:complexType>
  <xsd:complexType name="payementType">
    <xsd:sequence>
      <xsd:element name="date" type="xsd:date" />
      <xsd:element name="durée" type="xsd:positiveInteger" />
      <xsd:element name="total" type="xsd:positiveInteger" />
      <xsd:element name="type" type="xsd:string" />
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

Figure 25: Schéma XML associé au document XML de la figure 24

A.4 Les feuillets de styles

Un feuillet de style est un document permettant de mettre en forme un autre document, par exemple de transformer un document XML en un document PDF. Un langage de feuillet de style définit une syntaxe pour les feuillets de styles. Il existe plusieurs langages de feuillet de style, notamment XSLT et XSL-FO qui sont présentés dans cette section.

A.4.1 XSLT

XSLT (*eXtensible Stylesheet Language Transformations*) défini dans la recommandation XSL du W3C est un langage XML qui permet de passer d'un document dans un format XML à un document dans un autre format texte (XML, XHTML/HTML, CSV …). Il est apparu en deux versions, la première[4] se base sur XPath1, et la deuxième[5] sur XPath2.

L'algorithme de transformation de XSLT se résume à parcourir l'arbre du document XML grâce à des expressions XPath, et à exécuter des blocs d'instructions sur les éléments ainsi retournés pour construire le nouveau document. Pour ce faire, XSLT offre plusieurs instructions de bases présentes dans l'espace de noms « *http://www.w3.org/1999/XSL/Transform* ». D'autres instructions provenant d'autres langages peuvent aussi être utilisées en spécifiant l'espace de noms d'où elles proviennent.

A.4.2 XSL-FO

XSL-FO (*eXtensible Stylesheet Language - Formatting Objects*) fait aussi partie de la recommandation XSL du W3C[6]. Son rôle est d'offrir un format de présentation, comme HTML, mais servant à générer des formats plus complexes, comme les formats PDF, RTF et PostScript. Il vient ainsi compléter XSLT qui n'aurait pas pu être utilisé comme tel pour des formats manipulant des données binaires.

Un document XSL-FO est composé d'instructions de mise en page et d'instructions d'affichage contenues dans l'espace de noms « *http://www.w3.org/1999/XSL/Format* ». Les instructions de mise en page concernent le dimensionnement des pages du document et le découpage des pages en régions. Les instructions d'affichages concernent le contenu des pages et permettent de définir le contenu de chaque région d'une page. Grâce à ce système, XSL-FO permet de gérer facilement les régions statiques comme les pieds de pages et la numérotation des pages.

A.5 Autres technologies XML

Il existe un grand nombre de technologies basées sur XML, non moins importantes les unes que les autres, que nous ne pouvons toutes présenter dans cette annexe. Parmi ces technologies:

- *GML*: Langage pour encoder, manipuler et échanger des données géographiques;
- *MathML*: Langage pour décrire les notations mathématiques;
- *RelaxNG*: Langage de schéma pour la validation des documents XML, au même titre que les DTD et les schémas XML;
- *SVG*: Format vectoriel qui sert à représenter un dessin en 2D dans un formalisme XML;
- *TEI*: Format pour faciliter la création, l'échange et l'intégration de données textuelles;
- *XHTML*: Langage de présentation de documents WEB. Dérive de HTML.

[4] http://www.w3.org/TR/xslt/
[5] http://www.w3.org/TR/xslt20/
[6] http://www.w3.org/TR/xsl/

B La plateforme de RI Terrier

B.1 Introduction

Terrier[7] (TERabyte RetrIEveR) est une plateforme de recherche d'information open-source (libre) entièrement écrite en Java, qui a été développée par le département informatique de l'université de Glasgow en Écosse. Parmi les domaines d'application de Terrier, on trouve la recherche Ad-hoc, la recherche Web et la recherche multilingue dans des environnements centralisés et distribués. Il offre pour cela un ensemble d'outils pour l'indexation, la recherche et l'évaluation des résultats.

La version actuelle de Terrier est la version *3.5* apparue en Juin 2011. Elle apporte plusieurs améliorations par rapport à la version précédente (*3.0*) tout en restant compatible avec celle-ci (les structures d'index de Terrier 3.0 pouvant être réutilisées dans Terrier 3.5). Les modifications qu'apporte la version de Terrier 3.5 se situent notamment au niveau de la tokénisation pour une gestion améliorée de la recherche multilingue et au niveau de l'appariement où plusieurs nouveaux modèles de RI ont été ajoutés.

Dans ce qui suit, nous nous intéressons à la version 3.5 de Terrier. Nous commençons par présenter le package Terrier 3.5 qui est librement téléchargeable dans le site officiel de Terrier[7]. Ensuite, nous nous intéressons à l'utilisation de Terrier. On présentera notamment comment configurer Terrier, et les différentes applications de Terrier. Enfin, nous nous pencherons sur le fonctionnement de Terrier. On y présentera notamment la procédure à suivre afin d'étendre Terrier.

B.2 Le package Terrier 3.5

Le package Terrier 3.5 peut être obtenu gratuitement dans le site officiel de Terrier[7]. Une fois extrait on trouve les dossiers suivants:

B.2.1 Le dossier bin

Contient un ensemble de scripts qui permettent de lancer différentes applications de Terrier. Ces scripts sont présents en deux formats: dans le format Batch pour une utilisation sous Windows et dans le format sh (Shell) pour une utilisation sous Unix.

B.2.2 Le dossier doc

Contient la documentation de Terrier. Il s'agit d'un ensemble de pages HTML qui présentent Terrier et du « *javadoc* »[8] généré automatiquement avec java depuis le code source de Terrier.

