REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université Mouloud MAMMERI de Tizi-Ouzou

Faculté de génie électrique et informatique
Département informatique

MEMOIRE

MEMOIRE

En vue de l'obtention du diplôme de Master en informatique

Option : Systèmes Informatiques

Thème

Recherche d'information sémantique dans les documents XML

Présenté par :

M^elle^ HAMDI Souhila

M^elle^ HENNOUS Souhila

Proposé et dirigé par :

M^me^ AMIROUCHE Fatiha

Année Universitaire
2014/2015

Résumé

Avec le développement du web, les documents ont connu un véritable essor, ce qui a produit une variété dans son contenu (texte, images et son). Dans les systèmes de recherche d'information classiques, les documents sont considérés comme étant des unités atomiques indépendantes les unes des autres et constituées d'un ensemble de mots et de phrases. L'avènement de nouveaux standards de représentation des documents, et particulièrement XML, a poussé la communauté de RI à exploiter la richesse présente dans ces documents et à développer de nouveaux concepts pour l'indexation et l'interrogation du corpus XML, puisque XML tend à former la majorité des documents numériques mis à disposition des utilisateurs. La réponse fournie à l'utilisateur n'est plus un document entier mais des parties de document apportant une information pertinente à un besoin utilisateur, puisque les documents semi-structurés peuvent être représentés sous forme d'arbre, et le but est alors de trouver les sous-arbres de tailles minimales répondant à la requête.

Dans un autre côté, les SRI classique reposent sur l'indexation par les mots clés des documents et requêtes, donc seuls les documents qui partagent les mots clés avec la requête sont renvoyés ce qui réduit la précision des résultats. L'indexation sémantique tente de pallier à ce problème par une représentation par concepts.

L'objectif de notre mémoire est d'implémenter le modèle LSI qui permet la recherche sémantique dans les documents XML. Pour ce faire, on a utilisé le système XFIRM (XML Flexible Information Retrieval Model) qui repose sur : Un modèle de représentation des données générique, qui permet de modéliser des documents possédant des structures différentes ; Un langage de requête flexible, permettant à l'utilisateur d'exprimer son besoin selon divers degrés de précision, en exprimant ou non des conditions sur la structure des documents ; Un modèle de recherche basé sur une méthode de propagation de la pertinence.

Mots Clés : recherche d'information, XML, indexation sémantique latente (LSI), documents semi-structurés.

Remerciements

Nous remercions tout d'abord Dieu qui nous a donné la foi et le courage pour accomplir ce projet.

Nous tenons aussi à exprimer notre reconnaissance et profonde gratitude à notre promotrice, M^me^ Amirouche pour nous avoir encadrés durant cette année, pour sa forte présence et sa disponibilité, pour son exigence scientifique et ses précieuses orientations méthodologiques, pour son encouragement et sa patience.

Que les membres du jury trouvent ici nos plus vifs remerciements pour avoir accepté d'honorer par leur jugement notre travail. Aussi, nous adressons nos remerciements à tous nos enseignants de l'UMMTO pour nous avoir appris le goût de l'effort et du travail.

Un grand merci aussi à toute personne qui nous a aidées ou qui a contribué de prés ou de loin à la réalisation de ce projet.

Dédicace

J'ai le plaisir de dédie ce travail à :

- *La mémoire de mes grands parents, que Dieu leur accorde sa miséricorde.*
- *Deux âmes les plus chères au monde mon père et ma mère qui m'ont toujours soutenu et accompagné durant ma vie. C'est grâce à leurs encouragements que je suis arrivé à ce stade, que Dieu vos offre la paix et le bonheur et vous garde en santé.*
- *Mes chers frères: Ahcene, Khelifa et a sa femme Nadia ainsi que leurs petits anges Cirine et Fares.*
- *Mes chères sœurs : Zakia et Souraya.*
- *Toute la famille : Hennous et Khelil.*
- *A tous mes amies, notamment ceux du département d'Informatique.*
- *Ma chère binôme : Souhila*
- *Toute la promotion 2015.*

HENNOUS SOUHILA

Dédicace

J'ai le plaisir de dédie ce travail à :

- *Deux âmes les plus chères au monde mon père Youcef et ma mère Fatma que dieu nous les préserve grâce à leurs Amour, leur encouragements, leur patience et leur sacrifices que j'ai peut arriver à ce stade, que Dieu vos offre la paix et le bonheur et vous garde en santé.*
- *Mes chères grands-mères à qui je souhaite une longue vie.*
- *Mes chers frères: Boualem,Mohand, Nassim et Mouloud*
- *Mes chères sœurs : Samira et Ania*
- *Mes petits neveux : Amoukrane, Ahmed, Yanis et Youcef*
- *Ma petite belle nièce : Amilia.*
- *Toute la famille Hamdi et Laouar.*
- *A tous mes amies, notamment Dyhia, Soraya, Marilia et Zazi.*
- *Ma chère binôme : Souhila*
- *Toute la promotion 2015.*

HAMDI SOUHILA

Sommaire

Chapitre 4: Expérimentations et résultats

Liste des figures :

Liste des tableaux

Introduction générale

Introduction générale

1. Contexte du travail :

La RI, née en 1950, permet de définir des modèles et des systèmes pour faciliter l'accès à un ensemble de documents sous forme électronique (corpus de documents), afin de permettre aux utilisateurs de retrouver les documents dont le contenu répond à leur besoin d'information.

Les documents ont connu une véritable croissance avec le développement du web, qui dispose d'un contenu très varié, constitué de texte, d'images et de son. En raison de cette diversité de masses documentaire, l'utilisateur trouve de plus en plus de difficultés pour accéder aux informations qui répondent à son besoin. Pour cela, le W3C (*World Wide Web Consortium*) a mis en œuvre des nouveaux standards de représentation des documents dont le plus populaire aujourd'hui est l'XML (*eXtensible Markup Language*). Ce dernier permet de combiner l'information structurelle avec l'information textuelle.

L'avènement de XML a poussé la communauté de RI à exploiter la richesse présente dans ces documents et à développer de nouveaux concepts pour l'indexation et l'interrogation du corpus XML. En effet la recherche d'information a évolué de l'accès à un document ou un ensemble de documents vers l'accès à des informations répondant à un intérêt particulier de l'utilisateur.

Les travaux réalisés dans le cadre de recherche d'information dans des documents semi-structurés XML sont basées sur l'indexation par des mots clés selon leurs fréquences d'apparition dans les documents ou les éléments du document. Ainsi ils ne tiennent pas compte de la sémantique des mots dans leurs contextes d'apparition. Cela permet de renvoyer un document non pertinent, bien que le document satisfasse la requête. Notre travail se situe dans le contexte de recherche d'information sémantique dans les documents semi-structurés de type XML.

2. Problématique :

Le nombre de documents structurés ou semi-structurés mis à notre disposition est toujours en croissance. Pour cela il faut prévoir de nouveaux systèmes capables d'indexer et d'effectuer des recherches dans tels volumes d'informations. De plus, on doit garantir à l'utilisateur la possibilité de retrouver l'information qu'il recherche avec un minimum d'efforts.

Comme l'indexation par des mots clés peut être ambiguë, Un même mot utilisé dans la requête et le document peut définir des sens différents (polysémie et homonymie), et plusieurs mots lexicalement différents utilisés dans le document et la requête peuvent refléter un même sens (synonymie). D'autre part, la recherche peut échouer si les termes de la requête n'y apparaissent pas dans le document, car un document est considéré pertinent pour la requête s'il existe des mots clés en commun avec cette requête. De ce fait des documents pourtant non pertinents contenant des mots de la requête, sont retrouvés, tandis que les documents sémantiquement

pertinents ne contenant aucun mot de la requête, ne sont pas retrouvés. La problématique qui se pose alors est : comment exploiter l'information sémantique pour améliorer la réponse apportée à l'utilisateur ?

Des travaux ont été entrepris dans ce sens et ont montré leur intérêt. Ces travaux se basent sur une indexation sémantique qui permet la representation des documents et des requêtes par des concepts (sens des mots). Ces derniers sont extraits à partir de ressources linguistiques(thesarus,…).

Dans ce mémoire, nous proposons un modèle de RI sémantique dans les documents XML. Notre proposition consiste à étendre le modèle LSI (Latent Semantic Indexing) proposé en RI classique pour qu'il prenne en charge les documents semi-structurés de type XML.

3. Contribution :

Notre contribution dans le cadre de la RI structurée se situe à quatre niveaux :

1. Réalisation d'un état de l'art sur la recherche d'information classique et structuré, on y pressente ainsi les travaux relatifs à la prise en compte de la sémantique dans la recherche d'information.
2. Proposition d'un nouveau modèle pour la prise en compte de la sémantique dans la recherche d'information.
3. Implémentation de ce modèle sur la plateforme XFIRM.
4. Evaluation de ce modèle sur la collection INEX 2006.

4. Organisation de mémoire :

Ce mémoire est organisé en quatre chapitres et quatre annexes.

Dans le premier chapitre nous introduisons dans la première section la recherche d'information classique. 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 la deuxième section nous présentons la recherche d'informations semi-structurées dans les documents XML. En commençant par la présentation de format XML qui permet de prendre en considération l'information textuelle et l'information structurelle dans un même document. Ensuite nous identifions les problématiques qu'engendre la prise en compte de la structure dans le processus de RI. Ainsi nous présentons les solutions à ces problématiques, qui constituent à modifier l'indexation et à définir de nouveaux langages de requête et modèles de RI.

Dans le deuxième chapitre, nous présentons la recherche d'information sémantique dans les documents XML. En commençant par l'identification des problématiques engendrées par l'introduction de la sémantique à l'indexation des documents semi-structurée. Ainsi nous présentons les solutions à ces problématiques qui constituent à la représentation des documents et des requêtes par des concept (sens des mots). Qui sont extrait par des ressources linguistiques(thesarus,…) ou des méthodes de désambiguisation des sens des mots.

Pour conclure, nous présentons la compagne d'évaluations INEX utilisé pour l'évaluation des SRIS (Système de Recherche d'Information Structuré).

Dans le troisième chapitre, nous présentons notre modèle pour la prise en compte de la sémantique dans la recherche d'information. On décrit le processus d'indexation, ainsi la mesure de similarité utilisé entre la requête et les éléments de documents XML.

Dans le quatrième chapitre, nous présentons le système de recherche XFIRM. Ainsi notre implémentation et les résultats de l'évaluation de notre modèle sur la collection INEX 2006.

Pour finir, nous présontons dans la premiére annexe les technologies de la famille XML, dans le deuxiéme, les mesures de simélarité sémantique, dans le troisième annexe nous représentons le système de recherche XFIRM et dans le dernier annexe un exemple de calcul SVD.

Chapitre1

La recherche d’information dans les documents XML

Chapitre1 La recherche d'information dans les documents XML

Introduction :

La recherche d'information est une branche de l'informatique qui s'intéresse à l'acquisition, l'organisation, le stockage, la recherche et la sélection de documents permettant de répondre de façon pertinente aux besoins exprimés par un utilisateur à travers des requêtes **[Boughanem et al., 2008].**

Les types des documents mis à la disposition des utilisateurs évoluent au rythme des évolutions technologiques et de l'information. Et alors qu'avant les utilisateurs traitaient et utilisaient des documents textes « plats » (i.e. non structurés), aujourd'hui avec l'avènement du web et la création des langages de balisage dont le plus populaire est XML (eXtensible Markup Language), les documents manipulés et traités par les utilisateurs sont des documents semi-structurés.

Du point de vue des systèmes de Recherche d'Information (SRI), l'accès aux documents textes « plats » ou semi-structurés de type « XML » soulève des nouvelles problématiques liées à l'indexation, à l'appariement document/requête et à l'interrogation des documents.

Ce chapitre a pour objectif de présenter les différentes problématiques soulevées par la RI classique et la RI semi-structurée, ainsi que les différentes solutions proposées pour tenir compte de la spécificité de chaque type du document.

I. La Recherche d'information classique :

I.1. Introduction :

La RI classique s'intéresse au traitement des documents plats. Dans cette section, nous présentons les concepts de base de la RI classique, puis nous passons en revue les modèles de recherche ainsi que les approches d'évaluation des SRI.

I.2. Concepts de base de la recherche d'information classique:

La tâche principale d'un système de recherche d'information (SRI) est de sélectionner dans une collection de documents préalablement stockée, l'ensemble des documents pertinents au besoin informationnel de l'utilisateur exprimé sous forme de requête.

Cette définition fait apparaître quatre notions clés qu'il convient de préciser : documents, base documentaire, requête, pertinence.

i. Document :

On appelle document toute unité qui peut constituer une réponse à une requête utilisateur. Le document peut être un texte, un morceau de texte, une page Web, une image, une vidéo, etc.

ii. Base documentaire :

Une base documentaire (corpus documentaire) est une collection de documents de différentes structures et à contenus divers (textes, images, pages web, vidéos…etc.).

iii. Les requêtes :

Une requête est l'expression formelle du besoin en information de l'utilisateur. Une requête peut être exprimée en :

- **Langage naturel :** sous forme de phrases et aucune contrainte n'est fixée a priori.
- **Langage contrôlé :** sous forme d'une liste de descripteurs (ou mots clés).

iv. La pertinence :

C'est une notion fondamentale en RI. Elle peut être définie comme une mesure d'informativité du document à la requête**.**

La pertinence peut être classée en deux types :

La pertinence système ou algorithmique : définie à travers le modèle de RI. Elle est formellement mesurée par un score de pertinence (ou RSV) qui est attribué par le SRI à chaque document de la base relativement à la requête. Cette pertinence est objective, déterministe.

La pertinence utilisateur : C'est une mesure subjective qui représente la satisfaction de l'utilisateur vis-à-vis des documents retournées par le système. Cette pertinence est évolutive car un document jugé non pertinent à l'instant t pour une requête peut être jugé pertinent à l'instant t+1 car la connaissance de l'utilisateur sur le sujet aura évolué.

L'objectif de tout SRI est de rapprocher la pertinence système de la pertinence utilisateur.

I.3. Le processus de recherche d'information :

Un SRI met en œuvre un processus de recherche qui a pour rôle de retrouver le maximum de documents pertinents pour une requête utilisateur. Ce processus intègre trois fonctions principales : l'indexation, l'appariement et la reformulation de la requête.

La figure suivante illustre le déroulement du processus de recherche, dit processus en U de la recherche d'information.

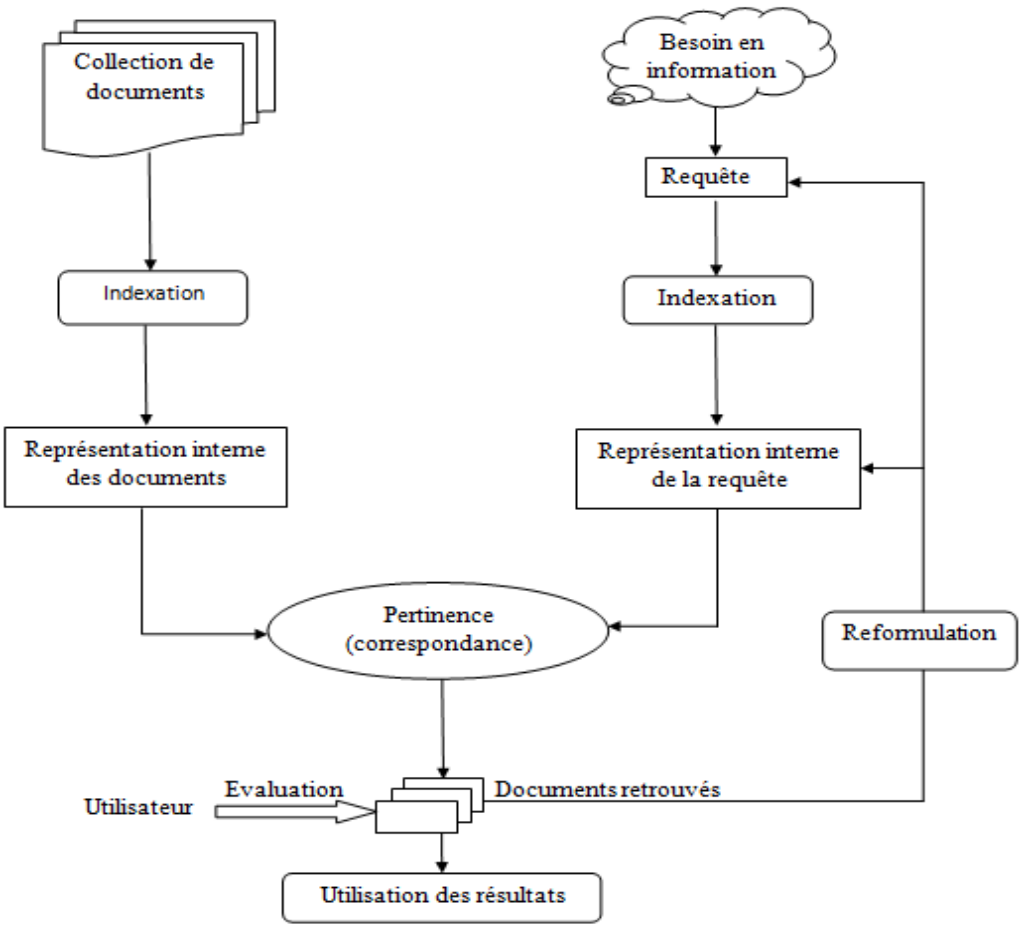


Figure 1.1 Processus en U de la RI **[Boubekeur, 2008]**

Dans ce qui suit, nous présentons chacune de ces fonctions.

I.3.1 L'indexation :

Dans un processus de RI, la requête et les documents sont difficilement exploitables à l'état ordinaire. Afin de rendre la recherche possible, une étape préliminaire, dite l'indexation, est nécessaire.

L'indexation permet de décrire les documents et la requête par un ensemble de termes ou de descripteurs. Ces descripteurs reflètent au mieux le contenu informationnel du document ou de la requête. De la qualité de l'indexation dépend en partie de la qualité de réponse du SRI. Par conséquent, pour avoir une réponse satisfaisante du système, l'indexation doit être efficace.

Différentes approches d'indexation sont utilisées comme suit:

I.3.1.1 l'indexation manuelle :

Cette approche d'indexation fait intervenir des documentalistes pour identifier les descripteurs pertinents qui caractérisent le mieux les documents. Ce mode d'indexation dépend de l'indexeur qui doit avoir une connaissance minimale des contenus des documents pour pouvoir choisir les informations qui les caractérisent, ce qui induit la subjectivité de ses

résultats car différents spécialistes peuvent indexer un document avec des termes différents. De plus, cette approche est lente et coûteuse, ce qui la rend inadaptée dans les bases documentaires de grandes tailles.

I.3.1.2 Indexation automatique :

C'est la technique d'indexation la plus utilisé lors de processus de RI. La construction des descripteurs se fait d'une manière automatique et rapide à l'aide d'un programme informatique. Cette approche est donc plus adaptée pour les bases documentaires de grandes tailles.

L'indexation automatique repose sur les étapes suivantes :

- **L'analyse lexicale :**

L'analyse lexicale est le processus qui permet de convertir le texte d'un document en un ensemble d'entités lexicales ou termes. Un terme est composé d'un ou plusieurs mots liés. Un mot est défini comme étant une suite de caractères délimitée par des séparateurs (blancs, virgules, …).

- **L'élimination des mots vides :**

Les mots vides (articles, proposition, conjonction, etc.) sont des mots non significatifs dans un document, car ils ne traitent pas le sujet du document. Lors de l'indexation, les mots sont éliminés de l'index.

On distingue deux techniques pour éliminer les mots vides :

a. L'utilisation d'une liste préétablie de mots vides (aussi appelée anti-dictionnaire ou stoplist) : si un terme appartient à la liste des mots vides alors il est exclu de l'index.

b. L'élimination des mots ayant une fréquents qui dépasse un certain seuil dans la collection.

L'élimination des mots vides réduit la taille de l'index, ce qui améliore le temps de réponse du SRI**.**

- **La normalisation:**

Dans le document les mots peuvent apparaître sous différentes formes. La normalisation des mots a pour but de réduire les variations morphologiques d'un mot en regroupant ces différentes variantes sous une même forme. La normalisation peut se faire selon l'une des techniques suivantes : la lemmatisation ou la troncature (racinisation ou stemming).

a. La lemmatisation :

La lemmatisation consiste à prendre la forme canonique du mot. L'algorithme de Porter 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.

b. La troncature :

La troncature permet de construire le radical d'un mot (stemme, racine), c'est-à-dire la forme qui reste après avoir supprimé (coupé) les préfixes et/ou suffixes.

Le stemme contrairement au lemme ne correspond pas à un mot réel de la langue. A titre d'exemple, en supposant une troncature à 7 caractères, on pourra réduire :

Proposer →propose
Proposition →proposi
Propositionnel→proposi

- **La pondération des termes**

La pondération des termes est une fonction essentielle en RI. La pondération a pour but d'affecter à chaque terme t_i d'un document d_j un poids w_{ij} qui exprime son degré de représentativité dans le document.

Deux principales fonctions de pondération sont utilisées : une pondération locale et une pondération globale.

a. La pondération locale :

La pondération locale reflète l'importance locale du terme t_i dans le document d_j exprimé en fonction du nombre d'occurrences de ce terme dans le document (soit $f(t_i,d_j)$). Elle est noté tf (term frequency). Cette fréquence est formellement exprimée comme suit :

$$tf_{ij} = 1 + log(f(t_i,d_j))$$

b. La pondération globale :

Mesure l'importance d'un terme dans toute la collection. Cette pondération est notée idf (Inverse Document Frequency). Un poids plus important doit être assigné aux termes qui apparaissent moins fréquemment dans la collection. Car les termes qui apparaissent dans nombreux documents de la collection ne permettent pas de distinguer les documents pertinents des documents non pertinents. Cette mesure s'exprime ainsi :

$$Idf = log \frac{N}{n_i}$$

Où :

n_i : nombre de document contenant le terme considéré.

N : nombre de documents dans la collection.

En règle générale, l'utilisation de tf ou bien idf tous seuls n'est pas satisfaisante, pour cela on utilise plutôt la combinaison tf * idf qui constitue une bonne approximation du poids du terme dans les documents.

I.3.1.3 Indexation semi-automatique :

Dans cette approche, les résultats obtenus par une indexation automatique sont soumises à un documentaliste pour les valider ou pour les enrichir.

I.3.2 L'appariement :

Il consiste à mettre en correspondance la représentation de la requête avec les représentations des documents. Un degré de pertinence (ou de similarité) noté « RSV » entre la requête et chaque document est ainsi calculé. En se basant sur ce degré, les documents qui sont jugés (par le SRI) similaires à la requête, sont retournés à l'utilisateur.

Cette fonction est différente d'un modèle de recherche d'information à un autre. D'ailleurs un modèle de RI est caractérisé par sa fonction de similarité et son modèle de représentation de l'index.

I.3.3 La reformulation de la requête :

C'est la génération d'une nouvelle requête qui répond mieux au besoin informationnel de l'utilisateur par apport à sa requête initiale.

Deux méthodes de reformulation sont possibles :

Les méthodes locales :

Les méthodes locales s'appuient sur la technique de réinjection de pertinence (relevance feedback). Le principe de la réinjection de pertinence est de faire participer l'utilisateur dans le processus de recherche de sorte à améliorer l'ensemble final de résultats. Le procédé de base est le suivant :

- l'utilisateur exprime son besoin via une requête,
- le système renvoie un premier ensemble de résultats de recherche,
- l'utilisateur marque quelques documents retournés comme pertinents ou non pertinents,
- le système calcule une meilleure représentation du besoin en l'information sur la base de la rétroaction utilisateur,
- le système visualise un ensemble révisé de résultats de la recherche.

Les méthodes globales :

Se base sur l'utilisation des informations externes pour reformuler la requête initiale. Cette méthode consiste à enrichir la requête par des termes d'indexation initialement absents. En général, ces termes sont issus d'une ressource externe telle que : les bases lexicales, les thésaurus et les ontologies. La ressource définit des relations sémantiques entre des termes d'indexation, ces relations sont exploitées pour sélectionner des termes qui peuvent être ajoutés à la requête. Parmi ces relations on peut citer la synonymie, l'équivalence, l'hyperonymie, la spécification/ généralisation, etc.

I.4 Les modèles de la recherche d'information :

Les modèles de recherche d'information se basent sur les représentations de la requête et des documents issues de l'indexation.

Un modèle de recherche d'information a pour rôle de fournir une formalisation du processus de RI et un cadre théorique pour la modélisation de la mesure de pertinence.

Dans ce qui suit nous présentons les principaux modèles utilisés.

I.4.1 Le modèle booléen:

- **Le modèle booléen de base :**

Ce modèle est le plus simple des modèles de RI, il est basé sur la théorie des ensembles de l'algèbre de Boole.

Un document d_i est représenté par une conjonction logique des termes non pondérés qui constitue l'index du document. Une requête q de l'utilisateur est représenté par une expression logique, composée de termes reliés par des opérateurs logiques (et, ou, non).

Dans ce modèle, les poids des termes dans l'index sont binaires, c'est à dire soit 0 (terme absent dans document) soit 1 (terme présent dans le document). La fonction d'ordonnancement vérifie alors si l'index de chaque document d_j implique l'expression logique de la requête q_i. Le résultat de cette fonction est un score binaire décrit comme suit : RSV (q_i, d_j) = {1, 0}.

La mesure de pertinence document-requête est calculée comme suit :

RSV(Dj,qi)=1 si qi ∈ Dj ; 0 sinon,
RSV(Dj,qi∧ qj)=1 si RSV(Dj,qi)=1 et RSV(Dj,qj)=1 ; 0 sinon,
RSV(Dj,qi∨ qj)=1 si RSV(Dj,qi)=1 ou RSV(Dj,qj)=1 ; 0 sinon,
RSV(Dj, ¬qi)=1 si RSV(Dj,qi)=0 ; 0 sinon,

La mise en œuvre du modèle booléen est facile, mais il présente deux inconvénients majeurs. En premier lieu le système retourne une liste de documents non-ordonnée suit à une requête, car les termes sont pondérés de la même façon 1 ou 0 donc il est difficile de juger qu'un terme important que l'autre (les termes ont la même importance). En deuxième lieu, les utilisateurs trouvent une difficulté d'exprimer leurs besoins avec les opérateurs logiques c'est-à-dire manipulation très mal des opérateurs booléens.

- **Le modèle booléen étendu :**

Extension du modèle booléen de base qui permet d'intégrer des poids des termes issue de l'indexation dans l'expression de la requête et des documents.

Cela permet de pallier les problèmes du modèle de base en ordonnant les documents retrouvés par le SRI **[Boubekeur, 2008].**

L'appariement requête-document est déterminé par les relations introduites dans le modèle p-norm basées sur les *p*-distances, avec $1 \leq p \leq \infty$. La valeur de p est indiquée au moment de la requête.

Si *m* est le nombre de termes dans la requête, les fonctions de similarité se calculent comme suit :

$$RSV(d, \mathrm{Q_{ou}}) = \left(\frac{x_1^p + x_2^p + \ldots + x_m^p}{m}\right)^{\frac{1}{p}}$$

$$RSV(d, \mathrm{Q_{et}}) = \left(\frac{(1-x_1)^p + (1-x_2)^p + \ldots + (1-x_m)^p}{m}\right)^{\frac{1}{p}}$$

Si p = 1, le modèle p-norm se ramène au modèle booléen de base.
Si p = ∞, le modèle, p-norm se ramène au modèle booléen flou (le prochain modèle).