B.2.3 Le dossier etc

Contient le fichier de configuration de terrier nommé « *terrier.properties* », ainsi que le fichier « *terrier.propertie.sample* » qui contient la plupart des propriétés de Terrier sur lequel on peut se baser pour créer une nouvelle configuration.

[7] http://www.terrier.org/
[8] http://download.oracle.com/javase/7/docs/api/

B.2.4 Le dossier lib

Contient un ensemble de librairies compressées dans le format « *jar* »[9]. Les librairies contenues dans ce dossier sont toutes ajoutées au « *Class Path* » de Java lors de l'utilisation des scripts fournis dans le dossier « *bin* ». Ainsi, si on veut, par exemple, ajouter une extension à Terrier il suffit d'ajouter les librairies de cette extension dans ce dossier, sans nécessairement recompiler Terrier.

Parmi les libraires qu'on trouve dans ce dossier, on trouve la librairie principale de Terrier nommée « terrier-3.5-core.jar » et l'ensemble des librairies externes utilisées par Terrier.

B.2.5 Le dossier licenses

Contient les licences des librairies utilisées par Terrier. La licence utilisée par Terrier est la licence MPL[10] dont un exemplaire se trouve à la racine du package et est nommée « *LICENSE.txt* ».

B.2.6 Le dossier share

Contient la liste des mots vides et un ensemble de documents pour tester Terrier.

B.2.7 Le dossier src

Contient le code source de Terrier. Celui-ci sera détaillé plus tard dans cette annexe.

B.2.8 Le dossier var

Contient le dossier « *index* » contenant les structures d'index, et le dossier « *results* » contenant les résultats des évaluations.

B.3 Utilisation de Terrier

L'utilisation de Terrier passe d'abord par sa configuration. C'est pourquoi, nous décrivons d'abord comment configurer Terrier. Ensuite, nous présentons l'utilisation de Terrier à travers la présentation des scripts dans le dossier « *bin* ».

B.3.1 Configuration de Terrier

Toute la configuration de Terrier, aussi bien pour l'indexation, la recherche et l'évaluation, est centralisée dans le fichier « *terrier.properties* » dans le dossier « *etc* ».

Ce fichier est un document texte où chaque ligne correspond à une propriété. Celle-ci a le format suivant:

```
Nom_de_la_propriété=valeur_de_la_propriété
```

L'insertion de commentaires est également possible en commençant la ligne contenant le commentaire avec le caractère « # ».

La plupart des propriétés de Terrier peuvent être retrouvées dans le fichier « *terrier.propertie.sample* » dans le dossier « *etc* ». Parmi elles on trouve les propriétés suivantes:

- *collection.spec*: indique le chemin du fichier qui contient la liste des documents qui seront indexés;
- *querying.default.controls*: permet de définir le modèle de RI qui sera utilisé lors de la recherche. Par exemple la valeur : « *wmodel:BM25* » indique que le modèle BM25 sera utilisé lors de la recherche;

[9] Java archive. Voir: http://java.sun.com/docs/books/tutorial/deployment/jar/
[10] « *Mozilla Public License* ». Voir: http://www.mozilla.org/MPL/

- *querying.postprocesses.order*: indique quels sont les « *postprocess* » qui seront utilisés lors de la recherche. Cette propriété permet par exemple d'utiliser l'expansion de requête;
- *stopwords.filename*: indique le chemin du fichier qui contient les mots vides;
- *termpipelines*: indique si une « *stoplist* » doit être utilisée lors de l'indexation, et quel est l'algorithme de lemmatisation qui doit être utilisé (par défaut c'est l'algorithme de *Porter qui est* utilisé);
- *terrier.var*: indique le dossier où se trouve l'index et les résultats d'évaluation de Terrier;
- *trec.qrels*: indique le chemin du fichier qui contient les jugements de pertinences correspondant aux requêtes;
- *trec.topics*: indique le chemin du fichier qui contient les requêtes (*topics*) pour l'évaluation.

B.3.2 Les scripts de Terrier

Les scripts de Terrier se trouvent dans le dossier « *bin* ». Nous décrivons dans ce qui suit quelques-uns de ces scripts. Nous nous limiterons pour cela aux scripts Shell (Unix). Les scripts Batch (Windows) s'utilisent de façon très similaire à leur équivalent en Shell.

B.3.2.1 Le script « terrier-env.sh »

Ce script est invoqué automatiquement par chaque script du dossier « *bin* ». Il est utilisé pour initialiser les variables d'environnement de Terrier. On peut par exemple, grâce à ce script définir un nouvel emplacement pour le fichier de configuration de Terrier.

B.3.2.2 Le script « desktop-terrier.sh »

Ce script peut être exécuté directement depuis un terminal Unix et ne requiert pas d'arguments particuliers. Il permet de lancer l'application Desktop-Terrier (voir la figure 26).

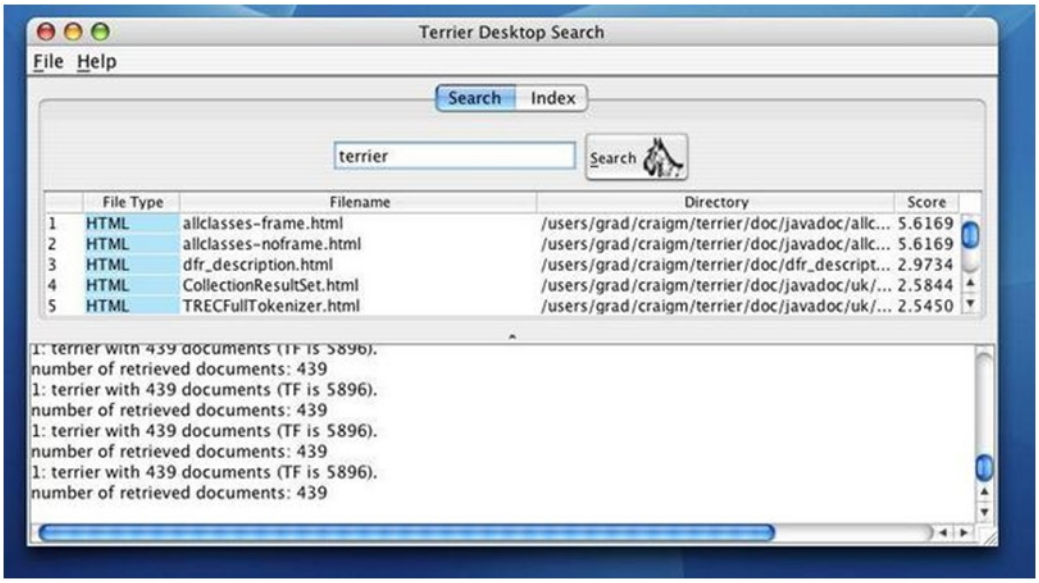


Figure 26: L'application Desktop-Terrier

L'application Desktop-Terrier présente deux onglets: un onglet pour la recherche et un autre onglet pour l'indexation.

L'onglet indexation permet d'indexer les dossiers choisis par l'utilisateur. Desktop-Terrier utilise la classe « *SimpleFileCollection* » pour parcourir récursivement ces dossiers. Cette classe permet de

reconnaitre un grand nombre de type fichiers par leur extension (notamment les fichiers Pdf, Word, PowerPoint, Excel, textes et XML).

L'onglet recherche permet de formuler une requête et de récupérer les résultats. Ces derniers sont affichés dans un tableau selon l'ordre décroissant de leur pertinence. Desktop-Terrier offre ensuite la possibilité d'ouvrir le fichier associé à un résultat en double cliquant dessus.

Desktop-Terrier permet de tester rapidement les fonctionnalités de Terrier, mais est néanmoins déconseillé pour des travaux complexes d'indexation et de recherche. De plus, Desktop-Terrier ne permet pas l'évaluation des résultats.

B.3.2.3 Le script « trec-setup.sh »

Le script Trec-Setup reçoit en argument le nom du dossier où se trouve la collection de documents à indexer. Ce dossier est parcouru récursivement pour créer le fichier « *collection.spec* » dans « *etc* » qui contiendra la liste des fichiers à indexer.

Ce script doit être exécuté au préalable avant d'appeler l'indexation avec le script Trec-Terrier.

B.3.2.4 Le script « trec-terrier.sh »

Le script Trec-Terrier permet l'indexation, la recherche et l'évaluation. L'indexation est réalisée à l'aide de l'argument « -i », la recherche avec l'argument « -r » et l'évaluation avec le paramètre « -e ».

Le mode indexation utilise le fichier « *collection.spec* » (créé avec le script « trec-setup.sh ») qui indique la liste des documents à indexer. Terrier permet l'indexation d'un grand nombre de types de fichiers, notamment les fichiers Pdf, Word, PowerPoint, Excel, textes et XML.

Le mode recherche permet d'exécuter un ensemble de requêtes (topics) et de récupérer les résultats de ces requêtes dans le dossier « *var/results* ». La liste des requêtes est spécifiée à l'aide de la propriété « *trec.topics*».

Le mode évaluation permet d'évaluer les résultats qui se trouvent dans le dossier « *var/results* ». Pour chaque résultat, la précision moyenne, la R-Précision et la précision à différents rangs et niveaux de rappel sont calculées et sauvegardées dans un fichier dans le dossier « *var/results* ». Pour obtenir la précision pour chaque requête individuellement il faut utiliser l'argument « -q ». Le fichier qui continent les jugements de pertinence est indiqué dans la propriété « *trec.qrels* ».

B.4 Fonctionnement de Terrier

Terrier offre trois API principales: l'API d'indexation, l'API de recherche et l'API d'évaluation. Nous présentons dans ce qui suit le fonctionnement de chacune de ces API à travers l'analyse du code source de Terrier (qui se trouve dans le dossier « *src* »).

B.4.1 Présentation de l'API d'indexation

Les classes qui gèrent l'indexation se trouvent dans les deux paquets « *org.terrier.structures* » et « *org.terrier.indexing* ». Les classes du premier paquet permettent de gérer les structures d'index de Terrier. Les classes du deuxième paquet permettent l'extraction des termes depuis différents types de documents et contiennent les classes d'indexation qui font appel aux classes gérant les structures d'index.

La classe d'indexation qui supervise toute l'indexation est la classe « *Indexer* ». Plusieurs classes étendent cette classe, notamment:

- la classe « *BasicIndexer* » qui permet une indexation à deux passes (construit d'abord l'index direct puis l'index inverse) sans sauvegarde des positions des termes;
- la classe « *BasicSinglePassIndexer* » qui permet une indexation à une passe (construit directement l'index inverse sans construction de l'index direct) sans sauvegarde des positions des termes;
- la classe « *BlockIndexer* » qui permet une indexation à deux passes avec sauvegarde des positions des termes;
- la classe « *BlockSinglePassIndexer* » qui permet une indexation à une passe avec sauvegarde des positions des termes.