L'avantage majeur de modèle booléen étendu par rapport au modèle de base est de pouvoir classer les documents selon leurs pertinences. Par ailleurs les requêtes restent complexes et difficilement formulées par l'utilisateur.

- **Modèle booléen flou :**

Ce modèle est une extension de modèle booléen de base qui tient compte de la pondération des termes dans les documents. Le document est donc représenté par un ensemble de termes pondérés. La requête reste toujours une expression booléenne. L'évaluation de l'appariement d'un terme à une requête repose sur une des évaluations classiques proposées par Zadeh **[Zadeh et al., 1965]** dans le cadre des ensembles flous :

$RSV(d, q_i) = a_i$
$RSV(d, q_i \text{ AND } q_j) = \min(RSV(d, q_i), RSV(d, q_j))$
$RSV(d, q_i \text{ OR } q_j) = \max(RSV(d, q_i), RSV(d, q_j))$
$RSV(d, \text{NOT} q_i) = 1 - RSV(d, q_i)$

Où :
a_i est le poids du terme t_i de la requête *qi* dans le document d.

Ces évaluations permettent de présenter les résultats par ordre de pertinence. Cependant, elles ne reflètent pas parfaitement le sens des opérateurs booléens **[Zargayouna, 2005].**

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 est un terme quelconque). Ceci fait retourner des résultats inattendus pour certaines requêtes **[Sadi, 2012].**

I.4.2 Le modèle vectoriel :

- **Le modèle vectoriel de base :**

Le modèle vectoriel *(VSM : Vector Space Model*) est un modèle algébrique où les documents et les requêtes sont représentés par des vecteurs de poids dans l'espace vectoriel des termes d'index.

La figure suivante illustre de façon graphique d'un document d_1 et d'une requête q dans un espace associé à 3 termes.

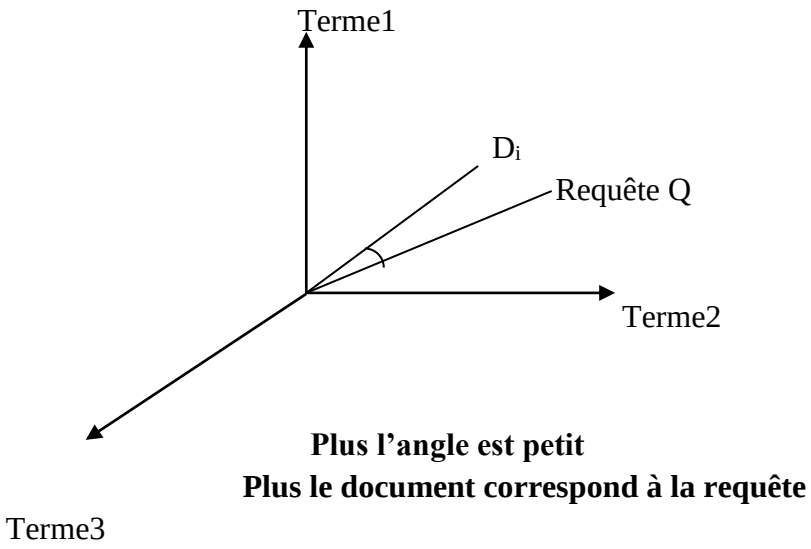


Figure 1.2 Exemple de représentation des documents dans un espace à 3 termes **[philippe,2007]**

Dans un espace d'information à m termes $T=\{t_1,t_2, \ldots, t_m\}$ un document d_i est représenté par un vecteur de poids w_{ij} de dimension m, dans l'espace vectoriel composé de tous les termes d'indexation. $D_i = (w_{i1}, w_{i2}, \ldots , w_{im})$.

Une requête q est aussi représentée par un vecteur de poids w_q défini dans le même espace vectoriel que le document. $q= (w_{q1} , w_{q2} , \ldots , w_{qm})$

Où w_{qj} est le poids de terme t_j dans la requête q, et w_{ij} son poids dans le document d_i.

Le modèle vectoriel prend en compte le poids de terme dans le document .Ce dernier peut être soit :

- une forme de tf*idf,
- un poids attribué manuellement par l'utilisateur.

La pertinence du document d_i pour la requête q est mesurée par le degré de corrélation de leurs vecteurs correspondants. Cette corrélation peut être exprimée par l'une des mesures suivantes :

Mesures	Formules
Le produit scalaire	$RSV(q,d_i)=\sum_{j=1}^{\lvert T\rvert} w_{qj}.w_{ij}$
La mesure de cosinus	$RSV(q,d_i)=\frac{q.d_i}{\lVert q\rVert.\lVert d_i\rVert}=\frac{\sum_{j=1}^{\lvert T\rvert} w_{qj}.w_{ij}}{\sqrt{\sum_{j=1}^{\lvert T\rvert} {w_{qj}}^2 \sum_{j=1}^{\lvert T\rvert} {w_{ij}}^2}}$
La mesure de Dice	$RSV(q,d_i)=\frac{2\times\sum_{j=1}^{\lvert T\rvert} w_{qj}.w_{ij}}{\sum_{j=1}^{\lvert T\rvert} {w_{qj}}^2+\sum_{j=1}^{\lvert T\rvert} {w_{ij}}^2}$
La mesure de Jaccard	$RSV(q,d_i)=\frac{\sum_{j=1}^{\lvert T\rvert} w_{qj}.w_{ij}}{\sum_{j=1}^{\lvert T\rvert} {w_{qj}}^2+\sum_{j=1}^{\lvert T\rvert} {w_{ij}}^2-\sum_{j=1}^{\lvert T\rvert} w_{qj}.w_{ij}}$

Tableau 1.1 les mesures de similarité utilisées dans le modèle vectoriel **[hammache, 2013].**

Le modèle vectoriel de base est l'un des modèles les plus utilisés en RI. Son avantage, par apport au modèle booléen de base, réside dans sa capacité d'ordonner les résultats de la recherche selon leur degré de pertinence pour une requête d'utilisateur. Cependant, ce modèle suppose que les termes d'index sont indépendants, et il ne tient pas compte des relations sémantiques qui peuvent exister entre ces termes dans le même document ou la même requête **[Azzoug, 2013].**

Le modèle vectoriel généralisé (Generalized Vector Space Model) permet de résoudre le problème d'indépendance des termes, cependant son efficacité n'a pas été démontrée.

- **Modèle LSI :**

L'idée de base du modèle LSI est que la correspondance entre un document et une requête donnée devrait être basée sur la correspondance des concepts plutôt que sur la correspondance des mots clés de l'index. Ce modèle a pour objectif de construire des index conceptuels portant sur la sémantique des documents (de donner une représentation conceptuelle des documents). Ainsi, les documents qui partagent des termes co-occurrents ont des représentations proches, ce qui permet de sélectionner un document même s'il ne contient aucun mot de la requête.

Ce modèle est fondé sur la construction d'un espace d'indexation de taille réduite k par l'application de la décomposition en valeurs singulières SVD (Singular Value Decomposition) de la matrice qui contient : en ligne les termes et en colonnes les documents. SVD permet d'une part de réduire l'espace des termes d'indexation, et d'autre part, de représenter les documents et les requêtes dans un espace qui ne dépend pas des termes d'indexation mais des concepts contenus dans les documents.

Une mesure de similarité standard entre les vecteurs dans l'espace réduit permet d'ordonnancer la pertinence des documents par rapport à la requête. Celle-ci, tout comme dans le modèle vectoriel de base, consiste en un ensemble de formules (produit scalaire, cosinus, mesure de Dice…) appliquées sur les nouveaux vecteurs de la requête et du document.

L'avantage principal du modèle LSI est son pouvoir de retrouver les documents pertinents pour une requête utilisateur même s'ils ne partagent aucun mot avec elle. Il permet en outre de résoudre partiellement les problèmes liés à la synonymie des mots dans la représentation des documents. Néanmoins, ce modèle perd son efficacité comparativement au modèle vectoriel de base, lorsque le nombre de documents est faible. En effet, une collection de petite taille donne une approximation erronée de la matrice documents-termes dans l'espace réduit.

I.4.3 Le modèle probabiliste :

- **Le modèle probabiliste de base :**

Le premier modèle probabiliste a été proposé par Maron et Kuhns au début des années 1960. Son principe de base consiste à estimer la probabilité de pertinence d'un document vis-à-vis d'une requête.

Il aborde la notion de probabilité de pertinence et de non-pertinence d'un document par rapport à une requête à travers « le principe de classement probabiliste » (Probability Ranking Principle PRP).

Etant donné une requête q et un document d, le modèle PRP peut être traduit comme suit : quelle est la probabilité que le document d soit pertinent pour la requête q?

Le modèle probabiliste tente d'estimer la probabilité que le document appartienne à la classe des documents pertinents (non pertinents). Un document est alors sélectionné si la probabilité qu'il soit pertinent à q notée (R/d), est supérieure à la probabilité qu'il soit non pertinent à q, notée $(\bar{R}/d)$. Le score d'appariement entre le document d et la requête q, noté $(d,$ q) est donné par **[Robertson, 1977]**:

$$RSV(d, Q) = \frac{P(R/d)}{P(\overline{R}/d)}$$

Le modèle probabiliste de base est un modèle tout à fait fonctionnel qui a démontré son efficacité en recherche d'information. Cependant, l'inconvénient du modèle probabiliste est de considérer les termes d'indexation comme étant indépendants les uns des autres, ce qui n'est généralement pas le cas.

- **Modèle de langue :**

Ce modèle est l'un des modèles probabilistes, qui est largement utilisés depuis 1998, en traitement du langage dans divers domaines : la reconnaissance de la parole, la traduction automatique, la recherche d'information…etc.

Son principe est de construire pour chaque document un modèle de langue noté M_d, ensuite, calcule la probabilité qu'une requête q puisse être générée par le modèle de langue du document d, noté$P(q/Md)$, qui sera considérée comme un score de pertinence de document vis-à-vis de la requête. Formellement :

$$RSV(d,q) = P(q = (t_1, t_2, \dots t_n)/M_d = \prod_i P(t_i/d)$$

Où :

M_d, le modèle de langue du document d,

P (q/M_d), la probabilité que la requête q soit générée par M_d.

La probabilité $P(t_i/d)$ est donnée par :

$$p(t_i/D) = \frac{tf(t_i/d)}{\sum_t tf(t/d)}$$

Où : $tf(t_i/d)$ est la fréquence du terme t_i dans le document 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 $P(q/Md)$ sera nulle. Pour remédier à cela, des techniques de lissage, dont le lissage de Laplace, le lissage de Good-Turing, le lissage Backoff, le lissage par interpolation sont utilisés. Leur principe consiste à assigner des valeurs non nulles aux termes qui n'apparaissent pas dans un document.

I.4.4 Modèle connexionniste :

Le modèle connexionniste est un modèle basé sur la théorie des réseaux de neurones dans la recherche d'information.

On distingue par réseaux de neurones, un certain nombre de modèles dont l'objectif est d'imiter quelques fonctions du cerveau humain en reproduisant certaines de ses structures de base.

Le fonctionnement du réseau se fait par propagation de signaux de la couche d'entrée vers la couche de sortie. Chaque neurone de la couche d'entrée reçoit une valeur d'activation, ensuite il calcule une valeur de sortie et il la transmet vers les neurones qui lui sont reliés dans la couche suivante.

Ce processus se reproduit afin d'atteindre la couche de sortie, les valeurs de sorties dans cette couche servant de critère de décision.

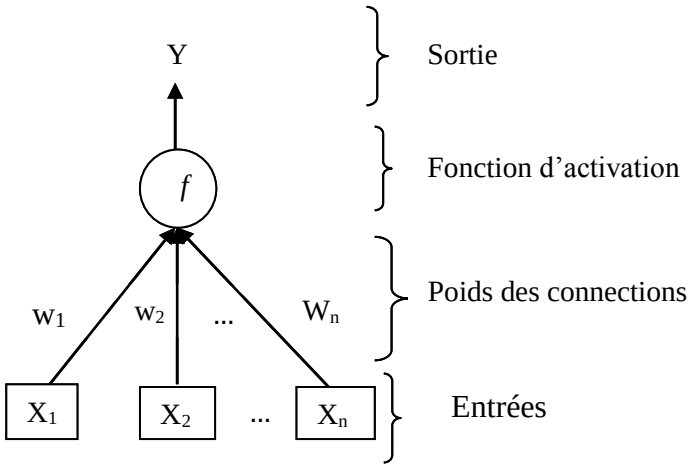


Figure 1.3 Représentation d'un neurone formel

La figure ci-dessous représente un modèle de réseaux de neurones pour la recherche d'information. Elle repose sur la modalisation par un réseau de neurones à trois couches interconnectées la couche requête, la couche termes et la couche documents. L'interrogation ce fait par une activation initiale des neurones termes induite par une requête, et qui se propage vers les documents à travers les connexions du réseau

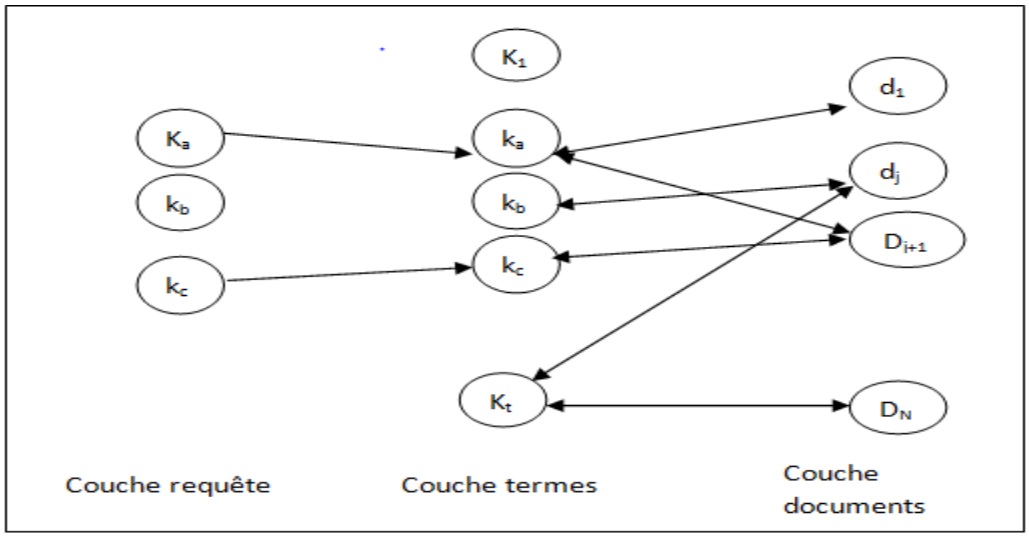


Figure 1.4 Un modèle de réseau de neurones pour la recherche d'information.
[Sauvagnat, 2005]

L'idée de base est que la RI est un processus associatif qui peut être représenté par les mécanismes de propagation d'activation des réseaux de neurones. De plus, les capacités d'apprentissage de ces modèles peuvent permettre d'obtenir des SRI adaptatifs. Les systèmes de RI connexionniste permettent de combler les insuffisances de certains modèles vectoriels. D'autre part les réseaux de neurones permettent de représenter différentes relations peuvent exister entre les termes et les documents :

– Relations entre les termes : synonymie, voisinage,. . .
– Relations entre les documents : similitude, référence,. . .

– Relations entre les termes et les documents : fréquence, poids... .

I.5 Evaluation des systèmes de recherche d'information :

L'évaluation des performances des systèmes de recherche est indispensable. L'évaluation des systèmes peut être entamée selon deux aspects : l'efficience et l'efficacité.

L'efficience regroupe le temps et l'espace dont un système est considéré meilleur lorsque le temps entre la formulation de la requête et la réponse du système est court et l'espace occupé par le système est faible.

L'aspect efficacité concerne la capacité du système à sélectionner le maximum de documents pertinents et un minimum de documents non pertinents.

On s'intéresse dans ce qui suit à présenter l'aspect efficacité qui est souvent mesuré par deux paramètres Rappel et Précision.

I.5.1 Mesures d'évaluation:

a) Le rappel de la précision :

Ce sont les mesures les plus utilisées pour l'évaluation des systèmes de recherche d'information.

La figure ci-dessous représente la répartition des documents d'une collection suite à une requête. Soit |DR| le nombre de documents retrouvés par un système pour une requête donnée, |DP| le nombre de documents pertinents dans la collection pour cette requête et |DPR| le nombre de documents pertinents renvoyés par le système.

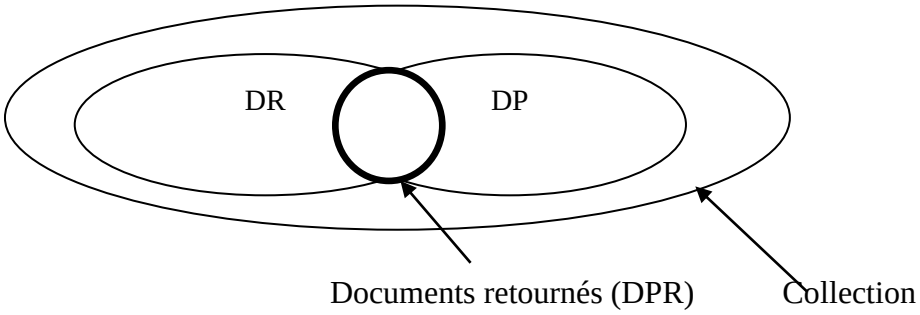


Figure 1.5 la répartition des documents d'une collection suite à une requête

- **La précision :**

La précision mesure la proportion de documents pertinents retrouvés parmi tous les documents retrouvés par le système.

Taux de précision :

$$P = \frac{|DPR|}{|DR|}$$

La précision permet aussi de définir le bruit documentaire qui mesure la proportion de documents non pertinents retournés par le système. Qui se calcule comme suit :

Bruit = 1-précision

- **Le rappel:**

Le rappel mesure la proportion de documents pertinents retrouvés parmi tous l'ensemble des documents pertinents de la collection.

Taux de rappel :

$$R = \frac{|DPR|}{|DP|}$$

Le rappel permet de définir le silence documentaire qui représente la proportion des documents pertinents non retrouvés par le système. Qui se calcule comme suit :

Silence= 1-rappel

Dans un SRI, l'objectif est de minimiser le bruit (les documents non pertinents restitués), et le silence (les documents pertinents non restitués).

- **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, lorsque la précision augmente, le rappel diminue et vice vers sa. Ainsi, pour mesurer les performances d'un système il faut utiliser les deux mesures simultanément. En faisant le calcul de la paire des mesures (taux de rappel, taux de précision) à chaque document restitué.

- **La mesure F-mesure :**

L'obstacle majeur des mesures de rappel et de précision est que ces deux mesures représentent des aspects différents de l'ensemble des documents retrouvés. D'où la nécessité de combiner ces deux mesures en une seule, ce qui nous donne d'autres mesures de performance comme F-mesure. Elle est définie comme la moyenne harmonique pondérée du rappel et de la précision :

$$F_1 = \frac{2 * P * R}{P + R}$$

b) Précision à n :

Cette précision mesure la proportion des documents pertinents retrouvés parmi les n premiers documents retournés par le système. Cette mesure ne prend pas en compte que certaines requêtes comportent peu de documents pertinents.

$$Pn = \frac{DPR}{n}$$

c) La précision exacte ou R-précision :

La précision exacte ou la R-précision est la précision à n obtenue quand n est égal au nombre total de documents pertinents par rapport à la requête. C'est une variante de la Pn où n est le nombre de documents pertinents pour la requête.

d) La précision moyenne:

La précision moyenne est la moyenne des valeurs de précision à chaque document pertinent de la liste ordonnée. Elle tient compte à la fois de la précision et du rappel. Elle présente la moyenne des précisions calculées pour chaque document pertinent à trouver. La précision moyenne, notée MAP (Mean Average Precision), pour une requête donnée, est calculée comme suit :

$$MAP = \frac{1}{n}\sum_{i=1}^{N} P(i) * R(i)$$

Avec :

R(i) = 1 si le i^{ème} document restitué est pertinent
R(i) = 0 si le i^{ème} document restitué est non pertinent
p(i) la précision à *i* documents restitués.
n le nombre de documents pertinents restitués.
N le nombre total de documents.

I.5.2 Collection de test :

Une collection de test est composée d'un ensemble de documents (corpus documentaire), d'un ensemble de requêtes (topics) et des jugements de pertinence (l'ensemble de documents jugé pertinent pour chaque requête) établis manuellement par des assesseurs humains.

Il existe plusieurs collection de test tels que : la collection CACM, CISI,… qui sont utilisées comme moyen d'évaluation des SRI qui se base sur le principe d'envoyer les requêtes test au SRI, puis comparer les réponses retournées par ce dernier avec les réponses idéales. L'écart entre ces deux réponses permet de mesurer la performance d'un système.

I.5.3 Les compagnes d'évaluation :

Sont des systèmes d'évaluations basées sur les collections de test. Il existe plusieurs compagnes d'évaluation tel que : TREC, CLEF (*Cross-language evaluation forum*) pour l'évaluation des systèmes multilingue et la recherche inter-langues spécifique pour les langues européennes, INEX (Initiative for the Evaluation of XML Retrieval) pour l'évaluation des SRI dans des collections semi-structurés XML… Ces derniers sont considérés comme étant les compagnes d'évaluation les plus expérimentées.

Dans ce qui suit, nous présentons la compagne d'évaluation TREC.

I.5.3.1 La compagne TREC :

Les compagnes TREC sont les plus utilisée dans le domaine d'évaluation des systèmes de recherche d'information. TREC (Text Retrieval Conference) est un projet lancé en 1992 par le NIST(le National Institute of Standards and Technology) et le soutien financier de DARPA (l'Advanced Research and Development Activity) Centre du Département de la Défense des États-Unis. TREC organise des conférences annuelles, afin d'encourager les travaux dans le domaine de la recherche d'information en fournissant l'infrastructure nécessaire (collection des tests) et en proposant des méthodologies d'évaluation des SRI.

L'évolution d'un SRI dans le cadre de la compagne d'évaluation TREC consiste à examiner les 1000 premiers documents qu'il retourne pour une requête (thème). Une précision moyenne est calculée qui permet d'obtenir une mesure de la performance globale du système.

Au départ, les recherches dans TREC étaient centrées sur deux tâches principales : la tâche de filtrage et la tâche ad hoc. Par la suite, d'autres tâches ont apparu, tel que : la tâche Terabyte (des évaluations sur des très grands corpus), la tâche QA (Question-Réponse), la tâche multimédia et la tâche web.

II. La recherche d'information dans les documents XML :

L'XML est une recommandation du W3C**,** il est créé pour faciliter l'échange et la standardisation des données.

La recherche d'information dans les documents semi-structurés de type « XML », se base sur la prise en compte de l'information textuelle et de l'information structurelle. D'où la nécessité d'adopter de nouveaux modèles de recherche d'information qui prend en charge les documents semi-structurés de type XML.

Dans cette section, nous présentons les documents semi-structurés de type XML, puis nous passons en revue les modèles de recherche qui prennent en charge ce type de document ainsi que les approches d'évaluation des SRIS.

II.1 Les documents XML :

Le document XML[1] est dit semi-structuré car ce type du document contient d'une part l'information textuelle « contenu » qui est représenté sous forme d'un ensemble de fragment exemple : paragraphe, phrase,… . D'autre part, on trouve l'information structurelle qui définit une organisation hiérarchique des données du document.

La structure du document XML est fixée par le créateur du document, il n'est pas obligé d'utiliser l'ensemble des balises existantes comme dans le HTML.

Un document XML est dit bien formé si seulement s'il est conforme aux règles syntaxiques et lexicales, ces règles sont les suivantes :

- Le document XML ne contient qu'une seule balise racine qui va contenir tous les autres balises ;
- Les balises fermées correctement (pas de chevauchement de balises) ;
- Le nommage conforme des balises (les noms ne peut pas débuter par un nombre ou un caractère de ponctuation, ne pas commencer par les lettres XML, ne doit pas contenir d'espaces, éviter les caractères -, ;, < et >).
- Si un élément possède un attribut, ce dernier doit être unique et il doit contenir obligatoirement une valeur.

Pour avoir un document XML exploitable, il faut que ce document soit bien formé. Ensuite en introduisant la DTD (Document Type Definition) à ce dernier on obtiendra un document valide.

[1] Voir Annexe 1 : les technologies de la famille XML.

```
<?xml version="1.0" ?>
<!--Exemple de fichier XML décrivant un article scientifique-->
<article annee="2003">
<en-tete>
        <titre>Recherche d'information sur le web : la grande révolution</titre>
        <auteur>André Dupont</auteur>
</en-tete>
<corps>
        <section>
                <sous-titre>Histoire de l'hypertexte : des pères fondateurs au World
                Wide Web</sous-titre>
                <par>Afin de maîtriser les enjeux des systèmes hypertexte,
                il convient, même si c'est une tâche ardue, de d'essayer de les définir...
                </par>
        </section>
        <section>
                <sous-titre>Moteurs de recherche</sous-titre>
                <par>On distingue plusieurs types de moteurs de recherche...</par>
                <par>Les annuaires...</par>
                <par>Les moteurs de recherche plein-texte...</par>
                <par>Les meta moteurs...</par>
        </section>
        <section>
                <sous-titre>L'analyse des liens</sous-titre>
                <par>...</par>
        </section>
</corps>
</article>
```

Figure 1.6 Exemple d'un document XML

II.2 Problématique :

L'objectif de la RI est de satisfaire le besoin de l'utilisateur en lui renvoyant les documents pertinents à sa requête. Dans le cas des documents semi-structurés de type XML la structure s'ajoute au contenu, ce qui crée les problèmes suivants :

En 1er lieu au niveau de l'indexation des documents XML, qui se base sur la manière de prise en compte du contenu et de la structure du document. Dont on peut poser les questions suivantes :

- Que doit-on indexer dans la structure des documents ?
- Comment associer cette structure au contenu du document ?
- Par rapport à quelles mesures (niveau élément, document, collection) les termes d'indexation seront pondérés ?

En 2éme lieu on trouve le problème liée à l'interrogation des corpus de documents XML. L'utilisateur doit être capable d'exprimer ses besoins selon plusieurs critères de précision qui font référence au contenu et/ou la structure des documents.

Le dernier problème concerne l'appariement élément/requête, qui vise à attribuer des scores de pertinence aux éléments des documents. Plus la requête et l'élément ont de mots en commun, plus l'élément est considéré comme étant pertinent.

II.3 Caractéristique de la recherche des documents semi-structurés :

L'introduction de l'information structurelle a posée de nouveaux défis pour la RIS. Ainsi, la RI semi-structurés distingue ses caractéristiques de la RI classique.

Cette spécificité porte sur :

- L'unité d'information pertinente ;
- La recherche sur le contenu et la structure ;
- L'interprétation de la requête.

a. L'unité d'information pertinente :

En RI classique, l'unité d'information pertinente est les documents retournés par le SRI suite à une requête.