La classe « *Indexer* » se base sur l'interface « *Collection* » pour déterminer la liste des documents à indexer. Parmi les classes qui implémentent cette interface on trouve notamment la classe « *SimpleFileCollection* » qui gère un grand nombre de types de documents, et la classe « *SimpleXMLCollection* » qui gère les documents XML. Le rôle de l'interface « *Collection* » est de retourner un ensemble de classes qui implémentent l'interface « *Document* ». Pour chaque document, la classe « *Indexer* » extrait les termes qu'il contient et les ajoute aux structures d'index. Ces termes passent au préalable dans le TermPipeline qui va éliminer les mots vides et lemmatiser les termes.

La figure suivante (figure 27) résume la procédure d'indexation de Terrier.

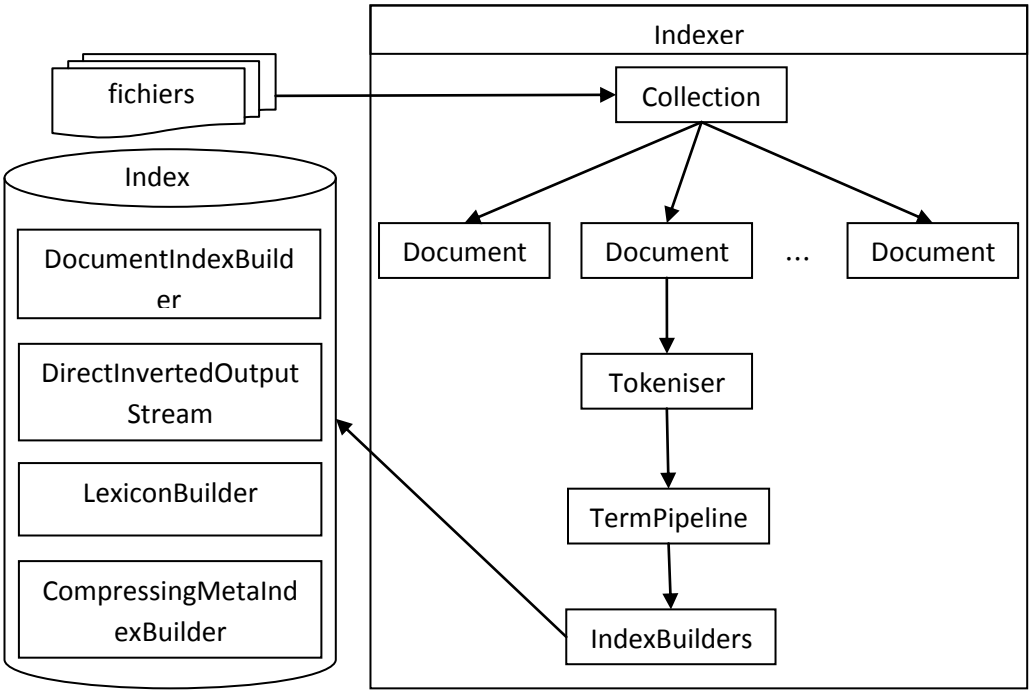


Figure 27: Indexation dans Terrier

B.4.2 Présentation de l'API de recherche

Les classes qui gèrent la recherche se trouvent dans les deux paquets « *org.terrier.matching* » et « *org.terrier.querying* ». De plus certaines classes du paquet « *org.terrier.structures* » sont utilisées pour interroger l'index.

La classe qui gère principalement la recherche dans Terrier est la classe « *Manager* » dans le paquet « *org.terrier.querying* ». Cette classe permet de créer une requête avec un ensemble de mots clés donnés par l'utilisateur. Cette requête est ensuite utilisée pour retrouver les documents pertinents à la requête. Pour ce faire, le « *Manager* » exécute une séquence de procédures sur la requête:

- « *Manger. runPreProcessing* » exécute un ensemble de prétraitements sur la requête;
- « *Manger.runMatching* » lance la recherche, et sélectionne un certain nombre de documents en utilisant les modèles de RI;
- « *Manger.runPostProcessing* » exécute un ensemble de traitements sur les résultats retournés par « *Manger.runMatching* ». Ces traitements peuvent complètement altérer les résultats comme le fait l'expansion de requête;
- « *Manger.runPostFilters* » utilise un ensemble de filtres pour diminuer le nombre des résultats retournés par Terrier.

La figure 28 illustre la procédure de recherche de Terrier.

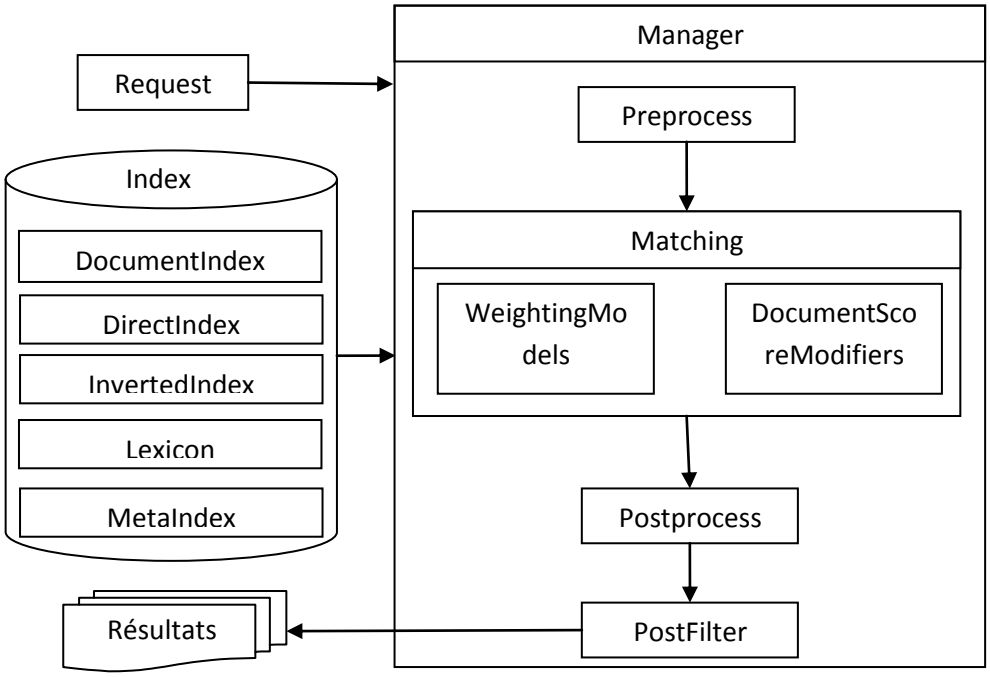


Figure 28: Recherche dans Terrier

B.4.3 Présentation de l'API d'évaluation

Les classes qui gèrent l'évaluation se trouvent dans le paquet « *org.terrier.evaluation* » de Terrier. Parmi elles, on trouve la classe «*TRECQrelsInMemory* » qui permet de charger les jugements de pertinence en mémoire, et les classes qui permettent l'évaluation d'un ensemble de résultats. Ces dernières étendent la classe abstraite «*Evaluation* ».

Le fonctionnement de l'API d'évaluation est illustré dans la figure 29.

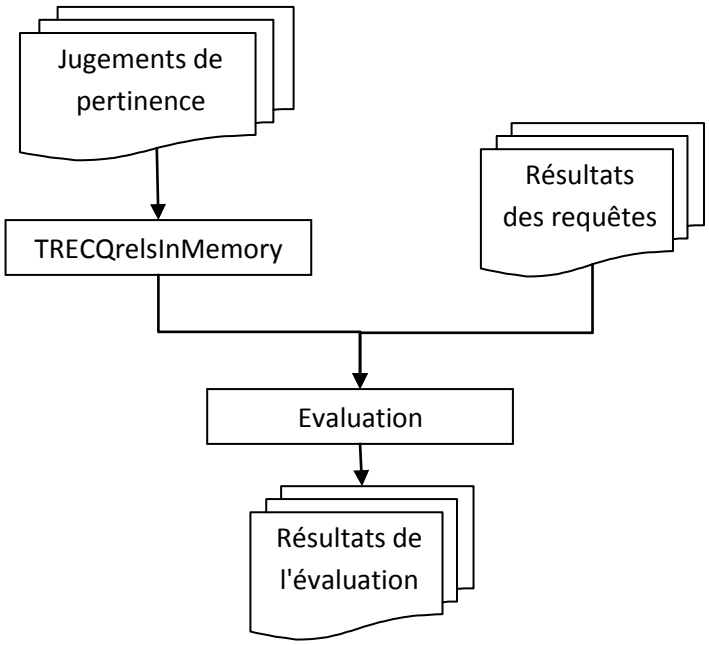


Figure 29: L'évaluation dans Terrier

Bibliographie

1 Abiteboul S, Quass D, Mchugh J, Widom J, Wiener J. The Lorel Query Language for Semistructured Data. International Journal on Digital Libraries. 1997:68-88.

2 Abolhassani M, Fuhr N. Applying the Divergence from Randomness Approach for Content-Only Search in XML Documents. In: ECIR'04. Berlin: Springer; 2004.

3 Anh VN, Moffat A. Compression and an IR Approach to XML Retrieval. In: INEX Workshop. 2002. p. 99-104.

4 B. S, Kamps J, de Rijke M. An Element-based Approach to XML Retrieval. In: INEX'03. 2003.

5 Belew RK. Adaptive information retrieval: using a connectionist representation to retrieve and learn about documents. SIGIR Forum. 1989:11-20.

6 Belkin NJ, Oddy RN, Brooks HM. Ask for information retrieval: part I.: background and theory. In: Readings in information retrieval. San Francisco, CA: Morgan Kaufmann Publishers Inc.; 1997. p. 299-304.

7 Ben Aouicha M. Une approche algébrique pour la recherche d'information structurée. Thèse de doctorat. Université Paul Sabatier; 2009.

8 Boubekeur-Amirouche F. Contribution à la définition de modèles de recherche d'information flexibles basés sur les CP-Nets. Thèse de doctorat. Toulouse: Université Paul Sabatier; 2008.

9 Boubekeur F, Boughanem M, Tamine L, Daoud M. Using WordNet for Concept-Based Document Indexing in Information Retrieval. In: The Fourth International Conference on Advances in Semantic Processing SEMAPRO. 2010.

10 Boughanem M. Systèmes de recherche d'information: d'un modèle classique à un modèle connexioniste. Thèse de doctorat. Toulouse: Université Paul Sabatier; 1992.

11 Boughanem M, Kraaij W, Nie JY. Modèles de langue pour la recherche d'information. In: Les systèmes de recherche d'informations. 2004. p. 163–182.

12 Brin S, Page L. The anatomy of a large-scale hypertextual Web search engine. Computer Networks and ISDN Systems. 1998:107-117.

13 Cattell RGG, Atwood T. The Object database standard, ODMG-93: release 1.2. Morgan Kaufmann Publishers; 1996.

14 Chebolu P, Melsted P. PageRank and the random surfer model. In: Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms. Philadelphia, PA: Society for Industrial and Applied Mathematics; 2008. p. 1010-1018.

15 Chiaramella Y, Mulhem P, Fourel F. A Model for Multimedia Information Retrieval. Technical report. University of Glasgow; 1996.