Dans le cas des documents XML, l'unité d'information correspond à l'élément XML ou un nœud (sous arbre) dans la représentation arborescente. La pertinence d'un document vis-à-vis d'une requête est exprimé en fonction de :

- **L'exhaustivité :** si cette unité contient toutes les informations concernant la requête fournie.
- **La spécificité :** si l'unité ne contient que les informations correspondantes à la requête fournie.

Le principe de la recherche dans les documents XML est le renvoie des unités d'informations les plus exhaustives et spécifiques répondantes à une requête, c'est-à-dire trouver les sous arbres de taille minimale pertinents à la requête.

b. La recherche sur le contenu et la structure

Dans la recherche des documents XML, il existe deux types de requêtes utilisateurs selon leurs connaissances du corpus :

- **Requête orientée contenu :**

Ce type de requête est composée d'un ensemble de mots clés comme en RI classique. Elle est utilisée dans le cas où les utilisateurs n'ont pas d'idée précise de ce qu'ils recherchent où ils n'ont pas des connaissances concernant la structure des documents.

- **Requête orientée contenu et structure:**

Ce type de requête est composé d'un ensemble de contraintes sur le contenu et des contraintes sur la structure. Elle est utilisée dans le cas où les utilisateurs ont des connaissances partielles de la structure de la collection qu'ils interrogent.

c. L'interprétation de la requête :

En RI semi-structuré, l'utilisateur formule son besoin par une requête portant sur le contenu et la structure. La communauté BD a été la première qui a proposé des langages de requêtes

pour l'interrogation des documents XML. Ces langages sont basés sur des syntaxes proches de SQL.

SQL évalue des expressions de type "attribut=valeur". Ainsi, le traitement des requêtes est fait d'une manière booléenne et il est impossible de renvoyer une liste triée en fonction de leur pertinence à l'utilisateur.

d. Approches de la recherche d'information semi-structurée :

Deux communautés se sont intéressées à l'exploitation des collections de documents XML :

- **Approche orientée données :**

La communauté BD propose d'indexer toutes les informations textuelles et structurelles des documents en le stockant au sein de tables des BDD. Le contenu textuel des documents est indexé en tant que chaines de caractères et non sous forme de mots-clés indépendants, ce qui pose le problème pour la recherche sur le contenu textuel des documents. Les langages de requêtes proposées sont généralement basés sur la syntaxe du SQL en spécifiant des contraintes sur la structure des documents.

Au niveau de l'appariement, la pertinence est calculée d'une façon booléenne. Donc, seuls les éléments qui répondent exactement à la requête sont retournés.

- **Approche orienté texte :**

La communauté de la RI utilise des techniques traditionnelles pour extraire des termes d'indexation et tient compte de la structure pour l'indexation et la recherche dans les documents XML.

Cette approche cherche à évaluer le degré de similarité entre la requête et les unités d'information et attribuent à ces dernières un score de pertinence. Donc, la liste des résultats sera triée.

II.4 Indexation des documents XML :

En RI semi-structuré, l'indexation revient à indexer l'information textuelle et l'information structurelle, et de présenter les relations entre les deux types d'information.

L'indexation des documents XML est caractérisée selon deux critères : le schéma de stockage des documents, et les types de transformations possibles entre les documents XML et les structures de stockage.

Il existe deux approches de stockage pour les documents XML.

- **L'approche de stockage XML natif**

Les SGBD natifs XML sont développés spécifiquement pour XML. Ils permettent le stockage des documents complets ou des parties de documents dans des fichiers sans transformation en tables (mapping).

- **Les approches orientées SGBD**

Un middleware assure la conversion entre SGBD et XML. Il permet la déstructuration du document XML afin de stocker les éléments et les attributs en colonnes de tables relationnelles. La structure du document est mémorisée dans des tables séparées pour permettre la recomposition du document lors des recherches.

La seconde dimension représente les différents types de transformation possibles entre les documents XML et les structures de stockage. On distingue :

- **les approches de transformation basées sur un modèle :**

Ces approches créent un schéma générique de base de données qui reflète le modèle de données du format XML. Le schéma de l'index est fixe et connu à l'avance. Des variantes simples de ces approches prennent la représentation en graphe des documents XML et stockent les nœuds et les arcs du graphe dans une base de données. D'autres variantes utilisent la représentation du graphe la plus détaillée du modèle DOM, qui distingue des types de nœuds comme les éléments, les attributs ou les commentaires. Ces solutions, considérées comme extensibles, n'ont pas besoin de la DTD des documents pour les indexer, mais souvent, des fonctionnalités manquent aux index pour répondre à des requêtes portant sur des XPath, sur des hiérarchies ou sur des conditions de contenus relatives à des éléments de structure.

- **les approches de transformations basées sur la structure** :

Ces approches utilisent la structure logique des documents XML ou leur schéma. L'idée est de construire automatiquement un schéma d'index (qui correspondant à un schéma de base de données) en considérant la sémantique de l'application. Contrairement aux approches de mapping basées sur des modèles, dans le cas de mapping basé sur la structure, des applications XML ayant des structures de documents XML différentes donneront lieu à des schémas d'index différents. Ces solutions sont non-extensibles, car les documents possédant des structures différentes ne peuvent pas être ajoutés.

Dans ce qui suit, nous allons présenter les différentes techniques adoptées pour l'indexation du l'information textuelle ainsi l'information structurelle.

II.4.1 Approches d'indexation de l'information textuelle :

L'indexation de l'information textuelle dans les documents XML se situe dans les éléments texte (éléments feuille). L'indexation de cette information est attachée à la structure car l'importance du texte est liée à son emplacement dans le document. Pour cela deux conflits

apparaissent dont le 1er concerne la portée des termes d'indexation et le 2éme concerne la pondération de ses termes.

II.4.1.1 Portée des termes d'indexation :

Les éléments textes (contiennent l'information textuelle) peuvent avoir un nombre quelconque d'éléments parents. Donc plusieurs questions se posent sur la façon d'association de l'information textuelle à l'information structurelle :

Faut-il lier l'information textuelle avec tous les éléments parents où elle apparait ?

Ou la lié avec un seul de ces éléments ?

Deux solutions ont été proposé : l'indexation avec les sous arbres imbriqués et l'indexation avec la décomposition en unité disjointes.

- **Approche d'indexation par sous arbres imbriqués :**

Dans ce type d'approche, les termes d'un nœud feuille sont rattachée à tous les nœuds ancêtres dans le document.

Puisque les documents XML possèdent une structure hiérarchique, les nœuds de l'index sont imbriqués les uns dans les autres. Par conséquence, l'index va contenir beaucoup d'informations redondantes.

La figure ci-dessous représente un exemple d'indexation d'un document par l'utilisation de cette approche. Les termes « André Dupont » sont par exemple reliés aux nœuds /article/en-tête/auteur, /article/en-tête, et /article.

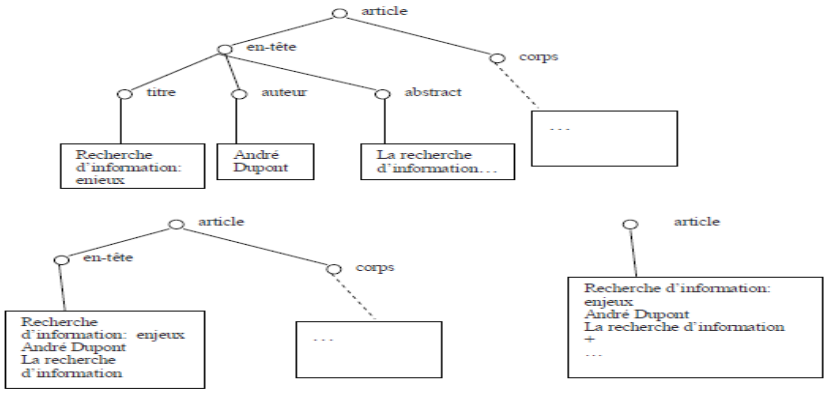


Figure 1.7 indexation par sous arbres imbriqués **[Sauvagnat, 2005]**

- **Approche d'indexation par unités disjointes :**

Dans cette approche, le document XML est décomposé en unités disjointes, de telle manière que le texte de chaque nœud de l'index est l'union d'une ou plusieurs parties disjointes.

Les termes des nœuds feuilles sont reliés uniquement à un et un seul nœud (l'élément qui les contient).prenant un exemple de la figure présenté ci-dessus : les termes « recherche d'information enjeux » seront uniquement reliés au nœud /article/en-tête/titre. Et le nœud /article/en-tête n'a aucun terme associé.

II.4.1.2 La pondération des termes :

Les techniques courantes de pondération des termes sont basées sur les notions de fréquences des termes dans le document tf (Terme Frequency) et de la fréquence de ces termes dans la collection des documents idf (Inverse Document Frequency). De plus dans la RI structuré les unités d'information sont imbriquées les unes dans les autres, d'où l'importance d'un terme dans l'élément dépend fortement de l'importance de ce terme dans les sous éléments qui le composent. Pour cela la mesures ief (Inverse Element Frequency) a été proposé comme une alternative à idf qui est jugé peu efficace dans les documents XML où le nombre de répétitions des termes est généralement assez réduit.

Parmi les formules utilisées pour la pondération avec la prise en compte de nouvelle granularité d'information on trouve :

- L'adaptation des formules de pondération de la RI classique dans la RI semi-structuré :

$$w_i^q = tf_i^q * idf_i$$
$$w_i^{nf} = tf_i^{nf} * idf_i$$

Où :

tf_i^q et tf_i^{nf} sont respectivement la fréquence du terme i dans la requête q et dans le nœud feuille nf.

$$idf_i = log\left(\frac{|D|}{(d_i+1)} + 1\right)$$

Avec : |D| : est le nombre total du document dans la collection.

d_i: est le nombre du document contenant le terme t.

- Certains auteurs ont proposé l'adaptation d'ief pour la pondération des termes dans les nœuds feuilles par des différentes formules dont :

Formule 1 :

$$w_i^q = tf_i^q * ief_i$$
$$w_i^{nf} = tf_i^{nf} * ief_i$$

[Sauvagnat, 2005] et d'autres

Avec :

tf_i^q et tf_i^{nf} sont respectivement la fréquence du terme i dans la requête q et dans le nœud feuille nf.

$$ief_i = log\left(\frac{|F_c|}{|n\ f_i|}\right) + 1$$

Où :

$|nf_i|$: est le nombre de nœuds feuilles contenant le terme i

$|F_c|$: est le nombre total de nœuds feuilles de la collection.

Formule 2 :

$$w_i^q = tf_i^q * ief_i * idf_i \qquad \textbf{[Fellag, 2006]}$$

Avec :

- $ief_i = \log\left(\frac{Ne}{ne_i} + \alpha\right)$ avec $0{,}5 \leq \alpha \leq 1$

Où :

Ne : nombre total de nœuds feuilles dans le document

ne_i : nombre de nœuds feuilles contenant ce terme dans le document

- $idf = \log\left(\frac{N}{n}\right)$

Où :

N : nombre total du document dans la collection

n : nombre de document contenant le terme t

- tf : reflète le nombre d'apparition d'un terme dans le document.

D'autre paramètre peuvent aussi être pris en compte dans la pondération des termes par exemple l'impotence de terme ne soit pas seulement pris en compte dans l'élément qui le contient mais aussi dans le contenu de nœud lui-même , dans le contenu de ses descendant, dans le contenu de ses voisins directs, et dans le contenu des nœuds auquel il est reliés.

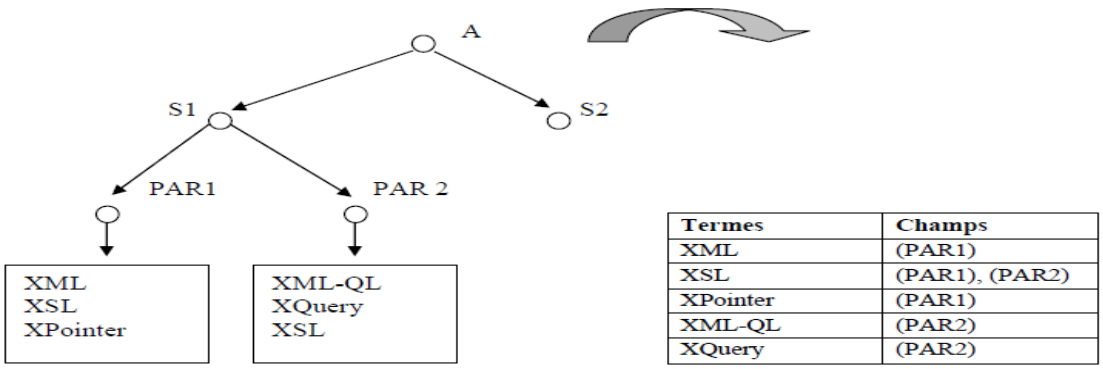
L'agrégation de ces paramètres peut être réalisée à l'aide différent opérateurs notamment avec l'operateur OWA (qui permet la quantification de l'impotence des valeurs partielles en fonction de leur position dans l'ordre croissant indépendamment des attributs).

II.4.2 Approches d'indexation de l'information structurelle :

Il existe différentes approches qui ont été proposées afin d'indexer l'information structurelle selon des caractéristiques variées. On distingue trois types d'approches :

- **Indexation basée sur les champs :**

Ce type d'indexation permet de prendre en compte une structure très simple. La méthode se base sur l'association de chaque terme au nom de la balise dans laquelle il apparait. Cela permet d'effectuer une recherche restreinte à certains champs, car les termes d'index sont construits par la combinaison de nom du champ (nom balise) et les termes de contenu.

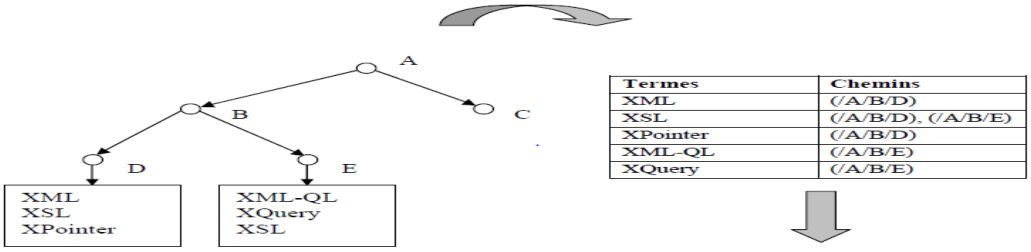


Termes	Champs
XML	(PAR1)
XSL	(PAR1), (PAR2)
XPointer	(PAR1)
XML-QL	(PAR2)
XQuery	(PAR2)

Figure 1.8 Indexation basée sur des champs

- **Indexation basée sur les chemins (Path):**

Cette méthode associe à chaque terme d'indexation le chemin de l'élément où il se trouve. Elle est conçue dans le but de retrouver rapidement des documents ayant des valeurs connues pour certains éléments ou attributs. Elle facilite aussi la navigation dans les documents par la résolution efficace des expressions XPATH, car les index de chemin donnent pour chaque valeur répertoriée d'un chemin de balise la liste des documents contenant un élément atteignable par ce chemin et ayant cette valeur.



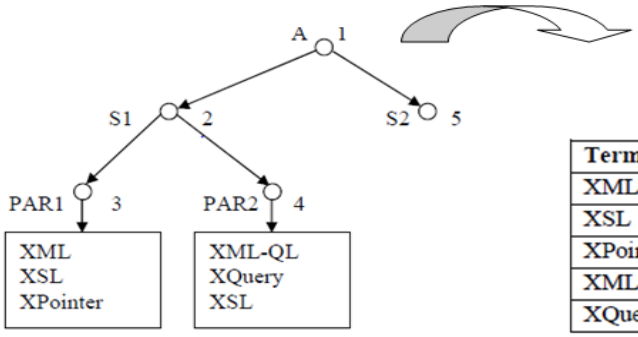
Termes	Chemins
XML	(/A/B/D)
XSL	(/A/B/D), (/A/B/E)
XPointer	(/A/B/D)
XML-QL	(/A/B/E)
XQuery	(/A/B/E)

Chemins	Documents
(/A)	Doc1,…
(/A/B)	Doc1,…
(/A/B/D)	Doc1,…
(/A/B/E)	Doc1,…
…	…

Figure 1.9 Indexation basée sur les paths

- **Indexation basée sur des arbres :**

Dans ce type d'indexation, un identifiant unique UID est attribué à chaque nœud (élément) de l'arbre représentant le document XML. Les termes sont alors associés à cet identifiant, ce qui permet de localiser avec précision l'endroit où ces termes sont apparus et de retrouver les relations hiérarchiques entre les éléments. Cet identifiant associé à un élément peut être calculé sur la position des nœuds dans l'arbre du document.



Termes	Noeuds
XML	3
XSL	3, 4
XPointer	3
XML-QL	4
XQuery	4

Figure 1.10 Indexation basée sur les arbres

II.5 Langages de requête :

Les langages de requête offrent un moyen d'expression du besoin des utilisateurs. Dans le cas des documents XML on trouve une nouvelle forme d'interrogation en plus de requête orienté contenu, s'ajoute les requêtes orientées contenu et structure.

Pour cela de nombreux langages de requêtes ont été proposés dans la littérature. Nous décrivent quelques-uns dans ce qui suit :

II.5.1 Le langage XPATH (Xml PATH Language) :

Xpath est une recommandation du W3C, ce langage est utilisé pour localiser un ou des éléments d'un document XML. Il fournit une structure d'adressage non ambigüe pour faciliter l'extraction des informations requises avec précisions. L'adressage des documents est fait en spécifiant des chemins de localisations qui indiquent dans quelle partie de l'arbre (par rapport au nœud courant) il faut chercher les nœuds à retourner. Un chemin peut être absolu (le chemin à partir de la racine) ou bien relatif(le chemin à partir d'un nœud actuel).

XPath offre la possibilité de navigation selon différents axes qui permettent de descendre dans l'arbre ou de remonter,….

Le test de nœud permet la sélection des nœuds en fonction de leur nom ou de leur type. Ainsi :

« /descendant-or-self::node()/child::S1 » ou //S1

Dans l'exemple on a fait une sélection par apport au type d'élément « node » donc on va sélectionner tous les éléments « S1 » descendants de l'élément actif.

On trouve aussi les prédicats qui permettent le filtrage des éléments sélectionner par les axes. Il est possible de filtrer selon plusieurs critères :

- La position : « titre[1] » sélectionne le 1er titre de l'élément courent.
- Le contenu structurel : « chapitre[titre] » sélectionne les éléments chapitre ayant au moins un enfant de type « titre ».
- Le contenu structurel et textuel : « chapitre [titre="introduction"] » sélectionne les éléments « chapitre » dont un enfant de type « titre » a la valeur « introduction ».

II.5.2 XQuery (XML Query Langage):

XQuery est un langage de requête proposé par W3C. Il permet des expressions de chemin basées sur XPATH, des expressions *FLWR (For, Let, Where, Return*) des expressions conditionnelles (*If, Then, Else*), des expressions quantifiées (*Some, Every*), des opérateurs classiques (*arithmétiques, de comparaison, booléens*) ainsi que l'utilisation de variables et constantes. L'expression *FLWR* compose le squelette de l'expression XQuery, elle est similaire à la construction *SELECT-FROM-Where* de SQL.

Voici en exemple un document XML, une requête XQuery et le résultat qu'elle renvoie. Le document XML suivant, situé à l'URL http://www.example.com/ et nommé exemple.xml :

```
<employes>
  <employe>
    <nom>Durant</nom>
    <prenom>Albert</prenom>
    <date_naissance>23/09/1958</date_naissance>
  </employe>
  <employe>
    <nom>Dupont</nom>
    <prenom>Alphonse</prenom>
    <date_naissance>23/12/1975</date_naissance>
  </employe>
  <employe>
    <nom>Dupont</nom>
    <prenom>Isabelle</prenom>
    <date_naissance>12/03/1967</date_naissance>
  </employe>
...
</employes>
```

La requête XQuery :

```
for $b in document ("http://example.com/exemple.xml")//employe
where $b/nom = "Dupont"
return
  <dupont>{
    $b/prenom,
    $b/date_naissance
  }</dupont>
```

Le résultat :

```
<dupont>
  <prenom>Alphonse</prenom>
  <date_naissance>23/12/1975</date_naissance>
</dupont>
<dupont>
  <prenom>Isabelle</prenom>
  <date_naissance>12/03/1967</date_naissance>
</dupont>
```

II.6 Les modèles de la RI structurée :

Dans les modèles classique de la RI la notion de structure n'est pas prise en compte. Néanmoins, elle est essentielle dans le cadre de la RI structurée, donc cette notion doit être considérée par les modèles de la RI car elle améliore le processus de recherche en apportant des informations supplémentaire.

Plusieurs modèles ont été proposés pour la RI semi-structuré. Dans ce qui suit, on décrira quelque modèle.

II.6.1 Le modèle vectoriel étendu :

Ce modèle est une extension du modèle vectoriel proposé en RI classique où on trouve que la structure est séparé de contenu. En effet ce modèle tente de mesurer la similarité de chaque élément vis-à-vis la requête. Pour le réaliser, chaque élément est représenté sous la forme d'un vecteur de termes pondérés.

Ce modèle a connu plusieurs adaptations qui se base sur l'indexation des sous arbres imbriqués par la propagation des termes des nœuds feuille dans l'arbre du document. Ensuite les éléments seront renvoyer à l'utilisateur par ordre décroissent de pertinence.

Parmi les premières adaptations du ce modèle celle des auteurs **[Fuller et al. ,1993]**, où ils considèrent que chaque élément e défini par un type T reflétant son emplacement dans l'arbre du document XML. La similarité d'un nœud *n* à une requête *q* est exprimée selon la formule suivante :

$$\text{sim(q,n)} = \alpha(T)\cos m(q,n) + \sum_{k=1}^{s} \frac{\cos m(q,n^k)}{\beta^{k-1}}$$

Où :

- α(T) : est un facteur permettant de prendre en compte le type du nœud.
- S : est le nombre de nœuds enfants n_k de n.
- β: est un paramètre permettant d'assurer que le nombre d'enfants n'introduit pas un biais dans la formule.
- La fonction cosm est définie de la façon suivante :

$$\text{Cosm(q,n)} = \sum_{i=1}^{T} \frac{w_i^q * w_i^n}{|n|}$$

Avec

- w_i^q , w_i^n respectivement sont le poids du terme t_i dans la requête q et dans le nœud n.
- |n| le nombre de termes dans le nœud n.

La pertinence d'un nœud peut être calculée puis combinée avec la pertinence des nœuds descendent.

Ce modèle peut être généralisé en permettant le traitement des requête orientées contenu et structure, par l'application de modèle récursivement à chaque sous arbre de la hiérarchie afin d'effectuer ensuite un agrégat des scores.

On présente une autre adaptation où les auteurs **[Yang et al., 2007]** proposent d'étendre le modèle vectoriel par la représentation d'un document XML par une matrice. Ainsi, un élément (sous arbre) XML et la requête sont représentés respectivement par deux matrices, M_E, M_Q. La figure ci-dessous illustre la représentation matricielle d'un document XML. Les valeurs de cette matrice sont les poids des termes dans les différents éléments.

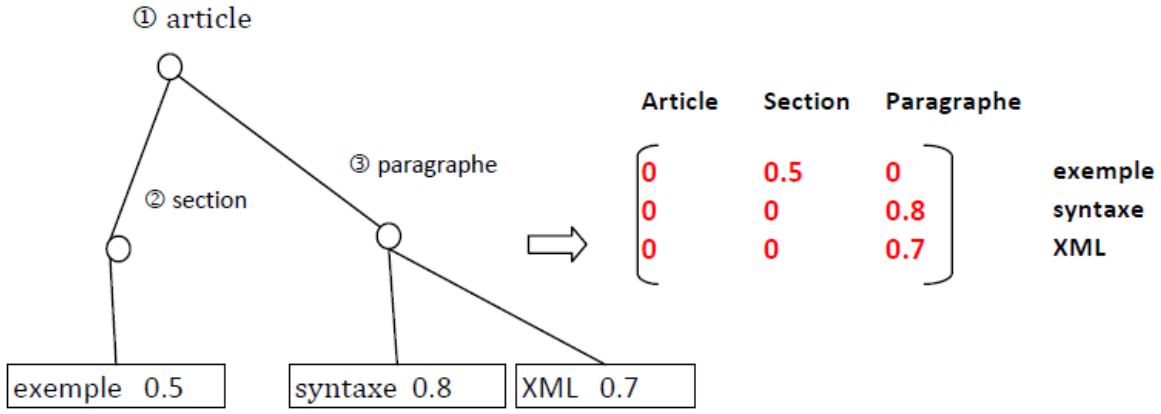


Figure 1.11 Représentation matricielle d'un document **[Yang et al., 2007].**

La fonction de similarité est calculée en effectuant le cosinus entre les deux matrices comme suit :

$$RSV(Q,E) = \frac{M_Q * M_E}{\|M_Q\| * \|M_E\|}$$

II.6.2 Modèle booléen pondéré:

Ce modèle est principalement adapté pour les requêtes orientées contenu, il permet de définir l'unité d'information à renvoyer à l'utilisateur comme étant tous les éléments contenant au moins un élément feuille.

L'objectif de ce modèle est la prise en compte de la structure, grâce au p-distances et la propagation des scores de pertinence, dont les auteurs **[Hatano et al. , 2002]** ont proposé de propager les scores depuis les feuilles de l'arbre documentaire jusqu'à l'élément considéré. Pour cela chaque élément feuille e_j (j=1,…, N) est représenté par un vecteur :

$$F(e_j) = \left(w_{t_1}^{e_j}, w_{t_2}^{e_j}, \dots, w_{t_n}^{e_j}\right)$$

Où :

- *n* représente le nombre de termes indexés dans la collection.
- $w_{t_i}^{e_j}$: représente le poids d'un terme t_i dans un élément feuille qui est donné par :

$$w_{t_i}^{e_j} = \frac{tf(t_i, e_j)}{\sum_{j=1}^{N}\sum_{k=1}^{n} tf(t_k, e_j)} . log \frac{N_d}{df(t_i)}$$

Avec :

- $tf(t_k, e_j)$:est la fréquence du terme t_k dans e_j
- N_d : est le nombre de documents dans la collection.
- $df(t_i)$: est le nombre de documents contenant t_i.

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

- si e est un élément feuille :

$$RSV(e,q) = \frac{W(e) * q}{|w(e)| \times |q|}$$

- si e n'est pas un élément feuille, le score de similarité saura calculer d'une manière récursive en partent de l'élément feuille :

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

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_i (avec $1 \leq i \leq n$) sont égales à 1 si le terme t_i apparait dans la requête, et 0 sinon.