16 Chibane I. Impact des liens hypertextes sur la précision en recherche d'information. Thèse de

doctorat. Université Paris Sud; 2008.

17 Cleverdon CW. Evaluation tests of information retrieval systems. Journal of Documentation. 1970:55–67.

18 Cleverdon CW. Report on the Testing and Analysis of an Investigation into the Comparative Efficiency of Indexing Systems. Cranfield College of Aeronautics; 1962.

19 Cohn D, Chang H. Learning to Probabilistically Identify Authoritative Documents. In: Proceedings of the Seventeenth International Conference on Machine Learning. San Francisco, CA: Morgan Kaufmann Publishers Inc.; 2000. p. 167-174.

20 Cooper WS. Getting beyond Boole. Information Processing and Management. 1988.

21 Cooper WS. The potential usefulness of catalog access points other than author, title, and subject. Journal of the American Society for Information Science. 1970:112-127.

22 Crestani F, Lee PL. Searching the Web by constrained spreading activation. Information Processing and Management. 2000:585-605.

23 Crouch C, Apte S, Bapat H. An IR approach to XML retrieval based on the extended vector model. In: INEX'02. 2002. p. 98-99.

24 Deerwester S, Dumais ST, Furnas GW, Landauer TK, Harshman R. Indexing by latent semantic analysis. Journal of the American society for information science. 1990:391-407.

25 Dumais S, Landauer T, Littman M. Automatic cross-linguistic information retrieval using Latent Semantic Indexing. In: SIGIR'96 - Workshop on Cross-Linguistic Information Retrieval. 1996. p. 16-23.

26 Fellag-Berchiche S. Propagation de pertinence et exploitation du texte ancre des liens et de la balise titre pour améliorer la recherche dans les documents XML. Technical Report. Tizi-Ouzou: Université Mouloud Mammeri de Tizi-Ouzou; 2012.

27 Florescu D, Kossmann D. Storing and querying XML data using an. IEEE Data Engineering Bulletin. 1999:27-34.

28 Foltz PW, Dumais S. Personalized information delivery: an analysis of information filtering methods. Communications of the ACM. 1992.

29 Fox C. Lexical analysis and stoplists. In: Information retrieval. Upper Saddle River, NJ: Prentice-Hall; 1992.

30 Frakes WB, Baeza-Yates R. Information Retrieval: Data Structures and Algorithms. Englewood Cliffs, NJ: Prentice Hall; 1992.

31 Frei HP, Stieger D. The use of semantic links in hypertext information retrieval. Information Processing and Management. 1995:1-13.

32 Fuhr N, Grossjohann K. XIRQL: a query language for information retrieval in XML documents. In: SIGIR'01. New York, NY: ACM; 2001. p. 172-180.

33 Fuller M, Mackie E, Sacks-Davis R, Wilkinson R. Structured answers for a large structured document collection. In: SIGIR '93. New York, NY: ACM; 1993. p. 204-213.

34 Furnas GW, Deerwester S, Dumais ST, Landauer TK, Harshman RA, Streeter LA, Lochbaum KE. Information retrieval using a singular value decomposition model of latent semantic structure. In: Proceedings of the 11th annual international ACM SIGIR conference on Research and development in information retrieval. New York, NY: ACM; 1988. p. 465-480.

35 Furnas G, Landauert T, Gomez L, Dumais S. The vocabulary problem in human-system communication. Communications of the ACM. 1987:964-971.

36 Govert N, Abolhassani M, Fuhr N, Grobjohann K. Content-oriented XML retrieval with HyRex. In: INEX Workshop. 2002. p. 26-32.

37 Grabs T, Scheck HJ. Flexible information retrieval from xml with PowerDB XML. In: INEX Workshop. 2002. p. 26-32.

38 Grossman DA, Frieder O. Information Retrieval: Algorithms and Heuristics. Norwell, MA: Kluwer Academic Publishers; 1998.

39 Grust T. Accelerating XPath location steps. In: SIGMOD'02. New York, NY: ACM; 2002. p. 109-120.

40 Guo L, Shao F, Botev C, Shanmugasundaram J. Xrank : Ranked search over XML documents. In: SIGMOD'03. New York, NY: ACM; 2003. p. 16-27.

41 Gutierrez A, Motz R, Viera D. Building Databases with Information Extracted from Web Documents. In: Proceedings of the XX International Conference of the Chilean Computer Science Society. IEEE Computer Society; 2000. p. 41-.

42 Harter S. Psychological relevance and information science. Journal of the American Society for Information Science (JASIS). 1992:602–615.

43 Harter S, Hert C. Evaluation of information retrieval systems: approaches, issues and methods. In: Williams ME. Annual Review of Information Science and Technology. Vol 32. 1997. p. 3-94.

44 Hatano K, Kinutani H, Yoshikawa M, Uemura S. Information Retrieval System for XML Documents. In: Database and Expert Systems Applications. Berlin: Springer; 2002. p. 5-33.

45 Haveliwala TH. Topic-sensitive PageRank. In: Proceedings of the 11th international conference on World Wide Web. New York, NY: ACM; 2002. p. 517-526.

46 Huck G, Macherius I, Fankhauser P. PDOM : Lightweigt persistency support for the document object model. In: Succeeding with Object. John Wiley; 2000. p. 107-118.

47 Jang H, Kim Y, Shin D. An effective mechanism for index update in structured documents. In: Proceedings of the eighth international conference on Information and knowledge management. New York, NY: ACM; 1999. p. 383-390.

48 Jeh G, Widom J. Scaling personalized web search. In: Proceedings of the 12th international conference on World Wide Web. New York, NY: ACM; 2003. p. 271-279.

49 Jones CB, Purves R, Ruas A, Sanderson M, Sester M, Van Kreveld M, Weibel R. Spatial information retrieval and geographical ontologies an overview of the SPIRIT project. In: Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval. New York, NY: ACM; 2002. p. 387-388.

50 Kakade V, Raghavan P. Encoding XML in Vector Spaces. In: ECIR'05. Berlin: Springer; 2005. p. 96-111.

51 Kamps J, de Rijke M, Sigurbjornsson B. Length normalization in XML. In: SIGIR'04. New York, NY: ACM; 2004. p. 80-87.

52 Kamps J, Pehcevski J, Kazai G, Lalmas M, Robertson S. INEX 2007 Evaluation Measures. In: INEX'07. Berlin: Springer-Verlag; 2007. p. 24-33.

53 Kamvar SD, Haveliwala TH, Manning CD, Golub GH. Exploiting the Block Structure of the Web for Computing PageRank. Technical Report. Stanford; 2003.

54 Kanne CC, Moerkotte G. Efficient Storage of XML Data. In: Proceedings of the 16th International Conference on Data Engineering. Washington, DC: IEEE Computer Society; 2000. p. 198-.

55 Kazai G, Lalmas M. INEX 2005 evaluation measures. In: Proceedings of the 4th international conference on Initiative for the Evaluation of XML Retrieval. Berlin: Springer-Verlag; 2006. p. 16-29.

56 Kazai G, Lalmas M, Roelleke T. Focused document retrieval. In: 9th International Symposium on string processing and information. Lisbon 2002.

57 Kleinberg JM. Authoritative sources in a hyperlinked environment. Journal of the ACM. 1999:604-632.

58 Kolda T, Bader B. The tophits model for higher-order web link. In: Workshop on Link Analysis, Counterterrorism and Security. Vol 7. 2006. p. 26-29.

59 Kwok KL. A neural network for probabilistic information retrieval. SIGIR Forum. 1989:21-30.

60 Lalmas M. Dempster-Shafer's theory of evidence applied to structured documents: modelling uncertainty. SIGIR Forum. 1997:110-118.

61 Lalmas M, Kazai G, Kamps J, Pehcevski J, Piwowarski B, Robertson S. INEX 2006 Evaluation Measures. In: INEX'06. Springer-Verlag; 2006. p. 20-34.

62 Lelu A, François C. Information retrieval based on a neural unsupervised extraction of thematic fuzzy clusters. Les réseaux Neuro-mimétiques et leurs applications (Neuro Nimes). 1992:93-104.

63 Lempel R, Moran S. SALSA: the stochastic approach for link-structure analysis. ACM Transactions on Information Systems. 2001:131-160.

64 Levy A, Fernandez M, Suciu D, Florescu D. XML-QL: A query language for XML. W3C technical report. 1998.

65 Luk RWP, Leong HV, Dillon TS, Chan ATS, Croft WB, Allan J. A survey in indexing and searching XML documents. Journal of the American Society for Information Science and Technology. 2002:415-437.

66 Marchiori M. The quest for correct information on the Web: Hyper search engines. Computer Networks and ISDN Systems. 1997:1225-1235.

67 Maron ME, Kuhns JL. On Relevance, Probabilistic Indexing and Information Retrieval. Journal of the ACM. 1960:216-244.

68 Mass Y, Mandelbrod M, Amitay E, Carmel D, Maarek Y, Soffer A. JuruXML - an XML Retrieval System at INEX'02. In: INEX Workshop. 2002. p. 73-80.

69 Mattaoui M, Mezghiche M. Prise en compte des liens pour améliorer la recherche d'information structurée. In: CORIA 2009. Alger 2009. p. 363-372.

70 Mizzaro S. Relevance, the whole story. Journal of the American Society for Information Science. 1997:810–832.

71 Noreault T, McGill M, Koll MB. A performance evaluation of similarity measures, document term weighting schemes and representations in a Boolean environment. In: Oddy RN, Robertson SE, Van Risjbergen CJ, Williams PW. Proceedings of the 3rd annual ACM conference on Research and development in information retrieval. London: Butterworth & Co; 1981. p. 57-76.

72 Ounis I, Amati G, Plachouras V, He B, Macdonald C, Liona C. Terrier: A high performance and scalable Information Retrieval platform. In: ACM SIGIR OSIR Workshop 2006. Seattle, WA 2006.

73 Piwowarski B. Expected ratio of relevant units : A measure for structured information retrieval. In: INEX'03. 2003.

74 Piwowarski B, Faure GE, Gallinari P. Bayesian networks and INEX. In: INEX'02. 2002.

75 Ponte JM, Croft WB. A language modeling approach to information retrieval. In: Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval. New York, NY: ACM; 1998. p. 275-281.

76 Porter MF. An algorithm for suffix stripping. In: Readings in information retrieval. San Francisco, CA: Morgan Kaufmann Publishers Inc.; 1997. p. 313-316.

77 Robertson SE. The probability ranking principle in IR. Journal of Documentation. 294-304.

78 Robertson SE, Sparck Jones K. Relevance weighting of search terms. Journal of American Society of Information Science, JASIS. 1976:129-146.

79 Robertson SE, Walker S. On relevance weights with little relevance information. SIGIR. 1997:16–24.

80 Robie J, Lapp J, Schach D. XML Query Language (XQL). In: W3C QL'98 (Query Languages 98). 1998.

81 Rolleke T, Lalmas M, Kazai G, Ruthven I, Quicker S. The Accessibility Dimension for Structured Document Retrieval. In: Proceedings of the 24th BCS-IRSG European Colloquium on IR Research: Advances in Information Retrieval. London: Springer-Verlag; 2002. p. 284-302.