II.6.3 Modèle probabiliste :

L'auteur **[Fuhr, 2001]** présente une méthode basée sur le modèle probabiliste et sur le langage de requêtes XIRQL implémenté au sein du moteur de recherche HyRex. Ils considèrent que les nœuds sont des unités disjointes. Donc on ne peut associer à un nœud que les termes qui n'appartenant pas à ses descendants. Le poids de pertinence des nœuds dans le cas de requêtes orientées contenu est calculé par la propagation des poids des termes les plus spécifiques dans l'arbre du document et est diminué par multiplication par un facteur d'augmentation. Considérant l'exemple suivant représentant la structure d'un document XML contenant des termes pondérés et la requête '*XML'*

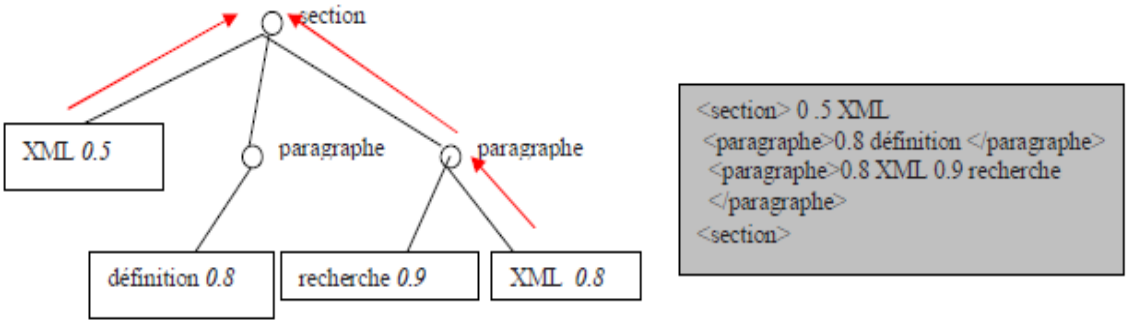


Figure 1.12 Modèle d'augmentation **[Fuhr et al., 2003]**

Le poids de pertinence de l'élément section est calculé comme suit, en utilisant un facteur d'augmentation égal à 0.7 :

P([section,XML])+P([paragraphe[2]]).P([paragraphe[2],XML)-
P([section,XML]).P([paragraphe[2]]).P([paragraphe[2], XML)
= 0.5 + 0.7*0.8 - 0.5*0.7*0.8=0.68.

Le nœud paragraphe mieux classé que le nœud section car il a une pertinence de 0.8 à la requête. Pour les requêtes orientées contenu et structure, des probabilités d'apparition de chaque terme de la condition de contenu dans les éléments répondant aux conditions de structure sont calculées, et des sommes pondérées de ces probabilités sont ensuite effectués.

Conclusion :

L'objectif des SRI est l'identification des parties des documents les plus pertinentes à une requête donnée. Nous avons ainsi présenté les principales approches d'indexation et d'appariement proposées en RI classique et RI structuré.

Les approches présentées dans la recherche d'information dans les documents semi-structurés « XML » et dans la recherche d'information classique sont basées sur des systèmes d'indexation à base de mots clés. Les seules informations utilisées leurs fréquences d'apparition dans les documents ou les éléments du document. Ainsi, ces approches ne prennent pas en compte le sens du mot. Ils ne distinguent pas les mots selon leurs contextes d'apparition. Ces mots clés présentent une forte ambigüité. En effet un mot peut varier de sens selon le contexte où il apparait (polysémie). Aussi, ces approches ne prennent pas en compte la synonymie.

Pour pallier à ce problème, de nouveaux modèles ont été proposés. C'est l'objet de la recherche d'information sémantique que nous présentons dans le chapitre suivant.

Chapitre 2

La recherche d'information sémantique dans les documents XML

I. Introduction :

L'objectif principal des SRI est de satisfaire le besoin des utilisateurs en information. Différent systèmes ont été proposés et développés pour raffiner la recherche en retournant que les documents pertinents à la requête fournie par l'utilisateur. Ainsi, la recherche sémantique est apparue pour améliorer la précision de la recherche et ces résultats par la compréhension des significations contextuelles des termes, car les SRI basé sur les termes peuvent renvoyer un document non pertinent, bien que le document satisfasse la requête.

Dans ce chapitre nous intéressons à la problématique liée à la recherche d'information sémantique dans les documents semi-structurée. Nous présentons l'état de l'art du l'indexation sémantique en RI classique et en RI semi-structurée « XML ». Ainsi les déférentes ressources sémantiques utilisées lors de recherche d'information sémantique. Puis on présente les techniques d'évaluation des SRIS (Système de Recherche d'Information Structurée) à travers la présentation de la compagne d'évaluation INEX.

II. Problématique de la RI sémantique :

L'indexation classique est une extraction d'un ensemble de mots clés qui décrivent une requête ou un document dont la plupart sont imprécise, donc des nouveaux défis ont été rencontré lors de la recherche d'information basé sur les mots-clés qui sont l'ambiguïté des mots et leur disparité **[Boubekeur, 2008]**.

L'ambiguïté des mots, dite ambigüité lexicale. Elle représente des mots lexicalement identiques mais ils portent des sens différents. Cette ambigüité est classée en deux types **[Krovetz, 1997] :**

- **L'ambiguïté syntaxique :** un mot peut avoir plusieurs formes grammaticales, par exemple : dans la phrase « Nous avions des avions », le mot « avions » peut apparaître en tant que nom et verbe.

- **L'ambiguïté sémantique :** elle se ramène à l'homonymie ou la polysémie des mots dont l'homonymie des mots représente les mots qui possède la même forme orale ou écrite mais ils ont des sens déférents par exemple les mots voie et voix se prononce de la même manière mais ils signifient des choses déférentes où le 1er veux dire chemin et le 2^{éme} est le son émis par l'homme. Ainsi la polysémie veut dire qu'un mot peut avoir plusieurs sens comme le mot « figure » dans les exemples suivant à plusieurs sens :
  - Figure de géométrie : Un triangle et un rectangle sont des figures.
  - Un visage : Il avait une figure un peu ronde.
  - Une personnalité : Napoléon est une des grandes figures de l'histoire de France.

L'ambiguïté des mots engendre dans les SRI classique le bruit documentaire ce qui réduit la précision des résultats car les documents non pertinents qui contiennent les mêmes mots de la requête seront retournés à l'utilisateur par le système de recherche.

La disparité des mots : (word mismatch problem) : fait référence à un ensemble des mots qui sont différent lexicalement mais ils portent des sens liés, exemple de synonymie. Ceci

implique qu'il est impossible de trouver des parties des documents représentés par un mot M1 synonyme d'un mot M2, où M2 représente une requête.

La disparité des mots cause le silence documentaire dans les SRI classique de fait que les documents pertinents qui ne partagent pas des mots avec la requête ne seront pas retenu par le système de recherche.

III. Notion de base de la RI sémantique :

- **Concept vs terme** :

Un concept est une idée conçue par l'esprit. Dans un texte les concepts sont indiqués par des termes. En effet, un terme est l'expression linguistique d'un concept repéré dans un texte. Un terme peut être constitué d'un mot (terme simple) ou bien de plusieurs mots (terme complexe).

La figure ci-dessous représente le triangle sémiotique proposé par les auteurs **[Ogden et al, 1923]** qui illustre la relation entre terme et concept :

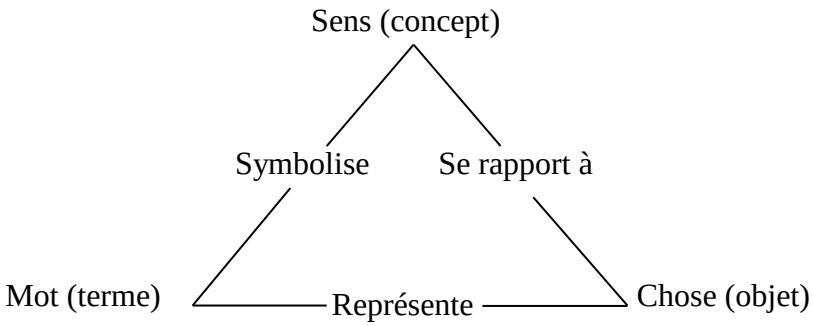


Figure 2.1 Le triangle sémiotique

Selon la norme [ISO 1087,1990] les éléments du triangle sémiotique sont définit comme suit :

- **Objet** : élément de la réalité, les éléments peuvent être matériels (par exemple statue de liberté) au immatériels (exemple la liberté).
- **Concept** : unité de pensée.
- **Terme** : désignation par une expression linguistique d'un concept définit dans une langue de spécialité. Un seul terme peut indiquer plusieurs concepts dans plusieurs domaines.

IV. Ressources sémantique :

IV.1 Types de ressources sémantiques :

Ces ressources jouent un rôle important dans le cadre de la RI sémantique. En effet, elles sont utilisées dans la phase d'indexation dans le but d'extraire les concepts à partir du texte d'un document ou d'une requête. Elles fournissent ainsi des connaissances sur les relations entre les concepts dans le texte. Par conséquent une meilleure représentation du document et

de la requête. Il existe plusieurs types de ressources, on représente les taxonomies, les thésaurus et les ontologies.

- **Taxonomie :**

C'est une forme simple des vocabulaires contrôlés, elle est fondé sur la hiérarchie simple des termes qui conforme à des liens de spécialisation /généralisation.

- **Thésaurus** :

Un thésaurus est un ensemble de termes structurés de façon à faire ressortir des catégories donc est une sorte de dictionnaire hiérarchisé. Ainsi ce dernier représente les termes génériques ou spécifiques à un domaine de connaissance. Ces termes indiquent les concepts d'un domaine particulier. Il existe plusieurs thésaurus et parmi ceux qui existent on trouve le thésaurus linguistique Reget, thesaurus médical Mesh, le méta-thésaurus médical UMLS.

- **Ontologie** :

Une ontologie est un ensemble de termes structurés de façon hiérarchique, conçue pour décrire un domaine et qui peut servir de structure à une base de connaissance. Les ontologies ont pour but d'être cohérentes, partageables (l'ontologie n'est pas la propriété d'un individu, mais elle représente un accord accepté par une commuté d'utilisateurs) et réutilisables.

Une autre définition de l'ontologie a été proposé par Gruber [Gruber, 1993] est que l'ontologie est une spécification formelle et explicite d'une conceptualisation partagé. Elle est considérée formelle car l'ontologie doit être lisible par une machine, la définition explicite des concepts utilisés et des contraintes de leurs utilisations. Le terme conceptualisation désigne la fourniture d'un vocabulaire formalisé de concepts et de leurs relations.

IV.2 Exemple de ressources sémantique :

Wordnet :

Est un dictionnaire électronique de l'anglo-américain développé depuis 1985 par des linguistes du laboratoire des sciences cognitives de l'université de Princeton. Sa structure conforme à celle d'un thésaurus. La majorité des systèmes utilisent Wordnet comme ressource sémantique car Wordnet est parmi les ressources rares disponibles en ligne. Son avantage est que Wordnet est une ressource suffisamment riche.

Un concept de Wordnet dit synset est formé de trois parties:

a. **Le terme représentant du synset** : (un synset est un ensemble de termes synonymes qui définit un concept de wordnet) :c'est le terme pour lequel le concept (synset) est identifié.

b. **Le terme synonyme** : c'est une liste de termes interchangeables séparés par des virgules.

c. **Le glossaire** : contient une définition du concept avec éventuellement un ou plusieurs exemples du monde réel.

Dans Wordnet les concepts sont reliés par des relations sémantiques. La relation de base dans Wordnet c'est la relation de synonymie. Elle relie les termes d'un même nœud (concept).

V. L'indexation :

L'indexation permet la représentation de l'information contenue dans un corpus à l'aide des mots clés appelé descripteur ou index. Dans les SRI classique l'indexation est limitée car elle se base sur le principe de l'existence d'une correspondance entre les mots et les sens alors que un mot peut être représenté par plusieurs sens et un sens peut être représenté par plusieurs mots, donc des documents non pertinents peuvent être engendrés lors de processus de recherche. De ce fait deux alternatives sont nées : l'indexation conceptuelle, l'indexation sémantique.

V.2 L'indexation conceptuelle :

Fondée sur l'indexation des documents avec des concepts d'une base de connaissance. D'où la nécessité d'une liste de concepts cible (exprime le sens des termes possibles) pour permettre la transformation du terme en concept. Les concepts sont extrait d'un vocabulaire contrôlé (les dictionnaires de synonyme, les ontologies, les thesaurus,…).

V.3 L'indexation sémantique :

L'ambiguïté des mots de la langue naturelle a conduit à la naissance de l'indexation sémantique. Cette indexation a pour objectif de représenter les documents et la requête par les sens des mots ce qui permet de lever l'ambiguïté de ces mots, par conséquent l'amélioration des résultats de la recherche.

L'indexation sémantique se base sur les techniques de désambiguïsation des sens des mots (WSD) pour attribuer les sens corrects aux mots d'un document (ou d'une requête).

Généralement, pour désambiguïser les sens d'un mot, les approches de WSD s'appuient sur des ressources ontologiques dont les plus utilisées est WordNet. Les entrées de wordnet sont des synest. Un synest est un concept, il est représenté par un ou plusieurs termes synonymes.

Si un mot d'index est associé à plusieurs synest, il est ambigu, pour le désambiguïser, on associe à chacun de ses synests un score de similarité avec les autres synests des autres mots de son contexte **[boubekeur, 2008]**, **[Bazziz, 2005]**. Le synest qui a le plus grand score est retenu comme sens correct du mot dans le document.

VI. Approches de la RI sémantique dans les documents XML :

Les documents XML sont désignés par l'existence de la structure qui organise leurs contenus textuels. Par conséquent, les systèmes d'indexation et les systèmes de recherche se basent sur trois approches dont le 1er tient compte du contenu textuel, la 2^{éme} de la structure et la 3^{éme} combine à la fois la structure et le contenu textuel.

Dans ce qui suit, on présente quelques approches qui prennent en charge la sémantique lors de processus de recherche dans les documents XML.

VI .1 Approche orientée contenu :

Se base sur l'indexation de contenu textuel. Parmi les approches proposées on présente celle de l'auteur **[Harrathi, 2010]** où il propose un modèle de recherche d'information conceptuelle dans les documents XML. Il représente un document semi structuré sous forme d'un arbre de nœuds (nœuds texte, nœuds éléments) et ils sont reliés par une relation de

structure (parent-fils, …), les nœuds feuilles représentent le contenu textuel du document. Les autres nœuds sont des nœuds internes de type élément.

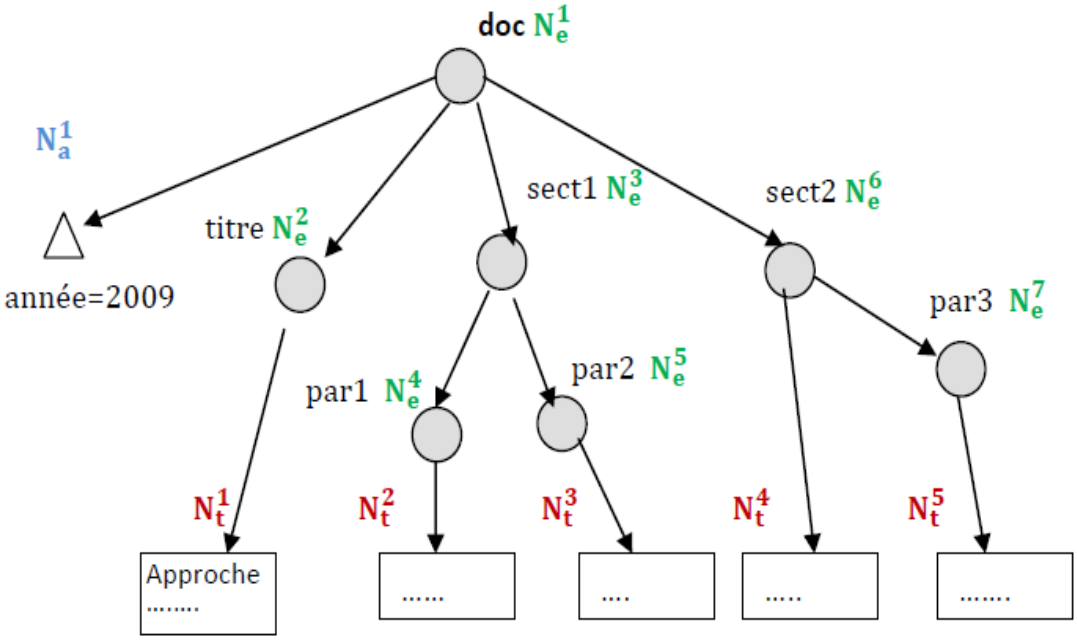


Figure 2.2 Modélisation d'un document XML sous la forme d'un arbre de nœuds **[Harrathi, 2010]**.

L'approche repose sur 3 étapes principales. En 1er lieu l'identification de concept à partir d'une unité textuelle (paragraphe, phrase) par la réalisation d'une analyse syntaxique (association à chaque mot une catégorie grammaticale sa racine « lemme ») ensuite l'auteur propose de repérer les séquences des mots qui peuvent être des termes. Ainsi, les vérifier par apport à une ressource sémantique et garder ceux qui existent dans la ressource. Ensuite il désambiguïse les termes afin de trouver le concept qui correspond au mieux à ces termes dans ce contexte d'énonciation. Ce qui peut être évaluée par l'utilisation des mesures de similarité sémantique. Pour cela il définit une moyenne de similarité entre les concepts d'une combinaison « CB » comme suit :

$$\text{Moyenne\_Sim(CB)} = \frac{2.\sum_{i=1}^{i=n} \sum_{j=i+1}^{n} sim_{\Omega}(c_i, c_j)}{n.(n-1)}$$

L'auteur utilise la fonction représenté ci-dessous pour la sélection de la combinaison ayant la plus grande moyenne de similarité qui jugé très proche de contexte d'énonciation

$$\text{Max} = \text{ArgMax}(\text{Moyenne\_Sim}(CB_i)) \text{ avec } 1 \leq i \leq \prod_1^{k=n} |concept_{\Omega}(t_k)|$$

Où

- CB_i est l'i^{ème} combinaison de concepts du contexte.
- t_k est le k^{ème} terme dans le contexte.
- n est le nombre des termes dans le contexte.

Après l'identification des concepts, il les pondère comme suit :

- Pondération locale d'un concept au sein d'un élément

CF_i^j = *nombre d'occurence du concept* c_j *dans un noeud texte* N_T^i

- Pondération globale d'un concept au sein de la collection

$$IECF_j = \log\left(\frac{|N_T|}{\left|N_T^{c_j}\right|}\right)$$

Où :

- $|N_T|$ est le nombre total de nœuds textes de la collection.
- $\left|N_T^{c_j}\right|$ est le nombre total de nœuds textes contenant le concept c_j.

Le poids d'un concept c_j dans un nœud texte N_T^i (dénoté par W_{ij}) est exprimé par la formule suivante:

$$W_{ij} = CF_i^j * IECF_j$$

En 2éme lieu on trouve la construction d'index, qui se base sur la propagation des concepts des nœuds de type texte dans l'arbre de document.

En dernier lieu il dénote l'appariement nœuds/requête dont le score de pertinence est calculé comme suit :

$$score(N_E, q) = cosinus(M_E, M_q) = \frac{\langle M_E, M_q \rangle_F}{\|M_E\|_F \left\|M_q\right\|_F}$$

Où

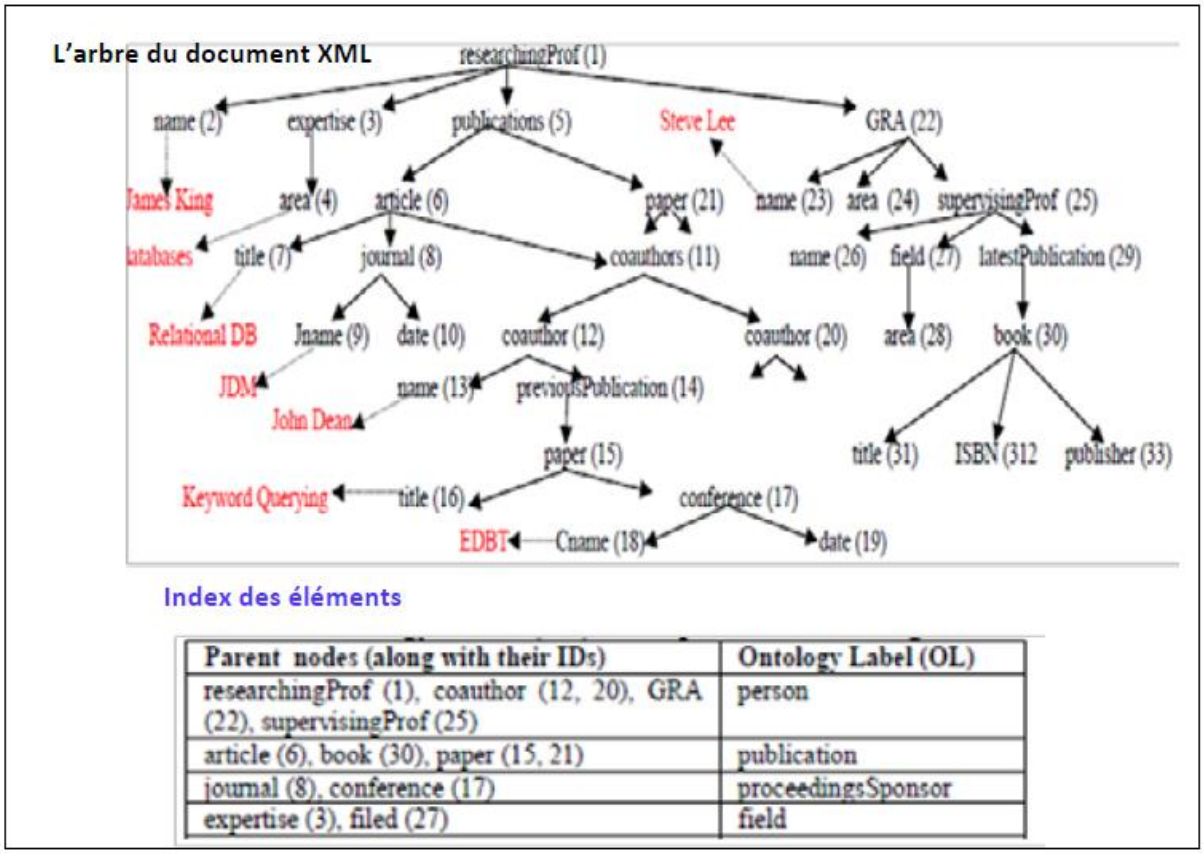
- M_E et M_q sont les matrices représentant les graphes sémantiques associés au vecteur du nœud NE et au vecteur du requête q.

VI.2 Approche orienté structure :

Cette approche s'intéresse uniquement à l'indexation de la structure. Cela est réalisé par la représentation des noms des éléments (nom balise) par un concept, en utilisant une ontologie de noms.

L'intérêt de cette approche réside dans le fait qu'elle supporte les requêtes vagues. Car si on trouve dans une requête une contrainte structurelle portant le nom d'éléments, le système de recherche peut retourner les éléments indexés par le même concept qui est spécifié dans la contrainte structurelle. Par exemple dans la requête on cherche un élément dont le nom est « university » le système va retourner tous les éléments dont le nom est « university » et « school ».

Parmi les travaux réalisés on cite celui de CXLEngine **[Taha et al., 2008]** qui utilise une ontologie de noms (ontology label) pour faire la relation entre les noms de balises et les concepts dans la hiérarchie de l'ontologie. Où l'ontologie utilisée est créé par les auteurs.



Parent nodes (along with their IDs)	Ontology Label (OL)
researchingProf (1), coauthor (12, 20), GRA (22), supervisingProf (25)	person
article (6), book (30), paper (15, 21)	publication
journal (8), conference (17)	proceedingsSponsor
expertise (3), filed (27)	field

Figure 2.3 L'indexation de la structure dans le système CXLEngine **[Taha et al., 2008]**

Utilités de cet approche sera plus bénéfique, quand la structure est sémantiquement riche, où les noms de balises ne servent pas seulement à décrire l'organisation du contenu textuel, mais aussi va contenir des connaissances qu'on peut l'exploiter.

VI.2 Approche orienté structure et contenu :

L'approche repose sur l'indexation de la structure et le contenu textuel par des concepts par l'utilisation d'une ontologie.

Certains auteurs **[Schenkel R et al., 2003]** définissent un document XML comme un ensemble de paires (concept élément, valeur) où la valeur indique l'index du contenu textuel qui est représenté par un ensemble de concepts pondérés.

Parmi les approches on présente le système de recherche XXL **[Schenkel et al., 2005]** qui permet l'interrogation des documents XML. Il se base sur trois principales structures d'index :

a. **Index des noms des éléments et des attributs** : les noms sont indexés par des concepts. L'index permet l'accès aux nœuds parents, descendent et ancêtres d'un nœud donnée. Cet index permet de calculer la distance entre ces deux nœuds.
b. **Index du contenu d'élément** : permet de repérer les éléments dans lesquelles un terme apparaît. La pertinence des termes est calculée par TF-IEF (Term Frequency, Inverse Element Frequency).

c. **Index ontologie** : repère les termes reliés sémantiquement à un domaine donnée à travers le calcule de similarité qui peut limiter un certain type de liens. De ce fait une mesure de similarité peut être calculée entre deux concepts.

Les auteurs proposent la construction d'une ontologie à partir de WordNet pour calculer la similarité entre les concepts. Cette ontologie est représentée selon un graphe dont les nœuds sont les concepts et les arcs sont les relations entre ces concepts.

Les arcs sont pondérés par des fréquences basées sur la relation entre les concepts à partir des pages web retournés par le moteur de recherche sur le web.

La mesure de similarité[2] entre deux concepts $C_1(n_0)$ et $C_2(n_1)$ suivent un chemin $p=(n_0,\ldots,n_k)$ est calculé comme suit :

$$sim_p(C_1, C_2) = \prod_{i=1}^{|p|-1} poids(n_i - n_{i+1})$$

Où :

- $|p|$ est la longueur du chemin p,
- $poids(n_i - n_{i+1})$dénote le poids de l'arc $e = (n_i - n_{i+1})$.

La similarité entre deux concepts est calculée par la formule suivante :

$$sim(c_1, c_2) = max\{sim_p(c_1, c_2)\}$$

La désambiguïsation dans XXL se base sur le contexte de terme (l'ensemble des termes co-occurents dans un document ou dans la requête), un terme peut être liée à plusieurs concepts donc il faut définir un sens fixe pour ce terme en choisissant le contexte dans les concepts sont très similaire.

Les langages de requête utilisés dans XXL sont XML-QL et XQUERY dont la syntaxe proche de celle de langage SQL. On présente ci-dessous un exemple d'une requête XXL :

```
Select * From Index
Where #.~publication  AS A
And  A.~title ~ "XML"
```

Figure 2.4 Exemple d'une requête XXL.

Où : « ~ »est un opérateur de similarité permet l'expression des conditions de similarité sémantique sur l'élément ainsi que sur le contenu textuel. Ainsi il retourne la liste des termes les proches dans l'ontologie, c'est-à-dire ceux qui ont un degré de similarité le plus élevé avec le terme recherché.

[2] Voir Annexe 2 : les mesures de similarité

VII. Evaluation des SRI semi-structurés :

L'évaluation des SRI est une phase très importante pour comparer leurs performances. Comme TREC (*Text Retrieval Conference*) pour la RI classique, INEX est considérée comme étant la seule campagne d'évaluation des SRI dans des documents XML. Elle vise particulièrement à mesurer la capacité de ce système à retrouver des éléments à la fois exhaustifs et spécifiques au sujet de la requête. L'évaluation de l'efficacité des SRI dans des documents XML nécessite une collection de test (documents XML, requêtes et jugements de pertinence), cette collection est fournie aux différents participants pour arriver à effectuer un classement de leurs systèmes au niveau de la campagne.