82 Salton G. Automatic text processing: the transformation, analysis, and retrieval of information by computer. Boston, MA: Addison-Wesley Longman Publishing Co., Inc.; 1989.

83 Salton G. The SMART retrieval system: Experiments in automatic document processing. Englewood Cliffs, NJ: Prentice-Hall; 1971.

84 Salton G, Fox E, Wu H. Extended Boolean information retrieval. Communications of the ACM. 1983:1022-1036.

85 Salton G, McGill JM. Introduction to Modern Information Retrieval. New York, NY (NY): McGraw-Hill, Inc.; 1986.

86 Salton G, Wong A, Yang CS. A vector space model for automatic indexing. Communications of the ACM. 1975:613–620.

87 Saracevic T. Relevance reconsidered. International Conference on Conceptions of Library and Information Science (COLIS). 1996:201-218.

88 Sauvagnat K. Modèle flexible pour la Recherche d'Information dans des corpus de documents semi-structurés. Thèse de doctorat. Toulouse: Université Paul Sabatier; 2005.

89 Sauvagnat K, Boughanem M. A la Recherche de noeuds informatifs dans des corpus de documents XML. In: CORIA. 2005. p. 119-134.

90 Sauvagnat K, Boughanem M. The impact of leaf nodes relevance values evaluation in a propagation method for XML retrieval. In: Joint Workshops on XML, IR and DB. Sheffield: University of sheffield; 2004. p. 19-22.

91 Savoy J, Rasolofo Y. Report on the TREC-9 experiment: Link-based retrieval and distributed collections. In: Proceedings of TREC-9. Gaithersburg, MD: NIST; 2000. p. 579-588.

92 Schlieder T, Meuss H. Querying and ranking XML documents. Journal of American Society for Information Science and Tecnology. 2002:489-503.

93 Shakery A, Zhai CX. Relevance Propagation for Topic Distillation UIUC TREC 2003 Web Track Experiments. In: TREC. 2003. p. 673-677.

94 Shin D, Jang H, Jin H. BUS: an effective indexing and retrieval scheme in structured documents. In: Proceedings of the third ACM conference on Digital libraries. New York, NY: ACM; 1998. p. 235--243.

95 Signore O, Garibald AM, Greco M. Proteus: a concept browsing interface towards conventional Information Retrieval System Database and Expert System Applications. In Proceedings of DEXA. 1992.

96 Singhal A, Buckley C, Mitra M. Pivoted document length normalization. SIGIR. 1996:21–29.

97 Smith M. Aspects of the p-norm model of information retrieval: Syntactic query generation, efficiency, and theoretical properties.. Technical report. Ithaca, NY: Cornell University; 1990.

98 Sparck Jones K. Reflections on TREC. Information Processing and Management. 1995:291-314.

99 Tamine L. Optimisation de requêtes dans un système de recherche d'information. Thèse de doctorat. Toulouse: Université Paul Sabatier; 2000.

100 Tebri H. Formalisation et spécification d'un système de filtrage incrémental d'information. Thèse de doctorat. Toulouse: Université Paul Sabatier; 2004.

101 Theobald A, Weikum G. The Index-Based XXL Search Engine for Querying XML Data with Relevance Ranking. In: EDBT'02. London: Springer-Verlag; 2002. p. 477-495.

102 Turtle H, Croft WB. Evaluation of an inference network-based retrieval mode. ACM Transactions on Information Systems. 1991:187-222.

103 Van Rijsbergen CJ. Information Retrieval. 2nd ed. London: Butterworth-Heinemann; 1979.

104 Weigel F, Schulz KU, Meuss H. Ranked retrieval of structured documents with the s-term vector space model. In: INEX'04. Berlin: Springer-Verlag; 2005. p. 238-252.

105 Wolff JE, Florke H, Cremers AB. Searching and browsing collections. In: IEEE advances in digital libraries. 2000. p. 141-150.

106 Wolff JE, Florke H, Cremers AB. Searching and Browsing Collections of Structural Information. In: ADL'00. Washington, DC: IEEE Computer Society; 2000. p. 141-150.

107 Wong RK, Lam F, Shui WM. Querying and maintaining a compact XML storage. In: Proceedings of the 16th international conference on World Wide Web. New York, NY: ACM; 2007. p. 1073-1082.

108 Wong SKM, Ziarko W, Raghavan VV, Wong PCN. On modeling of information retrieval concepts in vector spaces. ACM Transactions on Information Systems. 1987:229-321.

109 Wong SKM, Ziarko W, Wong PCN. Generalized vector spaces model in information retrieval. In: Proceedings of the 8th annual international ACM SIGIR conference on Research and development in information retrieval. New York, NY: ACM; 1985. p. 18-25.

110 Yager R. On ordered weighted averaging aggregation operators in multicriteria decision-making. IEEE Transactions on Systems Man and Cybernetics. 1988:183-190.

111 Yates B, Neto R. Modern Information Retrieval. New York, NY: ACM Press, Addison Wesley; 1999.

112 Zadeh LA. Fuzzy Sets. Information and Control. 1965:338-353.

113 Zargayouna H. Contexte et sémantique pour une indexation de documents. In: CORIA'04. Toulouse 2004. p. 161–178.

114 Zhang J. Visualization for Information Retrieval. Berlin: Springer; 2008.

115 Zhang J, Rasmussen EM. Developing a new similarity measure from two different perspectives. Information Processing and Management. 2001:279-294.