Dans la suite, nous décrivons brièvement le corpus INEX, les requêtes, les tâches, les jugements de pertinence et les mesures d'évaluation.

VII.1 La compagne d'évaluation INEX :

INEX (INitiative for the Evaluation of Xml retrieval) est créée en 2002 dans le but d'évaluer les systèmes de recherche d'information structurée. Le but était le développement d'une méthodologie et des outils (collection de test, requêtes et mesures d'évaluation), pour tester des systèmes et approches de recherche XML, en utilisant un langage de requêtes qui est NEXI (Narrowed Extend XPath I) et un ensemble de tâches telles que : ad-hoc, la tâche multimédia, la tâche « Relevance Feedback » et la tâche hétérogène, et par la suite plusieurs compagnies INEX dérivées de celle-ci à plusieurs évolutions et améliorations sont utilisées pour évaluer les SRI participant.

VII.1.1 Collection de test :

Elle contient les éléments suivants :

a. Collection de documents *:*

Les documents utilisés sont des articles scientifiques provenant de l'IEEE Computer Society, balisés en XML.

Les articles sont composés Les articles sont généralement structurés de la façon suivante :

- Chaque article est composé d'un en-tête (<fm>), un corps (<body>) et d'annexe.

Un élément, parmi ceux cités est conçu de la manière suivante :

- Le corps est composé de sections (<sec>)
- Une section est composée de paragraphes (<p>)
- Les annexes sont composées de références bibliographiques (<bibl>) et de curriculum vitae (<vt>).

A partir de 2006, la collection de base utilisée pour les tests étant la collection *Wikipedia,* qui est utilisée dans la plupart des tâches. Cette collection de 6 Go, est composée de 659.388 documents d'une profondeur (nombre de niveaux) moyenne de 6.72. Le nombre moyen de nœuds XML par document est 161,35. Cette collection est également utilisée dans la tâche multimédia, elle contient environ 246.730 images.

b. Les requêtes (topics):

Les requêtes crées par les participants doivent refléter les besoins des utilisateurs. On distingue deux principaux types de requêtes :

- **Les requêtes CO (Content Only) :**

Similaire aux requêtes classiques en RI. Les mots clés de la requête peuvent être regroupés sous forme d'expressions et Précédés par les opérateurs '+' (Spécifiant que le terme est obligatoire) ou '**-**' (Spécifiant que le terme ne doit pas apparaître dans les éléments renvoyés à l'utilisateur).

- **Les requêtes CAS (Content and Structure) :**

Ces requêtes contiennent de plus des contraintes sur la structure des documents qui sont présenté dans la figure ci-dessous :

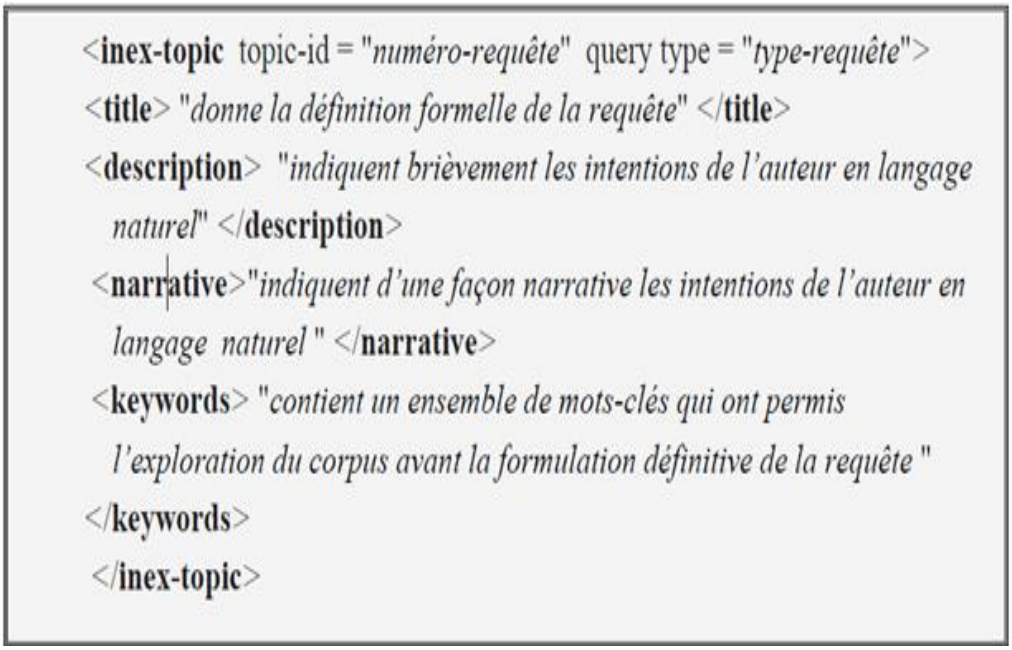


Figure 2.5 Anatomie d'une requête INEX

c. Jugement de pertinence :

Le jugement de pertinence permet d'évaluer les résultats retournés par les SRI participent à l'évaluation. Qui passe par une validation des éléments/documents qui sont jugés à la main en utilisent le système de jugement en ligne. Puis en 2002 une échelle à deux dimensions a été proposée et basée sur les degrés de pertinence et couverture, qui ont été remplacée par les notions d'exhaustivité et de spécificité, depuis 2003.

- **Une dimension d'exhaustivité :**

L'exhaustivité est mesurée selon une échelle à quatre niveaux, un élément est **[Lalmas et al., 2005] :**

- Pas exhaustif : l'élément ne traite pas du tout du sujet de la requête.
- Marginalement exhaustif : il traite peu d'aspects du sujet de la requête.
- Assez exhaustif : il traite de nombreux aspects du sujet de la requête.
- Très exhaustif : il traite la plus part ou tous les aspects du sujet de la requête.

- **Une dimension de spécificité :**

La spécificité décrit à quel point l'élément est focalisé sur la requête, i.e. la couverture du document/élément au sujet et elle est aussi mesurée sous quatre niveaux, l'élément peut être à:

- Pas de couverture : le thème traité n'a rien à avoir avec celui de la requête, i.e. n'est pas du tout spécifique.
- Couverture trop large : où marginalement spécifique où le thème de la requête est traité exactement dans un sous élément.
- Petite couverture : si l'élément renvoyé contient juste une partie de l'information pertinente.
- Couverture exacte : où l'élément est très spécifique et le seul sujet qui traite est celui de la requête.

Ces deux dimensions de la pertinence reflètent la pertinence d'un élément par apport à ses descendants. En effet, un élément peut être plus exhaustif que chacun de ses descendants pris séparément. De même, des éléments peuvent être plus spécifiques que leurs parents.

VII.2 Les tâches:

La tâche principale d'INEX est celle de recherche *ad-hoc*, elle est considérée comme une simulation de l'utilisation d'une bibliothèque, où un ensemble statique de documents est interrogé avec des besoins utilisateurs, c'est à dire des requêtes. La tâche ad-hoc est à son tour composée de sous-tâches divisées comme suit:

- **Tâche CO** (Content Only Task) :

Qui permet de répondre avec des éléments ou des documents XML à des requêtes utilisateur de type CO.

- **Tâche SCAS** (Strict Content And Structure Task) :

Consiste à répondre avec des éléments ou des documents XML aux requêtes CAS de manière stricte(en respectant toutes les conditions sur la structure et le contenu énoncées dans les requêtes, le champ Title des requêtes est basé sur une syntaxe XPath).

- **La tâche VCAS : (Vague Content And Structure task) :**

Consiste à répondre aux requêtes CAS de manière vague i.e. : des éléments ou des documents XML qui satisfont globalement les requêtes.

Depuis 2004, quatre nouvelles tâches ont été proposées aux participants :

a. **La tâche de "*relevance feedback*"** : qui se base sur l'utilisation du contenu et de la structure comme informations de base pour la formulation d'une nouvelle requête.

b. **La tâche de "*langage naturel*"** : formulation des requêtes en langage naturel.

c. **La tâche "*interactive*"** : fondé sur l'étude de comportement des utilisateurs face à des corpus XML pour cerner leurs besoins.

d. **La tâche "*hétérogène* "**, qui propose aux participants de nouvelles collections afin de développer des approches indépendantes des DTDs.

VIII. Mesures d'évaluation :

Jusqu'à 2004, les seules mesures appliquées dans l'évaluation des SRI étaient le rappel et la précision par la suite dans les compagnies INEX 2005 et 2006 d'autres mesures ont été définies pour mettre une meilleure évaluation des SRI en RI structurée **[Xavier, 2006]**

- **Le gain cumule (xCG) :**

Cette mesure représente la pertinence des éléments de la liste des résultats qui sont ordonné d'une façon décroissante par apports à leurs scores de pertinence :

$$xCG(i) = \sum_{j=1}^{i} xG(j)$$

Où :

- i : le rang de l'élément dans la liste ;
- xCG(i) : somme des scores de pertinence des documents j (j = 1,i) ;
- xG(j) : le score du document de rang j.

Après le calcule de gain cumulé des éléments, pour chaque requête on calcule un vecteur de gain idéal xCGI à partir de la base de rappel, et le xCG peut être alors comparé au xCI avec le nxCG :

$$nxCG(i) = \frac{xCG(i)}{xCI(i)}$$

Tel que : pour l'élément de rang i, le score de pertinence cumulé acquis sur le score de pertinence idéal voulu nous donne une valeur de norme comprise entre 0 et 1 tel que :

$$\text{Si nxCG(i)} \rightarrow 0 \text{ élément non pertinent}$$
$$\text{Si nxCG(i)} \rightarrow 1 \text{ élément pertinent}$$

Il reflète le gain relatif que l'utilisateur accumule jusqu'à ce rang si le système avait produit une liste triée optimale.

- **Mesure de precal :**

Elle a été utilisée lors de la compagnie d'évaluation 2002 pour définir la probabilité qu'un élément retrouvé et retourné à l'utilisateur soit pertinent, est calculée par :

$$P(pert/retr)(x)\frac{x.n}{x.n + esl_{x,n}}$$

Tel que : pert : document x pertinent

retr : document retrouvé

$esl_{x.n}$: nombre attendu d'éléments non pertinents retrouvés jusqu'à ce qu'un point de rappel x soit atteint

n : le nombre de documents pertinents dans la collection par rapport à une certaine requête.

- **L'effort précision (ep) :**

Elle représente l'effort (en nombre de liens à visiter) qu'un utilisateur doit fournir pour parvenir à un gain donné *r*, e_{system} (respectivement e_{ideal}) est le rang auquel le gain *r* est atteint

par le système (respectivement par la liste optimale). Cette mesure dépend du gain, car elle est calculée par :

$$ep(r) = \frac{e_{ideal}}{e_{system}}$$

Tel que :

- r : c'est le gain
- e : le rang correspondant au gain
- $e_{idéal}$: le rang auquel le gain est idéal

e_{system} : le rang auquel le gain est celui retourné par le système évalué

Conclusion

Nous avons présenté dans ce chapitre les concepts fondamentaux de la recherche sémantique en RI classique et RI semi-structurée.

La recherche sémantique se base sur l'utilisation des ressources sémantiques et des mesures de similarités sémantiques entre les concepts, car ils permettent l'identification des concepts à partir du texte d'un document ou d'une requête. Ces ressources offrent aussi des connaissances sur la corrélation entre les concepts dans le texte à l'aide des mesures de similarités sémantique.

Le problème engendré par l'utilisation d'une ressource sémantique dans la RI est la désambiguïsation sémantique qui influence la qualité des résultats obtenus.

Dans le chapitre suivant, nous présentons un modèle de recherche pour la prise en compte de la sémantique dans la recherche d'information semi-structuré.

Chapitre 3

L’approche de la RI sémantique dans les documents XML

Introduction

Dans le chapitre précédent, nous avons vu la plupart des travaux réalisés en RI sémantique dans les documents semi-structurés de type « XML ». Les modèles existants permettent de palier les limites de l'indexation classique, en utilisant des concepts en tant qu'entités d'indexation au lieu de simples mots-clés.

Nous nous proposons dans le cadre de notre travail de définir un modèle de RI sémantique dans les documents XML. Notre proposition consiste à étendre le modèle LSI (Latent Semantic Indexing) proposé en RI classique pour qu'il prenne en charge les documents semi-structurés de type XML. Dans la section suivante, nous décrivons en détail notre proposition.

Description de l'approche proposée

1.1 Modélisation d'un document semi-Structuré

On représente un document XML par un arbre de nœuds typés (nœuds de type texte, nœuds de type élément), reliés par des relations de structure (voir Figure 3.1).

Les nœuds feuilles de l'arbre sont des nœuds textes. Ils contiennent l'information contenue dans la structure correspondante dans le document XML.

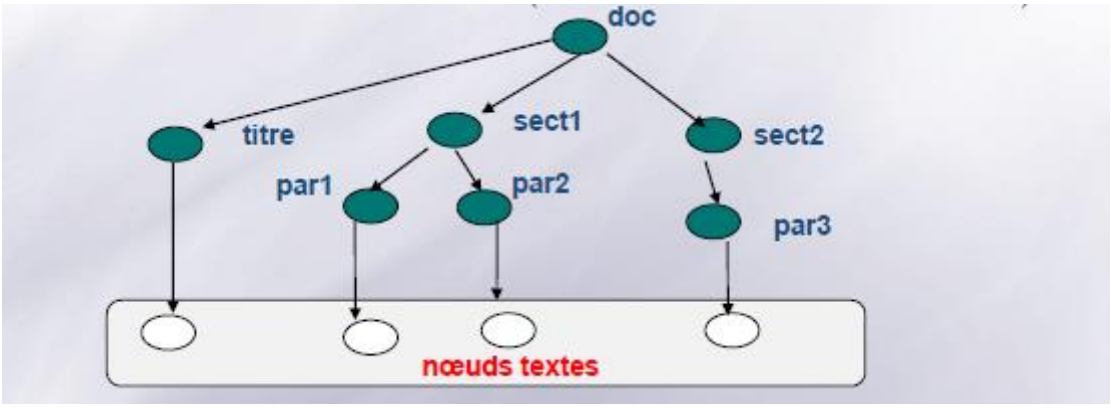


Figure 3.1 Modélisation d'un document XML sous la forme d'un arbre de nœuds.

Un nœud d'un document de la collection est identifié par l'entité NœudDocument définie par le couple: (id-nœud, id-document), où id-nœud est l'identifiant du nœud et id-document l'identifiant du document.

Si D_j est un document XML de la collection, représenté par un ensemble de k_j nœuds n_{1j}, n_{2j}, …, n_{kj} et N le nombre total de documents de la collection, alors le nombre de NœudDocuments de la collection est :

$$ND = \sum_{j=1}^{N} (k_j)$$

1.2 Indexation sémantique latente (LSI)

Le modèle LSI (*latent Semantic Indexing*), ou modèle d'indexation par sémantique latente, est une variante du modèle vectoriel de base qui a été introduite pour prendre en compte la sémantique des termes d'indexation. LSI se base classiquement sur la matrice termes-documents qu'elle réduit en appliquant la technique de décomposition en valeurs singulières (SVD) de la matrice termes-documents.

Pour appliquer LSI dans le cas des documents XML, nous proposons de construire, au lieu de la matrice termes-documents, la matrice termes-NoeudDocuments, notée A. Dans cette matrice, les lignes sont des termes, et les colonnes les entités NoeudDocuments définies pour l'ensemble des documents de la collection. La matrice A ainsi obtenue, de dimension n x m (où m est le nombre de NœudDocuments de la collection et n le nombre de termes de la collection), contient des valeurs binaires représentant l'absence ou la présence (0,1) d'un terme dans un nœud.

En appliquant la technique SVD à la matrice A, LSI permettra alors de clusteriser au sein d'un même « concept » des termes d'index sémantiquement proches tout en les associant aux nœuds des documents de la collection. La structure du document est ainsi sauvegardée tout en tenant compte de son contenu informationnel sémantique.

1.3 Construction de l'espace sémantique par SVD[3]

La décomposition en valeurs singulières (ou SVD) repose sur la réduction de la dimensionnalité de la matrice termes-NoeudDocuments. La nouvelle matrice obtenue représente l'espace sémantique des concepts associés aux nœuds des documents de la collection.

Pratiquement, la matrice A est de dimension m x n est décomposée comme suit :

$$\boldsymbol{A = U\,DV^{t}}$$

Où :

- A : matrice termes-NoeudsDocuments
- U, V : sont des matrices de dimension m × r , n × r respectivement (ou $r \leq \min(m, n)$) contenant un ensemble de vecteurs orthonormés ($U.U^t = 1$, $V.V^t = 1$).
- D : est une matrice diagonale de dimension r × r contenant les valeurs singulières de A.

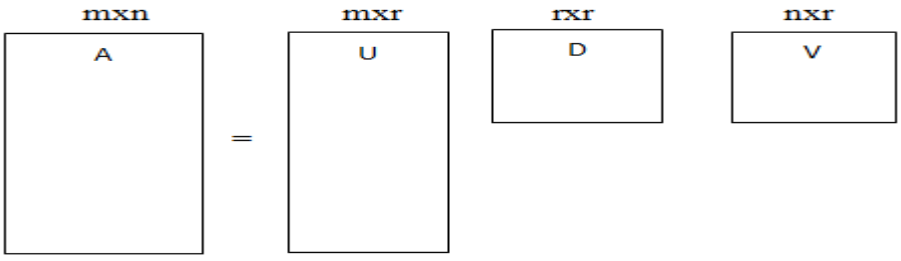


Figure 3.2 La décomposition de la matrice A par SVD.

[3] Voir Annexe 4 : exemple de calcul SVD

L'idée de réduire la dimensionnalité de la matrice A consiste à tenir compte seulement des k termes ayant les plus grandes valeurs singulières de la matrice (le coefficient k est déterminé expérimentalement). Les autres entrées sont mises à zéro ainsi que les colonnes correspondantes dans U et V. Le produit des matrices ci-dessus donne la nouvelle matrice termes-NoeudDocuments de dimension m x k dans l'espace sémantique réduit de dimension k (i.e. contenant k concepts).

$$A = U\,DV^t \cong \tilde{U}\,\tilde{D}\tilde{V}^t = \tilde{A}$$

Où :

- $\tilde{A}$: la nouvelle matrice créée en tenant compte du coefficient k.
- $\tilde{U}$, $\tilde{V}^t$: sont des matrices de dimension m×k et n×k respectivement, élaborées à partir de U et de V.
- $\tilde{D}$: est la nouvelle matrice créée en se basant sur le coefficient k

Ces matrices sont utilisées pour définir la collection de documents par une nouvelle matrice $\tilde{A}$ de dimension n′ × k ($n' \leq n$, $k \leq m$) car avec cette méthode certains termes et certains documents peuvent disparaître de la matrice.

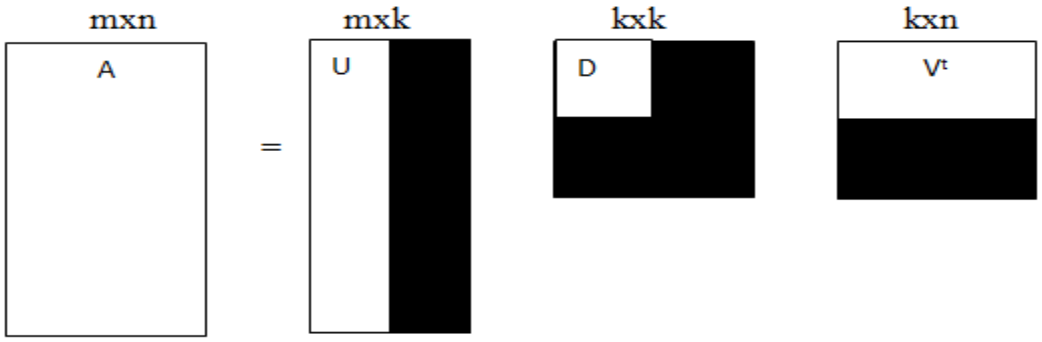


Figure 3.4 La nouvelle décomposition de la matrice A par SVD par application de k

Par ailleurs, la requête Q est aussi transformée dans ce nouvel espace sémantique en un pseudo-document D_q.

$$D_q = q^t U_k D_k^{-1}$$

Où : q est le vecteur contenant les mots-clés de la requête q. q^t est sont transposé.

Une mesure de similarité standard entre les vecteurs requête et NoeudDocument dans l'espace réduit permet alors d'ordonnancer la pertinence des nœuds de documents par rapport à la requête.

1.4 Appariement nœuds / requête

L'appariement est la comparaison entre la représentation de la requête avec les représentations des nœuds (les nœuds de type texte et les nœuds de type élément dans l'arbre XML) des documents dans l'espace sémantique réduit, afin de calculer pour chaque couple

requête-nœud, un score de pertinence système qui reflète le degré de similarité entre la requête et le nœud considéré.

La fonction de calcul de la similarité est donnée comme suit:

$$RSV(Q,E) = \frac{M_Q * M_E}{\|M_Q\| * \|M_E\|}$$

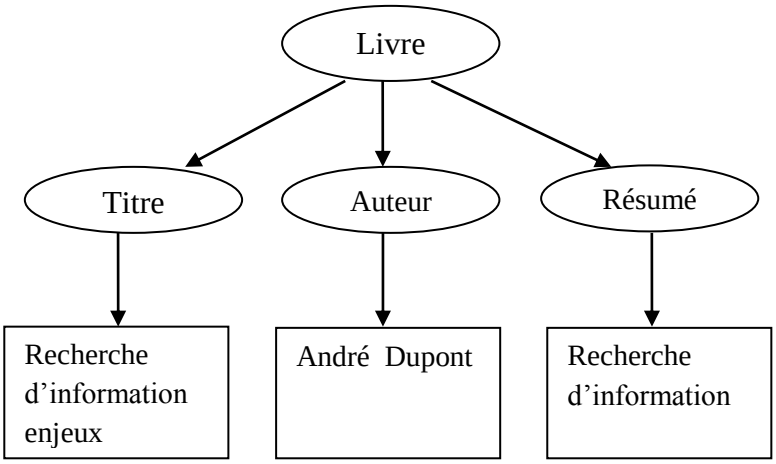
Où :

- $M_{\tilde{Q}}$ est le vecteur requête dans l'espace sémantique réduit
- M_E est le vecteur nœud du document dans l'espace sémantique réduit

1.5 Exemple d'une recherche d'information en modèle LSI dans les documents XML:

Dans cette section, nous expliquons notre approche proposée à travers son application à un exemple de document XML.

Soit un document XML donné par l'arbre suivant:



La matrice termes-NœudDocument A (où les colonnes représentent les termes et les lignes représentent les nœuds du document), et le vecteur requête q se présentent comme suit :

$$A = \begin{array}{c} \\ recherche \\ information \\ enjeux \\ André \\ Dupont \end{array} \begin{array}{c} \begin{array}{cccc} article & titre & auteur & résumé \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \end{array}$$

$$q = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Pour appliquer la décomposition en valeurs singulières de la matrice A, donne:

$$A = U\,DV^t \cong \tilde{U}\,\tilde{D}\tilde{V}^t = \tilde{A}$$

Tel que :

$$U_k = \begin{bmatrix} -0.657 & 0 & -0.261 & 0.707 \\ -0.657 & 0 & -0.261 & -0.707 \\ -0.369 & 0 & 0.929 & 0 \\ 0 & -0.707 & 0 & 0 \\ 0 & -0.707 & 0 & 0 \end{bmatrix}$$

$$D = \begin{bmatrix} 2.136 & 0 & 0 & 0 \\ 0 & 1.414 & 0 & 0 \\ 0 & 0 & 0.662 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$V^T = \begin{bmatrix} 0 & -0.788 & 0 & -0.615 \\ 0 & 0 & -1 & 0 \\ 0 & 0.615 & 0 & -0.788 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

Les coordonnées des vecteurs termes sont les lignes de la matrice U_k et les coordonnées vecteurs documents sont les colonnes de la matriceV_k^t.

On calcule les coordonnées du vecteur requête q dans l'espace réduit $D_q = q^t U_k D_k^{-1}$

$$D_q = [-0.615 \quad 0 \quad -0.788 \quad 0]$$

Pour calculer la pertinence des nœuds pour la requête, on utilise le score prédéfini suivant :

$$RSV(Q,E) = \frac{M_Q * M_E}{\|M_Q\| * \|M_E\|}$$

Ce qui donne :

$$RSV(q, n_{titre}) = \frac{(-0.615)(-0.788) + (-0.788)(-0.615)}{\sqrt{(-0.615)^2 + (-0.788)^2}\,\sqrt{(-0.615)^2 + (-0.788)^2}} = 0.485$$

$$RSV(q, n_{article}) = 0$$

$$RSV(q, n_{auteur}) = 0$$

$$RSV(q, n_{résumé}) = 0.435$$

Conclusion

Dans ce chapitre, nous avons proposé et décrit une nouvelle extension du modèle LSI pour la recherche des documents semi-structurés de type XML.

Nous avons en particulier adapté la représentation de l'index par une nouvelle matrice d'incidence termes-NoeudDocuments, puis adapté l'approche LSI à ce cas. L'idée étant de retrouver l'espace sémantique des termes associés à chaque nœud de chaque document de la collection. Nous avons en outre expliqué comment la requête peut être transformée dans ce nouvel espace sémantique, puis défini une mesure de similarité entre la requête et les nœuds des documents.

Dans le chapitre suivant nous nous intéressons à l'implémentation et à la validation de notre approche.

Chapitre 4
Expérimentations et résultats

Introduction

Dans le chapitre précédant nous avons présenté un nouveau modèle de RI sémantique dans les documents XML. Qui est une extension de modèle LSI proposé en RI classique pour qu'il prenne en compte les documents XML

Dans ce chapitre, nous présentons l'implémentation du modèle proposé sur le système de recherche XFIRM. Par la suite, nous montrons l'environnement de développement qui a servi d'appui pour notre travaille et à la fin, nous décrivons les résultats d'expérimentation effectués pour l'évaluation de l'approche implémentée.

I. Description des phases d'implémentation :

Afin de réussir l'implémentation du modèle LSI, il est indispensable de définir ces trois phases d'implémentation suivantes, dans le système de recherche XFIRM :

I.1 Phase d'indexation

Le système XFIRM utilise le parseur SAX qui permet d'analyser un document XML et de traiter tous ces nœuds afin de récupérer des informations de numérotation des nœuds, ainsi que le contenu du document. Il permet ainsi d'extraire les termes d'index du texte puis de remplir la base de données avec toutes les informations en mémoire (obtenues pendant le parcours de la collection de documents).

I.2 Phase de construction de la matrice termes-noeudDocuments

Afin d'appliquer le modèle LSI (*latent Semantic Indexing*), dans le cas des documents XML, nous proposons de construire, la matrice termes-NoeudDocuments. Où les lignes sont des termes, et les colonnes les entités NoeudDocuments définies pour l'ensemble des documents de la collection. Soit la matrice de dimension n x m (où m est le nombre de NœudDocuments de la collection et n le nombre de termes de la collection), contient des valeurs binaires représentant l'absence ou la présence (0,1) d'un terme dans un nœud.

I.3 Phase de recherche

Elle consiste à retrouver des éléments (sous arbres) en réponse à une requête. Ce traitement consiste à évaluer la similarité des nœuds feuilles retournés à la requête représentée (il s'agit alors du calcul du score des nœuds feuilles). En propageant vers le haut le score des nœuds feuilles dans l'arbre du document, et ce en privilégiant les nœuds les plus porteurs d'informations, et en propageant vers le bas le score du document dans sa globalité, afin de tenir compte du contexte du sous arbre dans l'évaluation de sa pertinence.

II. Tables et classes d'XFIRM utilisées

Afin de récupérer les données et de construire notre matrice termes-noeudDocuments, on a exploité un ensemble de tables et classes du système XFIRM, ainsi que des modifications ont été portées sur un ensemble de classes des différents packages.

II.1 Les tables utilisées

Dans le but d'implémenter le modèle LSI, on s'est basé sur les données stockées dans les tables de la BDD et on a eu accès aux tables suivantes :

- Terme : contient des informations sur les termes de la collection : fréquence dans la collection, nombre d'éléments le contenant …
- Path : contient des informations sur les nœuds de la collection : document conteneur, l'attribut, taille …

En plus des tables utilisées on a créé la table suivante :

Table	Rôle	Description des champs
TermNode(term_id NUMBER(38), node_id NUMBER(38))	contient tous les nœuds feuilles existant dans la base de données ainsi leurs termes associés.	**term_id:** identifiant d'un terme. **node_id:** identifiant du nœud contenant le terme.

II.2 Les classes utilisées

II.2.1 Avant modification

Pour implémenter notre modèle, on a utilisé les classes d'XFIRM suivantes :

- *Xfirm.util.Similarity* : classe permettant de gérer le calcul des scores des nœuds feuilles.
- *Xfirm.inex.TestInexRun* : est une classe à méthode *main* permettant de lancer un Run Inex (pour les tests dans le cadre d'INEX).
- *Xfirm.test.TestQuery* : classe à méthode *main* permettant de lancer une requête (pour les tests simples).
- *Xfirm.store.BaseReader* : cette classe permet de lire la base de données Oracle dans laquelle sont stockés les index des documents (configuration de la connexion et récupération des informations).
- *Xfirm.store.BaseWriter :* cette classe possède tous les méthodes nécessaires à l'écriture dans la base donnée Oracle.
- *Xfirm.util.Resultat_Final* : est une classe utilisée à la fin des tests pour la récupération et l'évaluation des résultats des requêtes. Elle calcule le gain cumulé et le gain idéal, (calculer le gain cumulé à 5, à 10, à 25, à 50) pour des résultats ordonnés par ordre décroissant, après la normalisation des scores en les divisant sur le plus grand score (le premier).
- *Xfirm.util.PathText* : Objet contenant pour un nœud donné ce qui doit être affiché à l'utilisateur : nom du document, Xpath et score.
- Xfirm.test.TestIndexer : Main permettant de lancer l'indexation.

II.2.3 Après modification

Voici la description des modifications portées sur chaque classe utilisée :

- *Xfirm.util.Similarity* :l'ajout d'une méthode qui sert a calculé la mesure de similarité entre les nœuds des documents et la requête.
- *Xfirm.inex.TestInexRun :* l'implémentation de modèle LSI par la construction de la matrice termes-NoeudDocuments ainsi le calcul du SVD de cette matrice.
- *Xfirm.test.TestQuery :* l'ajout de possibilité de lancer une requête par le modèle LSI.
- *Xfirm.store.BaseReader* : par l'ajout des nouvelles méthodes suivantes :
  - **docNodeId** : Pour renvoyer l'identifiant d'un document pour un nœud passé en paramètre.
  - **Id_Term** : Pour renvoyer les identifiants des termes d'un nœud passé en paramètre.
  - **Tous_Termes** : permet la sélection des termes existant dans la base de données.
  - **Node_Id** : permet la sélection de tous les identifiant des nœuds feuille existant dans la base de données.
  - **Term_Id** : la sélection de tous les identifiants des termes existant dans la base de données.
- *Xfirm.store.BaseWriter :* l'ajout d'une méthode pour la création et le remplissage de la table *TermNode.*
- *Xfirm.util.PathText* : la modification de type de score de similarité de float a double.
- *Xfirm.test.TestIndexer* : l'appelle à la méthode qui permet l'ajout des index à la table *TermNode.*

Ainsi on ajouter la classe TermPath dans le package index :

Xfirm.index.TermPath : un objet TermPath correspond à une ligne de la table *TermNode.* On a effectué aussi des modifications sur Le répertoire import qui contient jar nécessaires pour la compilation et l'exécution des classes par l'ajout d'une autre bibliothèque « lingpipe.jar » nécessaire au calcul de SVD.

- **La bibliothèque « lingpipe.jar » :**

Lingpipe.jar est une API java open source, contient plusieurs packages[4] et classes. Nous avons utilisé cette API pour la réalisation de SVD, donc nous avons exploité le package définit dans cette bibliothèque *com.aliasi.matrix* qui contient plusieurs classe où on trouve la classe *svdMatrix*.

La classe ***SvdMatrix*** fournit un moyen de stocker une matrice qui a été pris en compte par l'intermédiaire d'une décomposition en valeurs singulières (SVD). Cette classe fournit

[4] http://alias-i.com/lingpipe/docs/api.

également une méthode statique pour calculer régularisés décompositions en valeurs singulières de matrices partielles. L'appelle à cette classe s'effectue ainsi :

```
public static SvdMatrix.svd (double[][] values,
                    int NumFactors,
                    double featureInit,
                    double initialLearningRate,
                    double annealingRate,
                    double regularization,
                  double minImprovement,
                    int minEpochs,
                    int maxEpochs,
                   null)
```

Parameters:

- *values*: La matrice à décomposer.
- *NumFactors*- Ordre maximum de la décomposition « k ».
- *featureInit* - La valeur initiale de vecteurs singuliers
  initialLearningRate : Multiplicateur par accroissement d'erreur déterminant qui détermine à quelle vitesse l'etude se fait.
- annealingRate - Taux auquel le recuit se produit, des valeurs plus élevées fournissent un recuit plus progressif.
- *regularization* : Une régularisation constante pour l'apprentissage.
- *minImprovement* : Amélioration relative minimum de moyenne carrée requise pour finir une époque.
- *minEpochs* : Duré minimum pour l'exécution.
- *maxEpochs* : duré maximum pour l'exécution. Où *null* si aucune sortie est souhaitée.

Retours:
Décomposition en valeurs singulières de la matrice partielle spécifiée à l'ordre spécifié.

III Algorithme du l'implémentation de modèle LSI

L'implémentation de modèle LSI dans le système XFIRM se fait suivant les étapes suivantes:

- Récupération de la BDD les données suivantes:

- Les identifiant des nœuds feuilles dans la collection à partir de la table *TermNode.*
- Les identifiants des termes à partir de la table *TermNode.*
- Les termes dans tous les nœuds feuilles de la collection à partir de la table *termes*.
- L'identifiant d'un document pour un nœud feuille de la collection à partir de la table *path.*
- Les identifiants des termes d'un nœud feuille de la collection à partir de la table *TermNode.*

- La mise à jour de la classpath par l'ajout de chemin vers la bibliothèque lingpipe.jar.
- la construction de la matrice termes-NoeudDocuments.
- Le calcul du SVD (Singular Value Decomposition) qui permettra le regroupement au sein d'un même « concept » des termes d'index sémantiquement proches.
- L'ajout de la mesure de similarité.
- Lancement de l'exécution de programme.

IV Environnement et outils d'implémentation :

Pour implémenter notre modèle et pouvoir le tester, on a utilisé un ensemble d'outils de développement dans un environnement adéquat.

Cette application a été réalisée dans une plateforme Unix (UBUNTU 10.10) montée dans une VMware Workstation, en utilisant les outils suivants : XFIRM comme système de recherche d'informations, Eclipse comme environnement de développement, Java comme langage de programmation, Oracle 10g XE comme SGBD pour la gestion de la BDD du système et les pilotes ODBC & JDBC pour la connexion à la BDD du système XFIRM pour récupérer les données.

IV.1 VMware Workstation :

VMware Workstation est un environnement de test et de développement qui permet aux administrateurs système de créer et d'exécuter des machines virtuelles (VM) directement sur un bureau.

La VMware Workstation 9, est optimisée pour fonctionner avec les systèmes d'exploitation 64 bits et Windows 8. Workstation utilise une interface Web pour connecter les utilisateurs de machines virtuelles locales et serveur hébergé depuis un PC, un smartphone ou une tablette. La figure suivante montre l'interface de la VMware Workstation 9 :

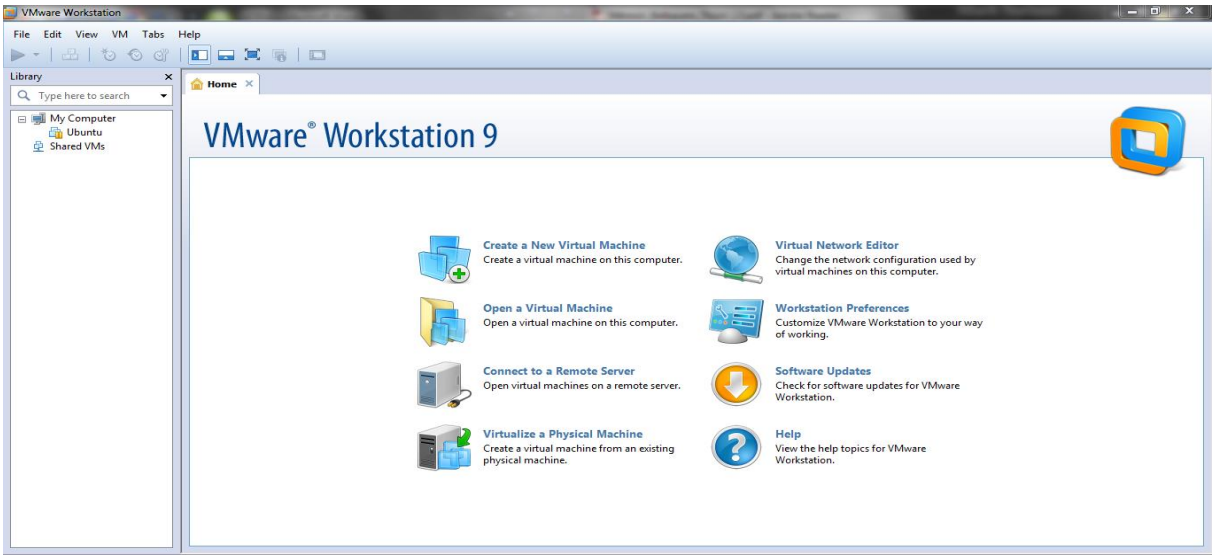


Figure 4.1 Interface de la VMware Worksation 9

IV.2 Le système de recherche XFIRM :

C'est un système de recherche d'informations développé en 2005 par Karen Sauvagnat, ce système se base sur le modèle vectoriel **[Sauvagnat, 2005]** et est orienté vers la RI structurée dans des corpus documentaires XML. Il a apporté de la flexibilité dans la recherche car il utilise une représentation des documents par un modèle générique de données (description des formats des documents dans une base de données) et un langage de requêtes en permettant l'expression du besoin en information de l'utilisateur avec des simples expressions du langage naturel puis une reformulation des requêtes.

IV.3 L'environnement de développement Eclipse :

Est un environnement de développement intégré (IDE) développé par I.B.M. pour des applications telles que java. Eclipse est une plateforme ouverte pour l'intégration de l'outil, pas un IDE. La question a été confondue car une force industrielle complète, y compris la fonction IDE Java est fournie avec la plate-forme Eclipse, sous la forme de plug-ins qui étendent les installations du cadre de base de l'Eclipse. En outre, les plug-ins Eclipse peuvent s'étendre d'autres plug-ins. Quand une application basée sur Eclipse démarre, il découvre et active tous les plug-ins qui ont été configurés pour le poste de travail. La plate-forme Eclipse est littéralement la somme de ses parties, car il est capable d'effectuer n'importe quelle fonction qui a été ajoutée à elle par les plug-ins qu'il contient pour le moment. Eclipse est dit universel et polyvalent, car il permet de créer des projets de développement, qui mettent en œuvre n'importe quel type de langage de programmation (Java, C++, PHP, JavaScript, …). La figure suivante présente l'interface de l'IDE Eclipse sous linux :

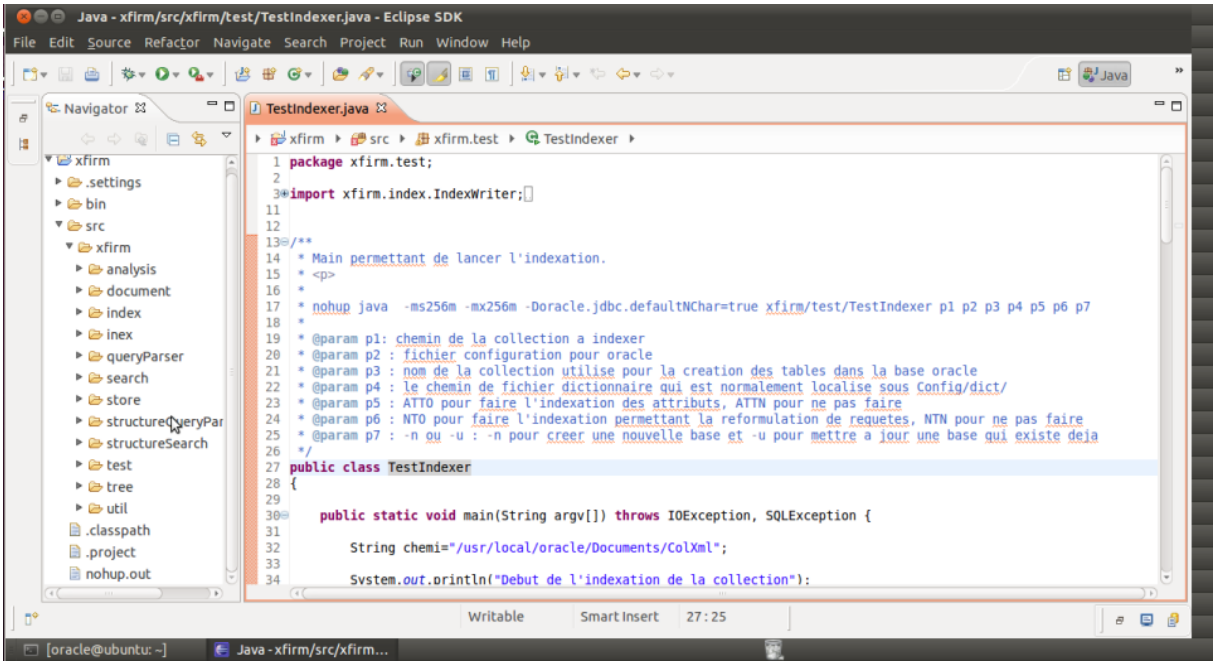


Figure 4.2 L'interface de l'IDE Eclipse

IV.4 Le langage Java :

Le choix de java comme langage de programmation s'est imposé vue que le code source d'XFIRM est entièrement écrit en java.

On peut faire remonter la naissance de Java à 1991. À cette époque, des ingénieurs de chez**Sun Microsystems** ont cherché à concevoir un langage applicable à de petits appareils électriques (on parle de *code embarqué*). Pour ce faire, ils se sont fondés sur une syntaxe très proche de celle de C++. L'idée consistait à traduire d'abord un programme source, non pas directement en langage machine, mais dans un pseudo langage universel, disposant des fonctionnalités communes à toutes les machines. Ce code intermédiaire, dont on dit qu'il est formé de *byte codes*, se trouve ainsi compact et portable sur n'importe quelle machine, il suffit simplement que cette dernière dispose d'un programme approprié (on parle alors de*machine virtuelle*) permettant de l'interpréter dans le langage de la machine concernée. Il estdit aussi un langage multiplateforme selon la célèbre maxime «Write once, run everywhere ».Java est un langage de programmation à usage général mais avec des fonctionnalités qui le rendent plus simple :

- Création des applications web,

- Les applets,

- Programmation procédurale, événementielle et implémentation flexible de l'orienté objet,

- Bibliothèques riches de méthodes.

- …

Dans notre cas, on s'est basé sur la programmation procédurale par l'exploitation des différentes bibliothèques permettant l'implémentation des différentes phases du système.

IV.5 Le SGBD Oracle10g EX :

Un SGBD (Système de Gestion des Bases de Données) est un logiciel permettant la création, manipulation et modification des BDD, ainsi que le contrôle et la confidentialité des données, dans notre cas, le SGBD utilisé est Oracle 10g :

IV.5.1 Définition :

Oracle est un SGBD écrit en langage ℂ et édité par la société du même nom (*Oracle Corporation* http://www.oracle.com), qui est un leader mondial des bases de données et a été créée en 1977 par Lawrence Ellison, Bob Miner, et Ed Oates. Elle s'appelle alors *Relational Software Incorporated (RSI2)* et commercialise un Système de Gestion de Bases de données relationnelles (SGBDR ou RDBMS pour *Relational Database Management System*) nommé *Oracle*.

En 1984 la première version d'Oracle (Oracle 4) est commercialisée sur les machines IBM. Et en 1985 Oracle 5 permet une utilisation client-serveur grâce au middleware *SQL*Net*. Et en 1988 Oracle 6 est disponible sur un grand nombre de plates-formes et apporte de nombreuses nouvelles fonctionnalités ainsi qu'une amélioration notable des performances. Et ce n'est qu'en 1992, qu'Oracle 7 sort sur les plates-formes UNIX (elle ne sortira sur les plates-formes Windows qu'à partir de 1995). Cette version permet une meilleure gestion de la mémoire, du

CPU et des entrées-sorties. La base de données est accompagnée d'outils d'administration (SQL*DBA) permettant une exploitation plus aisée de la base.

IV.5.2 Les fonctionnalités d'Oracle :

Oracle est un SGBD permettant d'assurer :

- La définition et la manipulation des données
- La cohérence des données
- La confidentialité des données
- L'intégrité des données
- La sauvegarde et la restauration des données
- La gestion des accès concurrents (transactions)

IV.5.3 L'interface SQL*PLUS :

Oracle propose également de nombreux outils de développement permettant d'automatiser la création d'applications s'interfaçant avec la base de données. Et parmi ces outils on a utilisé SQL*PLUS qui est une interface interactive permettant d'envoyer des requêtes SQL et PL/SQL à la base de données, la créer et de manipuler interactivement des objets de la base via une interface en ligne de commandes à travers le SGBD Oracle. SQL*PLUS est habituellement utilisé pour formuler une requête SQL et obtenir, sur un écran et de façon immédiate, le résultat attendu.

La figure suivante montre cette interface sur la plateforme linux :

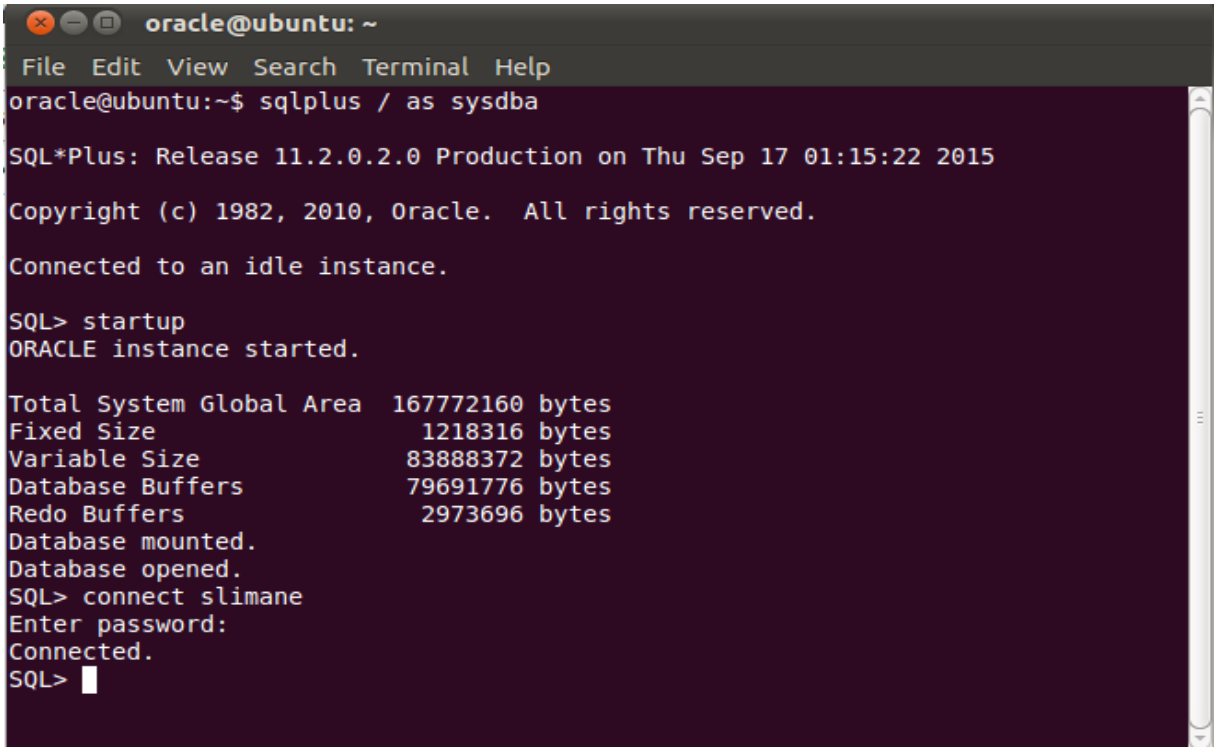


Figure 4.3 Illustration de l'interface SQL*PLUS sous linux

IV.6 Les outils de connexion au SGBD et à la BDD :

IV.6.1 Le pilote de connexion au SGBD Oracle :

La classe permettant la connexion au SGBD est : *java.sql.DriverManager* qui se charge de la gestion, du contrôle et de la connexion au SGBD en fournissant les méthodes principales suivantes :

- static void registerDriver(Driver driver): enregistre le driver pour un type de SGBD particulier ;
- static Connection getConnection (String url, String user, String password): Crée une connexion permettant d'utiliser une base de données.

Avec:

- driver : est un pilote dépendant du SGBD utilisé, dans notre cas il s'agit du pilote d'oracle sous linux : *jdbc:oracle:thin:@oracleserv.irit.fr:1521:test*
- url : identification de la base considérée sur le SGBD à format dépendant du SGGB utilisé
- user : nom de l'utilisateur qui se connecte à la base
- password : mot de passe de l'utilisateur.

IV.6.2 L'ODBC/JDBC :

- **Le JDBC :**(Java Data Base Connectivity)

Est un outil de connexion à la BDD conçu par Sun. JDBC est un Framework pour le langage Java permettant l'accès aux bases de données relationnelles dans un programme Java indépendamment du type de la base utilisée (mySQL, Oracle, Postgres...) et seule la phase de connexion au SGBDR change. Il permet de faire tout type de requêtes : sélection des données dans des tables, création et insertion d'éléments dans les tables et gestion des transactions.

Les packages java le configurant : *java.sql* et *javax.sql*

- **L'ODBC :**

Abrégé en ODBC Open Database Connectivity est un outil de connexion à une BDD, il construit un pont « *Bridge* » entre le pilote JDBC et la BDD configurée. Tel que le JDBC communique avec l'interface ODBC et non directement avec la BDD. L'intérêt réside dans le caractère standard de l'ODBC qui est utilisé dans les SGBD comme *Microsoft SQLServer* ou *Microsoft Access*. Ces différents drivers permettent d'accéder indifféremment à la plupart des SGBD.

- **Principales classes pour accéder à une BDD**

- *La classe Driver Manager* charge et configure le driver de la base de données qui permet de gérer l'accès à un type particulier de SGBD.
- *La classe Connection* réalise la connexion et l'authentification à la base de données.

- *La classe Statement (et PreparedStatement*) contient la requête SQL et la transmet à la base de données.
- *La classe ResultSet* permet de parcourir les informations retournées par la base de données dans le cas d'une sélection de données.

Dans ce qui suit, on présentera l'environnement d'évaluation de notre modèle : exemple de requêtes utilisées, la collection de base et les résultats des tests ainsi que leur évaluation par les mesures déjà présentées.

V Environnement d'évaluation :

V.1 Les requêtes :

Pour effectuer les tests des formules de pondération proposées et pouvoir les comparer, on a utilisé un échantillon de 30 requêtes CO (Content Only) extraites de 124 requêtes de la compagne INEX 2006 qu'on a pris aléatoirement quatre sous-ensembles de ces requêtes.

- emperor Napoleon I Polish
- genetic algorithm
- Italian Flemish painting Renaissance -French –German
- wifi security encryption

V.2 La collection de test :

La collection INEX 2006 contient 659388 documents, qui sont répartis en 22 ensembles. Le nombre d'éléments dans toute la collection est 50.000.000. Pour nos tests, nous avons utilisé un échantillon de 500 documents différents, extraits de la collection

V.3 Les mesures d'évaluation :

Nous avons utilisé les métriques d'évaluation de INEX afin d'évaluer la performance de notre système. Nous avons utilisé la classe *Resultat_final* permettant de calculer la valeur nxCG des scores :

$$nxCG[i] = \frac{xCG[i]}{xCI[i]}$$

Le principe consiste à calculer la valeur nxCG pour 4 rangs, ainsi *i* aura les valeurs (5, 10, 25, 50) qui représentent respectivement les nombre des premiers résultats considérés et la totalité des résultats. Cette valeur est calculée pour le système XFIRM et notre approche.

- **xCG** représente la somme des scores des *i* premiers résultats obtenu par le système considéré à savoir XFIRM avec ces propres formules et XFIRM avec notre approche.
- **xCI** représente la somme des *i* premiers scores obtenus pour les résultats de INEX 2006 (les scores idéales).

VI Expérimentations et résultats:

Le calcul de score :

Après l'implémentation de modèle LSI, on a lancé des tests simples avant de lancer les tests dans le cadre de la compagne INEX et cela en faisant varier le paramètre « k » lors de la construction de la matrice termes-NoeudDocuments.

Et voici les résultats d'un test simple sur une requête dont son contenu est : « tourism paris visit museum cathedral »

Figure 4.4 Résultat d'une recherche simple.

Dans le cadre d'INEX, les tests donnent tous les résultats d'une requête dans un fichier résultat.

VI Evaluation des résultats :

Après la récupération des résultats des tests pour l'ensemble des requêtes obtenues par le modèle implémenté et le système XFIRM, dans des fichiers résultats, les scores obtenus sont évalués en calculant le nxCG à 4 niveaux, tel que la montre le tableau suivant :

LSI				
identifiant requête	**Gain à 5**	**Gain à 10**	**Gain à 25**	**Gain à 50**
289	0,988	0,54	0,624	0,633
290	0,974	0,535	0,619	0,628
291	0,992	0,546	0,636	0,642
292	0,989	0,542	0,627	0,638
293	0,975	0,533	0,619	0,642
294	0,98	0,539	0,622	0,629
295	0,942	0,5131	0,597	0,605
296	0	0	0	0
297	0,984	0,538	0,623	0,632
298	0,996	0,541	0,624	0,631
299	0,974	0,535	0,620	0,620
300	0,973	0,548	0,644	0,653
301	0,979	0,534	0,616	0,626
302	0,994	0,539	0,620	0,628
303	0,951	0,516	0,593	0,604
304	0,999	0,548	0,641	0,654
305	0,994	0,551	0,645	0,654
306	0,998	0,549	0,638	0,650
307	0,973	0,534	0,619	0,630
308	0,944	0,51	0,590	0,597
309	0	0	0	0
310	0,989	0,537	0,618	0,630
311	0,977	0,536	0,620	0,627
312	0,985	0,533	0,616	0,624

313	0,991	0,54	0,627	0,640
314	0,982	0,53	0,610	0,620
315	0,993	0,541	0,629	0,638
316	0,984	0,532	0,629	0,445
317	0,99	0,547	0,629	0,640
318	0,972	0,534	0,629	0,630
La moyenne	**0,915**	**0,501**	**0,581**	**0,583**
Xfirm				
La formule tf*idf				
289	0,362	0,122	0,057	0,050
290	0	0	0	0
291	0,826	0,308	0,172	0,058
292	0,372	0,128	0,064	0,135
293	0,765	0,298	0,151	0,135
294	0,410	0,155	0,074	0,066
295	0,436	0,159	0,081	0,073
296	0,493	0,173	0,090	0,089
297	0,425	0,156	0,078	0,068
298	0,541	0,194	0,997	0,091
299	0,675	0,257	0,129	0,118
300	0,335	0,112	0,052	0,454
301	0,363	0,124	0,058	0,051
302	0,670	0,249	0,124	0,109
303	0,346	0,118	0,057	0,050
304	0,734	0,252	0,117	0,102
305	0,371	0,125	0,059	0,052
306	0,446	0,169	0,106	0,134

307	0,531	0,183	0,092	0,085
308	0,366	0,125	0,059	0,051
309	1,000	0,673	0,677	0,680
310	0,445	0,180	0,114	0,104
311	0,578	0,210	0,105	0,095
312	0,412	0,141	0,068	0,059
313	0,367	0,126	0,060	0,055
314	0	0	0	0
315	0,674	0,274	0,216	0,323
316	0,522	0,193	0,093	0,082
317	0	0	0	0
318	0	0	0	0
La moyenne	**0,449**	**0,173**	**0,132**	**0,112**
La formule tf*ief				
289	0,375	0,127	0,059	0,309
290	0	0	0	0
291	0,803	0,300	0,166	0,093
292	0,393	0,137	0,070	0,038
293	0,745	0,283	0,142	0,075
294	0,406	0,152	0,073	0,039
295	0,436	0,159	0,081	0,433
296	0,480	0,168	0,867	0,049
297	0,440	0,162	0,803	0,042
298	0,607	0,235	0,122	0,066
299	0,676	0,257	0,129	0,069
300	0,335	0,112	0,052	0,027

301	0,363	0,124	0,058	0,031
302	0,670	0,249	0,124	0,065
303	0,345	0,118	0,057	0,030
304	0,732	0,251	0,117	0,061
305	0,380	0,129	0,061	0,032
306	0,446	0,169	0,106	0,788
307	0,570	0,198	0,100	0,547
308	0,375	0,129	0,061	0,032
309	1	0,673	0,677	0,680
310	0,423	0,165	0,100	0,054
311	0,472	0,174	0,089	0,049
312	0,436	0,150	0,073	0,038
313	0,367	0,126	0,060	0,032
314	0	0	0	0
315	0,674	0,274	0,216	0,188
316	0,554	0,206	0,100	0,053
317	0,573	0,223	0,140	0,088
318	0	0	0	0
La moyenne	**0,503**	**0,202**	**0,174**	**0,148**
La formule tf*idf*ief				
289	0,686	0,270	0,187	0,145
290	0	0	0	0
291	0,845	0,401	0,302	0,211
292	0,689	0,296	0,189	0,116
293	0,864	0,439	0,400	0,279
294	0,901	0,449	0,418	0,278

295	0,699	0,317	0,241	0,178
296	0,862	0,444	0,380	0,273
297	0,645	0,255	0,182	0,145
298	0,894	0,449	0,361	0,264
299	0,742	0,331	0,246	0,172
300	0,758	0,303	0,209	0,159
301	0,898	0,401	0,265	0,189
302	0,766	0,367	0,273	0,192
303	0,702	0,290	0,207	0,166
304	0,815	0,396	0,278	0,198
305	0,875	0,402	0,265	0,169
306	0,678	0,309	0,222	0,158
307	0,719	0,339	0,222	0,163
308	0,668	0,268	0,191	0,155
309	0,943	0,490	0,392	0,341
310	0,759	0,366	0,335	0,278
311	0,795	0,390	0,303	0,212
312	0,878	0,428	0,329	0,216
313	0,749	0,299	0,203	0,155
314	0	0	0	0
315	0,813	0,370	0,275	0,207
316	0,764	0,335	0,235	0,174
317	0	0	0	0
318	0,907	0,456	0,396	0,306
La moyenne	**0,789**	**0,365**	**0,278**	**0,204**

Tableau 4.1 Les résultats des nxCG pour le modèle implémenté et XFIRM testé sur toutes les requêtes.

Durant les tests effectués, certaines requêtes n'ont pas donné des résultats et sont représentées dans le tableau ci-dessus par des zéros (0).

Et voici un tableau comparatif de la moyenne des nxCG pour le système XFIRM et le modèle LSI :

	Xfirm			
	F1	F2	F3	**LSI**
moyenne à 5	0,449	0,503	0,789	0,915
moyenne à 10	0,173	0,202	0,365	0,501
moyenne à 25	**0,132**	0,174	0,278	0,581
moyenne à 50	0,112	0,148	0,204	0,583

Tableau 4.2 Les moyennes des nxCG à 4 niveaux pour XFIRM et LSI

Dans ce qui suit, des histogrammes illustratifs du tableau des gains de modèle LSI et le système XFIRM, ainsi le tableau des moyennes de nxCG pour les deux systèmes :

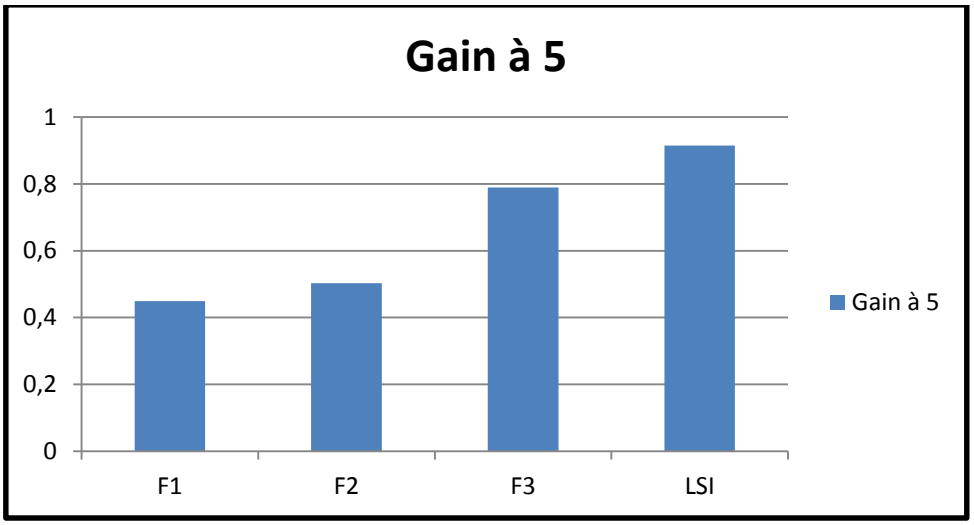


Figure 4.5 Histogramme des moyennes des gains correspondants aux cinq (5) premiers résultats de XFIRM et LSI.

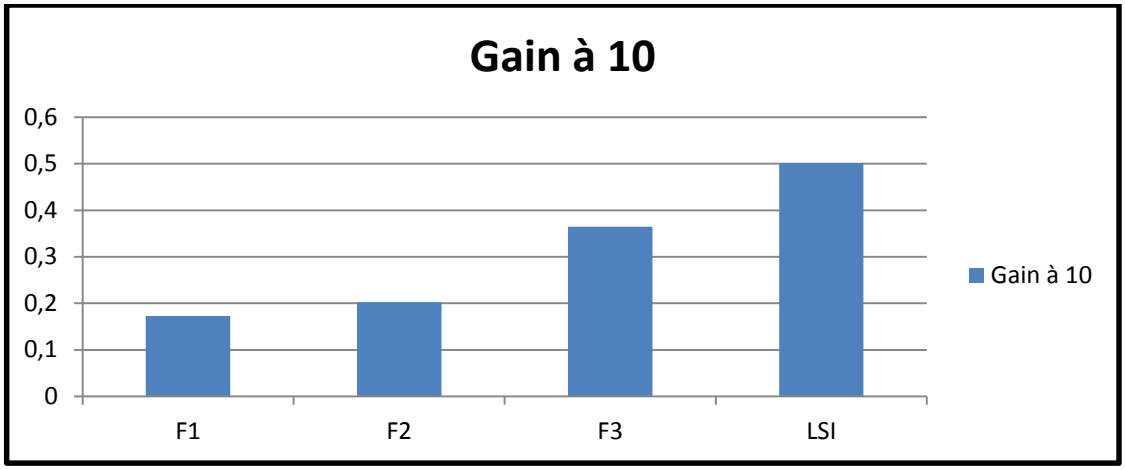


Figure 4.6 Histogramme des moyennes des gains correspondants aux dix (10) premiers résultats de XFIRM et LSI.

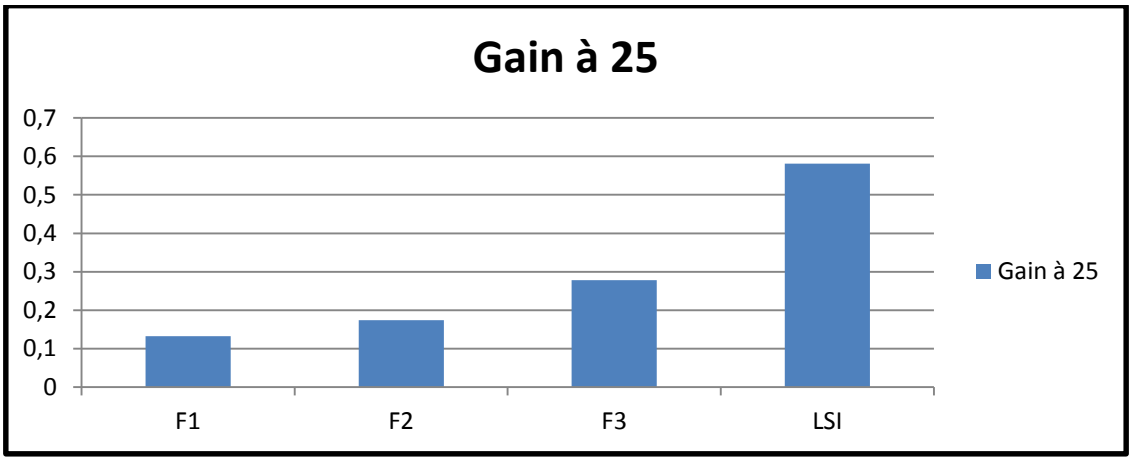


Figure 4.7 Histogramme des moyennes des gains correspondants aux vingt-et-cinq (25) premiers résultats de Xfirm et LSI.

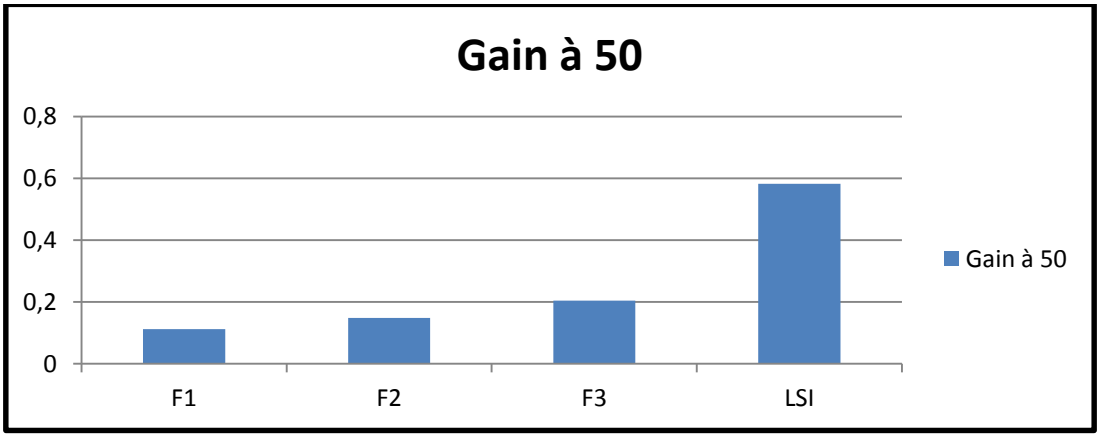


Figure 4.8 Histogramme des moyennes des gains correspondants aux cinquante (50) premiers résultats de XFIRM et LSI.

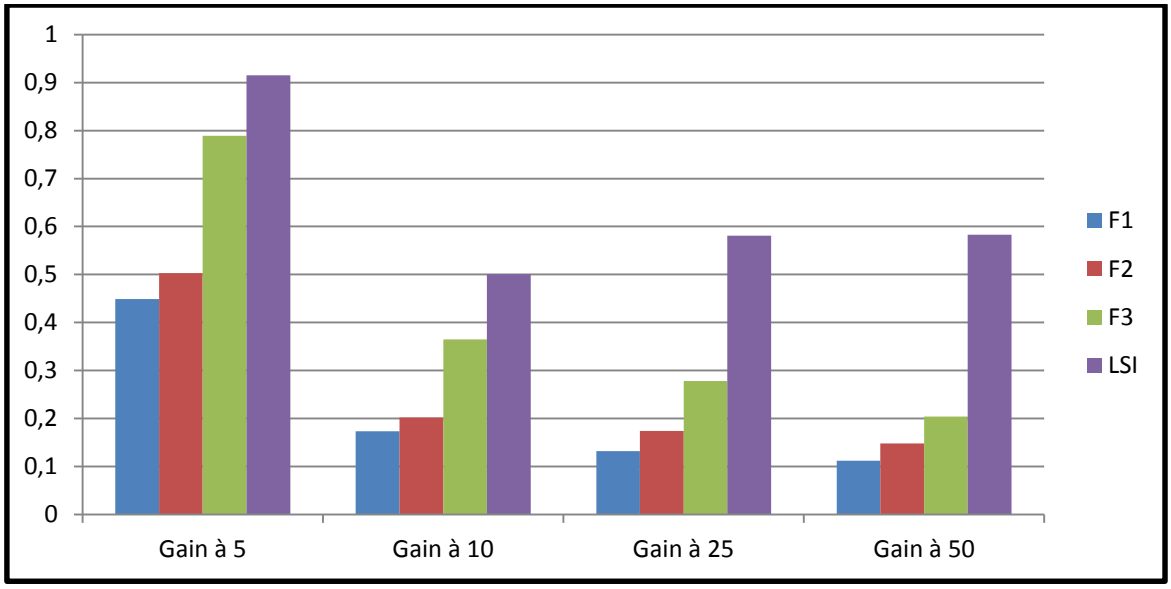


Figure 4.9 Histogramme comparatif des moyennes des gains correspondants aux différents niveaux du calcul pour le modèle LSI et le système XFIRM.

Interprétation des résultats :

Gain / Formule		Gain à 5	Gain à 10	Gain à 25	Gain à 50
F1		0,449	0,173	0,132	0,112
F2		0,503	0,202	0,174	0,148
F3		0,789	0,365	0,278	0,204
LSI	F1	46,6 %	32,8 %	44,9 %	47,1 %
	F2	41,2 %	29,9 %	40,7 %	43,5 %
	F3	12,6 %	13,6 %	30,3 %	37,9 %

Tableau 4.3 Les pourcentages du gain du modèle LSI.

A partir du tableau, en comparant les résultats de notre modèle (LSI) aux formules F1, F2 et F3 de système XFIRM, on constate que LSI a donné les meilleurs résultats.

Conclusion :

Dans ce chapitre, nous avons réalisé l'intégration de notre modèle dans le système XFIRM. On a présenté ainsi les résultats des tests qu'on a évalués et comparés en utilisant le gain d'information cumulé à quatre (04) niveaux.

L'évaluation de modèle proposé s'est effectuée en utilisant un échantillon cohérent de la collection documentaire et l'ensemble des requêtes d'INEX 2006, comme on a présenté des comparaisons concernant les résultats fournis par le système XFIRM .On peut dire que notre système a montré son efficacité par apport au modèle XFIRM vu les résultats obtenus.

La prise en compte de la sémantique dans la recherche d'information dans les documents XML permet l'amélioration et l'augmentation de la performance des SRI par l'obtention des résultats plus pertinents suite à une requête donnée.

Conclusion et perspectives

Conclusion et perspectives

Conclusion général :

Dans ce mémoire nous avons abordé la problématique de recherche d'information sémantique dans des documents semi structurés et nous avons cité la nécessité de développer des techniques permettant d'identifier les éléments les plus pertinents dans un document par apport à une requête donnée. Ces techniques doivent permettre à l'utilisateur d'interroger des documents structurés en spécifiant des conditions sur le contenu textuel et aussi sur la structure selon ses besoins.

Nous avons vu que la prise en compte de la sémantique peut améliorer la performance d'un SRI, par l'illimitation de phénomène de polysémie et de synonymie. Ainsi, nous avons abordé les approches de la RI sémantique dans les documents semi-structurés existantes. A partir de là, nous avons proposé d'étendre le modèle LSI (*latent Semantic Indexing*) proposé en RI classique pour qu'il prenne en charge les documents semi structurées de type XML.

Nous avons implémenté notre modèle sur le système de recherche XFIRM. Ainsi nous avons effectué des expérimentations sur la collection INEX2006 pour évaluer la performance du modèle proposé.

Perspectives :

- L'intégration de la pondération des termes au modèle proposé. Au lieu d'affecter des valeurs binaires représentant l'absence ou la présence (0,1) d'un terme dans un nœud à la matrice termes-NoeudsDocument on pondère les termes.
- Effectuer des tests supplémentaires en faisant varier le paramètre k utilisés dans le modèle implémentée qui réduit la dimension de la matrice termes-NoeudsDocument initiale (avant d'appliquer le SVD).
- L'utilisation ressource sémantique pour palier au problème d'ambiguïté des termes.
- la reformulation des requêtes de manière à retrouver les bons éléments dans les documents XML. en s'appuyant sur des concepts proches définis dans la ressource sémantique telle que la synonymie, généralisation/spécialisation **[Baziz, 2005]**. La reformulation des requêtes est très utile dans le cas où on ne dispose pas de mesures de similarité sémantique entre les concepts de la ressource sémantique utilisée.

Annexe

Annexe

Annexe1 : les technologies de la famille XML

1. Le langage XML :

1.1.Présentation de XML :

XML (eXtensible Markup Language) est un langage extensible générique où il permet à l'utilisateur de créer ses propres balises. De plus, il est un langage de balisage (utilisation de balises) afin d'organiser le document d'une manière hiérarchique et structuration des données pour obtenir un document XML sous forme d'arbre ordonné et étiqueté.

En effet, chaque nœud de cet arbre représente un élément et l'élément lui-même peut avoir des attributs. Les feuilles de l'arbre interprète les éléments textuels.

Le langage XML permet de décrire les données de manière compréhensible par l'homme et la machine.

XML a été développé dans le but d'échanger de données structurées entre les machines par internet. XML peut être défini comme un métalangage. Ce dernier veut dire que XML est un langage pour écrire d'autres langages. Il est à la base d'une collection des nouveaux langages comme : XHTML, MathML …etc, et de nombreuses technologies comme : XSL, Query, XLink …etc. Comme il est illustré dans la figure suivante

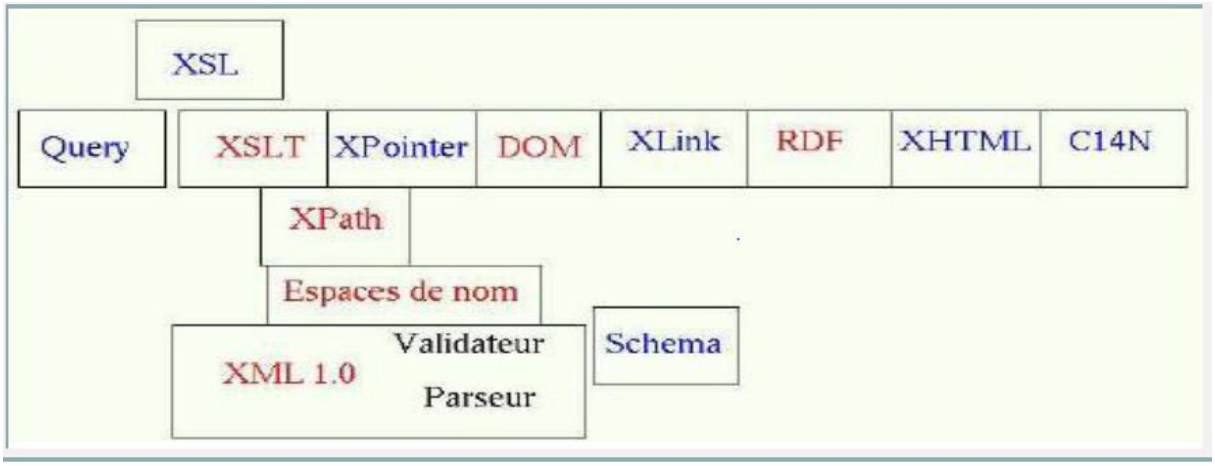


Figure1. Technologies de la famille XML **[Vieillard, 2000]**

XML est classé dans des langages de description. Ce dernier permet de décrire et de structurer un ensemble de données selon des règles et contraintes définies.

1.2.La structure d'un document XML :

Un document XML peut être divisé en deux parties : le prologue qui sert à donner des informations de traitement, le corps ou arbre d'élément qui sert de décrire le contenu

~~sémantique~~ du document (représentation logique du document XML), de plus on citera les commentaires qui sont nécessaires afin de donner une documentation du programmeur.

1.2.1. Les commentaires :

Un commentaire est un texte qui permet de donner une indication de ce que fait le programmeur. Il assure un grand aide pendant la rédaction d'un long document XML, donc il permet d'annoter un document et d'expliquer une partie de celui-ci. En XML, un commentaire a une syntaxe particulière. Il est de la forme suivante :

```
< !- - Ceci est un commentaire - - >
```

1.2.2. Le prologue :

C'est la première ligne d'un document XML. Il contient généralement :

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

Celui-ci indique :

- La version utilisée pour décrire les données est 1.0 (sachant que dans XML il y a deux versions 1.0 et 1.1 et le prologue n'est obligatoire qu'à partir de la version 1.1 mais il est plus conseillé de l'ajouter même en utilisant la version 1.0)
- Le codage utilisé «UTF-8» (c'est le codage par défaut). Il s'agit du jeu de caractères utilisé dans le document XML.
- Standalone permet de savoir si le document XML est autonome ou si un autre document lui est rattaché, autrement dit le document possède des déclarations externes. Dans ce cas, on n'a pas besoin de DTD externe.

1.2.3. Le corps (l'arbre d'éléments) :

Un corps d'un document XML est constitué d'un ensemble de balises décrivant les données. Il sert à construire la structure sémantique du document. Il y a cependant une règle importante à respecter dans la constitution du corps c'est la balise racine (élément racine) du corps, qui va contenir l'ensemble des autres éléments.

- **Elément** : c'est une unité sémantique composée des balises paires (ouvrante et fermante) portantes le nom de l'élément et son contenu.

Il est représenté selon cette syntaxe :

```
<nom-élément> contenu </ nom-élément >
```

Un élément vide ou dit encore une balise unique est représenté < balise/>

- **Attribut** : représente une information secondaire qui sert à donner des renseignements supplémentaires sur son contenu. En effet, l'attribut n'est pas l'information principale que la balise souhaite la transmettre.

Sa syntaxe est la suivante :

< Elément attribut="chaine de caractères" > contenu < /élément >

1.3.Les outils de définition des documents XML :

La structure d'un document XML est définie par une DTD ou un schéma XML :

a. La DTD (Document Type Definition) ou Définition de type de document permet d'établir les règles de définition de la structure d'un document XML, les éléments à inclure, leur type et les valeurs par défaut à leur attribuer.

Il existe deux types de DTD :

- **DTD interne** : qui est écrites directement dans le document XML lui-même ;

```
<? xml version= "1.0" encoding="UTF-8" standalone="yes"? >
< ! DOCTYPE boutique [
< ! ELEMENT boutique (telephone*) >
        < ! ELEMENT telephone (marque, modele) >
        < ! ELEMENT marque (#PCDATA) >
        < ! ELEMENT modele (#PCDATA) >
]>
<boutique>
    <telephone>
        <marque>Samsung</marque>
        <modele>Galaxy S6</modele>
    </telephone>
</boutique>
```

Figure 2. DTD interne

- **DTD externe** : elle est écrite dans un fichier différent séparé de celui du document XML avec l'extension (.dtd) et l'avantage de ce type est la réutilisation.

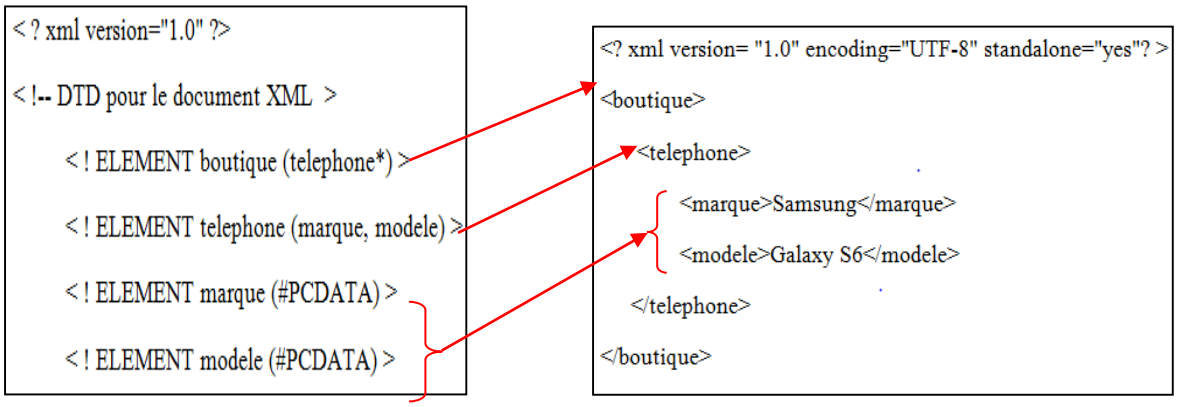


Figure 3. DTD externe

On a parlé déjà du document bien formé, on lui intégrant la DTD on obtiendra le document valide. D'où on peut dire qu'un document valide est un document bien formé conforme à une définition.

En effet, une définition d'un document est un ensemble de règles (décrit la façon dont le document XML est construit) que l'on impose au document.

b. **Le schéma XML** : il s'écrive grâce à XML, il sert à typer les données (création de type de données propres au programmeur). Ainsi, pour utiliser un document XML et le schéma qui lui est associé on a besoin seulement d'un seul outil (c'est le XML). Le schéma XML (avec l'extension .xsd) comporte un prologue, et le corps qui est constitué d'un ensemble de balises. Où on constate la présence de l'élément racine (la balise qui va contenir toutes les autres).

```
< !-- Prologue -->
< ?xml version ="1.0" encoding= "UTF-8">
< !--Elément racine -->
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
</xsd:schema>
```

Figure 4. Exemple d'un schéma XML.

En analysant cet exemple on trouve :

- L'élément racine sous la forme : < xsd :schema></ xsd :schema> (ou tout simplement < xsd :schema/>) ;
- xmlns : qui permet de déclarer un espace de noms (un espace de noms est identifié par URI (Uniform Ressource Identifier) qui assure l'identification d'une ressource de manière unique sur un réseau) ;
- xsd : tous les éléments doivent commencer par xsd.

2.5.L'analyseur (parser) XML :

Le parsing est réalisé par le parser dont son rôle est semblable à celui du compilateur. Un compilateur a deux fonctions :

a. Il réalise l'analyse syntaxique du programme et signale les éventuelles erreurs de compilations ;
b. Il fournit une traduction du programme en langage machine afin que ce dernier puisse être exécuté.

Comme le compilateur le parser va d'abord s'assurer que le document est bien formé (il signalera des erreurs en cas de mal formation). De plus certains parsers dits validant sont capables de vérifier que le fichier XML est conforme au schéma auquel il est lié. Là où le compilateur génère un arbre puis un exécutable utilisable par la machine, le parser s'arrête à la construction de l'arbre.

Annexe

Afin de lire un document XML dans un fichier et exploiter ces données, on utilise essentiellement les deux méthodes appelées SAX et DOM.

2.5.1. DOM (Document Object Model) :

Un document XML est structuré selon une hiérarchie sous forme d'un arbre d'unité d'information appelé nœuds (élément, attribut, valeur). En effet, DOM est une API indépendante de toute plateforme et langage de programmation. Cette technique se base sur la construction d'une arborescence du document XML qui décrit les relations existant entre les nœuds.

Cette API (***A**pplication **P**rogramming **I**nterface*) est basée sur les arbres d'objets chargés en mémoire. Elle permet l'accès direct à tous les nœuds de l'arbre (lire, modifier,…). Par conséquence, les programmes qui utilisent DOM ont généralement une empreinte mémoire volumineuse en cours du traitement.

2.5.2. SAX (Simple API for XML):

SAX est une API qui permet la lecture d'un fichier XML sous la forme linéaire, autrement dit SAX ne prend pas en compte la structure d'un arbre d'un document lors de la manipulation de document XML. Cet analyseur permet de prendre des événements (exemple : ouverture/fermeture d'une balise).

L'analyse d'un document XML par un parser SAX revient à :

- ✓ Détecter les signaux d'ouverture/fermeture d'éléments ;
- ✓ Détecter le contenu textuel ;
- ✓ Détecter les propriétés (couples attributs-valeurs d'un élément) ;
- ✓ …etc

Dans ce type de parseur, le document n'est pas chargé en mémoire ce qui permet d'analyser du document rapidement. Donc gain de mémoire par rapport à un parser de type DOM qui construit un arbre en mémoire. Mais les possibilités de manipulation sont limitées en le comparant avec DOM.

Annexe

Annexe2 : les mesures de similarité sémantique

1. Mesures de similarité sémantique :

1.1.Notion de la similarité sémantique entre concepts :

De nombreuses approches ont été proposées pour évaluer la similarité sémantique entre deux concepts. Ces approches se divisent **[Slimani et al. 2006]** en trois catégories : les approches basées sur les arcs, les approches basées sur le contenu informationnel et les approches hybrides.

Le rôle des mesures de similarité est d'évaluer la ressemblance entre les concepts (proximité sémantique entre les concepts auxquels les termes des requêtes et document appartiennent). Ainsi les mesures de similarité ont un rôle important, lors de processus de recherche d'information (dans le processus de désambiguïsation des concepts, la pondération des concepts et l'évaluation de la pertinence).

La similarité est la fonction inverse de la distance, plus deux concepts sont similaires, moins ils sont distants.

La distance est exprimée par une mesure δ représente comme suit :

a. nullité de la distance d'un concept avec lui-même : $\delta(c_i, c_i) = 0$
b. symétrie : $\delta(c_i, c_j) = \delta(c_i, c_j)$
c. inégalité triangulaire : $\delta(c_i, c_j) + \delta(c_j, c_k) \geq \delta(c_i, c_k)$

Où c_i, c_j et c_k sont trois concepts quelconques.

La mesure de similarité entre deux entités A et B est définissent selon Tversky **[Tversky, 1977]** comme une fonction des caractéristiques communes (dénoté par comm) et des caractéristiques différentes (dénoté par $diff$). On utilise trois constantes α, β, γ pour définir la similarité entre A et B :

$$Sim(A,B) = \alpha.comm(A,B) - \beta.diff(A,B) - \gamma.diff(B,A)$$

Plus les entités ont des caractéristiques en commun, plus elles sont similaires.

1.2.Technique de calcul des mesures de similarité sémantique :

On distingue trois méthodes de calcul de mesures de similarité sémantique les méthodes basées sur le comptage de lien, les mesures basées sur le contenu informatif et les méthodes hybrides.

1.2.1. Méthode basées sur le comptage de lien :

Ce type de mesure se base sur la structure de la ressource sémantique, qui propose un comptage du nombre d'arcs séparant deux concepts. En se servant de la structure hiérarchique de l'ontologie afin de déterminer la similarité sémantique entre les concepts. Parmi les travaux on peut citer :

- **La mesure de Wu-Palmer** [Wu-Palmer, 1994].

La similarité est présentée dans une ontologie par rapport à la distance qui sépare deux concepts dans la hiérarchie en se basant sur leur position par rapport à la racine. La similarité entre c_1 et c_2 est :

$$Sim_{WPalmer}(c_1, c2) = \frac{2*prof(c)}{dist(c_1,c)+dist(c_2,c)+2*dist(c)}$$

Où :

- *c :* est le concept le plus spécifique qui représente les deux concepts c_1 et c_2
- prof(c) : est le nombre d'arcs qui sépare **c** de la racine
- dist(ci,c) : le nombre d'arcs qui séparent ci de **c**.

- **La mesure de Resnik** [Resnik P.1995]:

La similarité se calcule par rapport à la longueur des chemins qui relient deux concepts dans la hiérarchie. La similarité entre c_1 et c_2 est :

$$Sim_{RensikEdge}(c_1, c_2) = 2D - len(c_1, c_2)$$

Où

- D : est le maximum des longueurs des chemins possibles qui relient c_1 et c_2.
- $len(c_1, c_2)$:le plus petit chemin entre c_1 et c_2.

1.2.2. Méthodes basée sur le contenu informationnel :

Resnik **[Resnik P.1999]** a introduit La notion de contenu informationnel (CI). Le contenu informationnel d'un concept représente la pertinence d'un concept dans le corpus en considérant sa spécificité ou sa généralité. Dans le but de retrouver le contenu informationnel, La fréquence de concepts dans le corpus est calculée. Cette fréquence regroupe la fréquence d'apparition du concept lui-même également des concepts qu'il subsume. La formule est la suivante :

$$CI(c) = -\log(P(c))$$

Où :

P(c) est la probabilité de trouver un mot du corpus qui soit une instance du concept c .

$$P(c) = \frac{Freq(c)}{N}$$

Où : N est la taille totale d'échantillon de texte

Freq(c) est la fréquence d'occurrence des mots dénotant le concept c dans la collection.

Parmi les mesures basées sur le contenu informationnel on peut citer :

- **La mesure de Resnik** [Resnik P.1999].

La similarité entre deux concepts est dépendante de l'information qu'ils partagent en commun La similarité entre c_1 et c_2 est définie de la façon suivante :

$$sim_{Rensik}(c_1, c2) = CI(c)$$

Où :

c est le concept le plus spécifique qui subsume les deux concepts c_1 et c_2.

1.2.3. Méthodes hybrides :

Ces méthodes sont basées sur un modèle mixte qui regroupe les deux approches vu précédemment : approches basées sur le comptage des liens et approches basées sur le contenu informationnel qui est considéré comme facteur de décision.

- **La mesure de Jiang et Corath** [Jiang et al., 1997] :

Cette mesure relié le contenu informationnel du concept le plus particulier (dénoté par c) à ceux des concepts et le nombre d'arcs. La similarité est représenté par :

$$sim_{Jiang}(c_1, c_2) = \frac{1}{CI(c_1) + CI(c_2) - 2 * CI(c)}$$

Annexe

Annexe 3 : Le système de recherche XFIRM

1. Présentation générale du modèle XFIRM :

XFIRM est un modèle flexible pour la recherche d'information dans les documents semi-structurés. Ce modèle a été proposé par SAUVAGNAT en 2005.

XFIRM est un système orienté pertinence, il repose sur la propagation de pertinence à travers les éléments constituants un document : un score est calculé pour les nœuds feuilles ensuite il est propagé dans l'arbre du document et il est diminué durant la propagation pour répondre au critère de spécification d'une unité pour une requête fournie. Ce modèle repose sur la représentation de l'information structurelle et l'information textuelle des documents en utilisant une représentation arborescente où l'information textuelle se trouve au niveau des nœuds feuilles.

Le langage d'interrogation proposé dans ce modèle permet à l'utilisateur d'exprimer son besoin selon son degré de précision. La formulation des requêtes peut être à base de mots-clés ou bien la requête peut contenir des contraintes sur la structure des documents. Le premier type est utilisé dans le cas où l'utilisateur n'a aucune idée de l'information qu'il désire voir renvoyer. Le second type, ces conditions de structure lui permet d'indiquer le type de l'unité d'information qu'il désire voir retournées par le système. Lorsque la requête contient des conditions de structure, elle peut être assimilée à un arbre, comme les documents.

Dans le but d'assurer un fonctionnement cohérent, il présente un ensemble de modules qui ont donné lieu au développement d'un prototype permettant l'indexation de la collection documentaire XML. Ce prototype est entièrement réalisé en langage JAVA en utilisant des API (Application Progrming Interface) telles que SAX (Simple API for XML) de Xerces pour parser les documents XML et un JDBC pour l'accès aux BDD du système. Ses composantes principales ainsi que leurs liaisons et structures sont présentées par l'architecture du prototype présentée dans la figure suivante

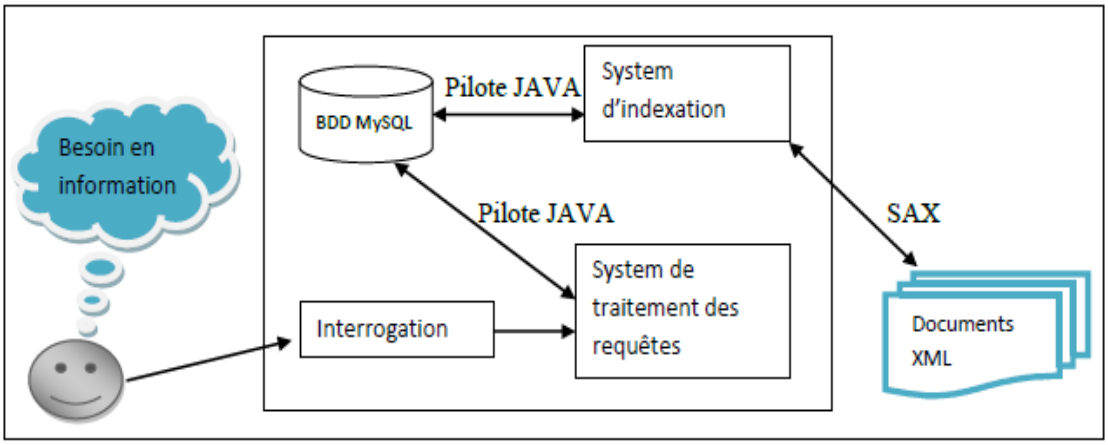


Figure1. Architecture de base du modèle XFIRM

Description :

XFIRM est constitué principalement d'une BDD (Base De Données) MySQL qui consiste en l'élément centrale de tout le système et contenant ses index, autour de la BDD sont placés des éléments ayant des accès à celle-ci via un pilote JAVA, ces élément sont des modules qui construisent la partie dynamique du système et sont : *Le module d'indexation, le module d'interrogation* et *le module de traitement des requêtes* (orientées contenu et orientées

contenu et structure). Afin de répondre aux exigences utilisateur en structure et contenu en lui retournant une liste triée d'éléments répondants à sa requête.

2. Processus d'indexation :

L'indexation sous XFIRM passe par certaines étapes. Pour crées les index, on suit les étapes suivantes :

1. Création des tables d'indexation pour le stockage des index,
2. Traitement des documents de la collection en remplissant les tables correspondantes : *Document, Path, Link, Ttag, Termtemp, Attributs, Collection, Image, NodeToTerm.*
3. Remplissage de la table *Terme* à partir de la table *TermTemp* qui est une table temporaire à supprimer à la fin de l'indexation,
4. Remplissage de la table *Dict*,
5. Définition des contraintes des lignes des tables,
6. Suppression des tables temporaires.

3. Présentation des données de base : les documents XML

3.1 Modèle de représentation :

Pour représenter des documents XML, XFIRM s'est basé sur une simplification des modèles XPath et XQuery où un nœud peut être : un document entier, élément d'un document ou attribut d'un élément ou encore du texte, un espace de noms, une instruction, ou alors un commentaire.

Un document structuré d est un arbre, défini par les ensembles N, F, A et L.
Document Structuré : d = (N, F, A, L)
Avec N = {n_1, n_2, ...} l'ensemble des nœuds internes, F = {nf_1, nf_2, ...} l'ensemble des nœuds feuilles, A = {a_1, a_2, ...} l'ensemble des attributs et *L* l'ensemble des liaisons (arcs orientés) présentés par des paires *(u,v)* des éléments de *N*, *F* ou *A* avec *v* successeur de *u* en conservant toute les propriétés d'un arbre.

3.2 Pondération des termes des nœuds feuilles :

Comme l'information textuelle se trouve au sein des nœuds feuilles, alors la pondération des termes se focalisera sur ce niveau. Un nœud feuilles contenant du texte, en XFIRM est représenté par :

$$nf_i = \{(t_1, w_1^i), (t_2, w_2^i), \dots (t_j, w_j^i)\}$$

Où : W_j^i est le poids du terme t_j dans le nœud feuille nf_i.
Selon Sauvagnat, le poids d'un terme va traduire son importance non seulement au sein du nœud feuille qui le contient mais au sein du document en cours ainsi que toute la collection et son calcul dépend de la formule de pondération considérée et peut être en fonction de :
- tf_j^i: fréquence du terme t_j dans le nœud feuille nf_i
- idf_j : fréquence inverse du document en cours pour le terme t_j calculée par :

$$idf_j = \log(\frac{|D|}{|d_j|})$$

Où |D| est le nombre total de document de la collection
et $|d_j|$ est le nombre de documents contenant le terme t_j

- ief_j la fréquence inverse d'élément pour le terme t_j (adaptation de l'idf_j à la granularité : nœuds feuilles) calculée par :

$$ief_j = \log(\frac{|Fc|}{|nf_j|})$$

$|Fc|$: nombre des nœuds feuilles de la collection
$|nf_j|$: nombre des nœuds feuille contenant t_j dans la collection

3.3 La représentation des données :

Le système XFIRM propose la construction de la structure des index au préalable afin de l'utiliser dans les différentes contraintes exprimées dans les requêtes. Les index sont stockés sous forme de tables dans une base de données relationnelle MySQL. Pour obtenir les différents index, les documents à indexer sont parcourus à l'aide d'un parseur SAX.

La description des tables est présentée dans le tableau suivant :

Table	**Rôle**	**Description des champs**	**Remarques**
Document: (doc_id NUMBER(38), primary key, document varchar(300), index, term_nb NUMBER(38), deb NUMBER(38), fin NUMBER(38), nbr_niv NUMBER(38), kuid NUMBER(38))	Contient les informations sur les documents de la collection	**doc_id:** identifiant d'un document **document:** le chemin de stockage d'un document **term_nb:** nombre de termes d'un document **deb:** identifiant du premier noeud d'un document **fin:** identifiant du dernier nœud d'un document **nbr_niv:** nombre de niveaux d'un document **kuid :** valeur calculée permet de restaurer la structure d'un document	

Tag : (tag_id NUMBER(38), primary key, tag VARCHAR(100), index)	contient des informations sur les balises de la collection	**tag_id:** identifiant d'une balise **tag:** nom d'une balise	
Path: (path_id NUMBER(38), primary key, doc_id NUMBER(38), foreign key references Document (doc_id), index, tag_id NUMBER(38), foreign key references Tag(tag_id), ordre NUMBER(38), nbrDirectFils NUMBER(38), path_uid VARCHAR(150), index, nbrTermUniq NUMBER(38), nbrTermsTot NUMBER(38), nbrFilsText NUMBER(38))	contient des informations sur les nœuds (éléments) de la Collection	**path_id:** identifiant d'un noeud **doc_id:** identifiant du document associé au nœud **tag_id :** identifiant d'une balise **ordre:** ordre d'un nœud **nbrDirectFils :** nombre de nœuds fils direct d'un nœud **path_uid:** identifiant unique d'un nœud **nbrTermUniq:** nombre de termes uniques d'un nœud **nbrTermsTot:** nombre de termes total d'un noeud **nbrFilsText :** nombre de nœuds textuels d'un nœud	
Termes : (term_id NUMBER(38), primary key, term VARCHAR(150), index, freqColl NUMBER(38), nbrDoc NUMBER(38), nbrNoeuds NUMBER(38), poids blob)	contient des informations sur les termes de la collection	**term_id:** identifiant d'un terme **term :** nom d'un terme **freqColl:** fréquence d'un terme dans la collection **nbrDoc:** nombre de	le champ « poids » est un blob contenant la liste de tous les nœuds contenant le terme, avec la fréquence du terme dans le nœud et ses

		document contenant le terme **nbrNoeuds:** nombre de nœuds contenant le terme **poids :** le poids d'un terme	positions.
AttributsType : (type_att_id NUMBER(38), primary key, att_name VARCHAR(100))	contient des informations sur les noms d'attributs présents dans la collection	**type_att_id :** identifiant de type d'attribut **att_name:** nom d'attribut	
Attributs : (node_id NUMBER(38), foreign key references Path(path_id), type_att_id NUMBER(38), references AttributsType(type_att_id), att_val VARCHAR(100), primary key (node_id, type_att_id))	permet de récupérer la valeur des attributs	**node_id:** identifiant du nœud contenant l'attribut **type_att_id:** identifiant de type d'attribut **att_val :** valeur d'attribut	
Dict : (tag_id NUMBER(38), primary key, foreign key references Tag(tag_id) list_tag_id blob)	permet d'établir des équivalences de balises dans la collection	**tag_id :** identifiant d'une balise **list_tag_id :** liste des identifiants de balises	le champ « list_tag_id » est un blob contenant la liste de toutes les balises équivalentes au tag_id associé.
Collection : (nbrDoc NUMBER(38), nbrNods NUMBER(38), nbrNodText NUMBER(38))	contient des informations générales sur la collection	**nbrDoc:** nombre de documents dans la collection **nbrNods:** nombre de nœuds dans la collection **nbrNodText:** nombre de nœuds feuilles dans	

		la collection	
Images : (nodeIm_id NUMBER(38), references Path(path_id) , trmIm VARCHAR(100), index)	contient des informations sur les images de la collection	**nodeIm_id :** identifiant d'une image **trmIm :** nom d'une image	
Links : (path_id NUMBER(38), references Path(path_id), doc_name VARCHAR(100))	contient des informations sur les liens de la collection	**path_id :** identifiant du nœud portant un lien **doc_name:** texte du lien	
NodeToTerm : (node_id NUMBER(38), primary key nbrTrmUq NUMBER(38), nbrTrmTot NUMBER(38), listTerm blob)	permet de connaître les termes présents dans un nœud donné	**node_id:** valeur pre d'un nœud feuille **nbrTrmUq:** nombre de terme unique d'un nœud feuille **nbrTrmTot:** nombre de termes total d'un nœud feuille **listTerm :** liste de terme d'un nœud feuille	cette table est utilisée seulement dans le cadre de la reformulation de requêtes. Le champ « listTerm » est un blob contenant la liste des termes du nœud node_id.
termTemp: (term_id NUMBER(38), term VARCHAR(100), doc_id NUMBER(38), node_id NUMBER(38), int NUMER(38))	A usage temporaire	**Term_id** : identifiant d'un terme **term :** texte du terme **doc_id :** identifiant du document contenant le terme **node_id:** identifiant du noeud contenant le terme **int :** position du terme dans le nœud	est une table temporaire utilisée afin de remplir la table Termes. Elle est supprimée à la fin de l'indexation

Tableau : Description des tables de la BDD du système XFIRM

4. Langage de requêtes :

Le langage de requête proposé par Sauvagnat est caractérisé une syntaxe simple qui est vu comme une simplification du langage XPath. En ce qui concerne la formulation des requêtes

est à base de simples mots clés, sans précision sur la structure, avec la possibilité de formuler des contraintes sur la structure des documents. De plus, il donne la possibilité de formuler des requêtes plus complexes, en introduisant la notion de hiérarchie entre les différentes contraintes de structure.

Le langage de requêtes XFIRM :

Le langage de requête XFIRM donne à l'utilisateur la possibilité de formuler son besoin selon quatre degrés de précision.

Les requêtes de précision P1 sont utilisées si l'utilisateur recherche simplement de l'information et que le type de l'unité d'information renvoyée lui importe peu pourvu qu'elle réponde à son besoin. Elles sont formulées avec des mots-clés simples, qui pourront être reliés par des opérateurs (opérateurs booléens, opérateurs d'importance). La recherche sur des expressions est aussi possible, en encadrant les expressions de " ".

Les requêtes de précision P2 permettent d'ajouter des contraintes sur la structure des documents. Ces requêtes sont exprimées par le nom d'un élément, et éventuellement préciser son besoin sur cet élément en ajoutant des conditions sur son contenu ou la valeur de ses attributs. L'élément retourné à l'utilisateur est donc l'élément spécifié dans la requête.

Les requêtes de précision P3 permettent d'ajouter la notion de hiérarchie entre les différentes conditions de structures (requêtes de type P2), qui sont alors séparées par le signe « // ».

Dans les requêtes de type P3, les nœuds retournés à l'utilisateur sont par défaut ceux spécifiés dans la première requête de type P2. Si l'utilisateur a une idée plus précise de ce qu'il recherche, il pourra spécifier l'unité d'information qu'il désire voir retournée. Cette unité d'information élément cible. Cet élément cible est spécifié grâce au signe "ec :" précédant une requête de type P2. Ainsi la requête de précision P4.

Exemple de requêtes :

P1 : +internet – "moteur de recherche"

P2 : titre["moteur de recherche"] ET section [@num=1] : signifie que l'utilisateur souhaite une unité d'information contenant à la fois un élément titre sur "moteur de recherche" ety un élément section ayant un attribut num de valeur 1.

P3 : //article[] // titre["moteurs de recherche"] ET section[internet google]

Signifie que l'utilisateur souhaite obtenir respectivement un nœud article ayant pour descendant un élément titre contenant les termes "moteurs de recherche" et un élément section parlant de internet et de google.

P4 : //article[@date-publi=2000]// ec : corps[] // par[google] ET sous-titre ["moteurs de recherche"]

Signifie que l'utilisateur souhaite obtenir un nœud corps ayant pour ancêtre un nœud article dont l'attribut date-publi vaut 2000 et pour descendant un nœud par parlant de google et un nœud sous-titre contenant l'expression "moteurs de recherche".

Annexe

5. Les répertoires système :

5.1 La structure des répertoires de XFIRM :

XFIRM comme étant un ensemble de répertoires (concernant un environnement de travail Linux), le système est composé des huit principaux répertoires, décris ainsi :

- **Config :** contient les fichiers nécessaires à la configuration du moteur (la connexion à la BDD et équivalence de balises).
- **Doc :** contient la javadoc ou la documentation associée au projet.
- **Exemples :** contient des exemples de lancement de recherche et d'indexation en shells ainsi que les fichiers requêtes et résultats.
- **Import :** contient toutes les librairies Java (les Jar) nécessaires à la compilation et exécution des classes.
- **ProceduresPlsql :** contient les procédures PL/SQL à exécuter sous Oracle avant toute indexation d'une nouvelle base.
- **Script :** contient les shells de lancement de la compilation, génération de la documentation et sauvegarde ou restauration de la BDD.
- **Xfirm :** ce répertoire contenant le code source du projet (les .java et les .classe) dont les packages sont décrit prochainement.
- **Utils :** c'est l'exécutable javacc pour la compilation des grammaires (les fichiers *.jj).

5.2 Utilisation des répertoires XFIRM :

Afin de permettre l'utilisation du système XFIRM et de lancer les différentes commandes, sous linux et à travers un terminal, se placer dans le répertoire contenant le dossier XFIRM et suivre certaines consignes :

5.2.1 Compilation du code source :

A partir du terminal, lancer le script de compilation du code source du système par la commande : *./script/compile.sh*.

Ce script génère les .class dans les répertoires contenant les .java ainsi que la documentation dans le répertoire doc.

5.2.2 Création des tables dans la BDD :

Pour une collection donnée, XFIRM doit créer les tables d'index à cette base documentaire en supprimant les anciennes tables d'une collection déjà utilisée (si elles existent) via la commande suivante :

Java xfirm/test/CreateTables cheminDuFichierDeConfiguration nomDeLaCollection

5.2.3 Indexation de la collection :

Après avoir créé les tables de la BD qui conviennes à la collection, lancer l'indexation de ses données via la commande suivante :

Java xfirm/test/TestIndexer P1 P2 P3 P4 P5 P6 P7

Telle que les $P_{i,(1..7)}$ sont décrits ainsi :

- *P1* : chemin de la collection à indexer

- *P2* : fichier de configuration de la BD pour Oracle
- *P3* : nom de la collection utilisé pour la création des tables dans la base Oracle
- *P4* : le chemin de fichier dictionnaire qui est normalement localisé sous config/dict/
- *P5* : *ATTO* pour faire l'indexation des attributs, *ATTN* pour ne pas faire
- *P6* : *NTO* pour faire l'indexation *NodeToTerm*, *NTN* pour ne pas faire.
- *P7* : *-n* ou *-u* : *-n* pour créer une nouvelle base et *-u* pour mettre a jour une base qui existe déjà

Remarques:

- Si *-n* (création de la base de données), on ajoute les contraintes à la fin de l'indexation.
- Si *-u* (mise à jour de la base de données qui existe déjà), on vérifie les contraintes en même temps que l'insertion.

Et voici un exemple, issu de la documentation XFIRM, sur le lancement de d'indexation de la collection I3E sans indexation des attributs en INEX :

>java xfirm/test/TestIndexer /usr/local/EQ-SIG/CORPUS/INEX/ADHOC/1.4/xml config/connect/OracleConfig.txt I3E config/dict/dictIEE.txt ATTN NTN –n

5.2.4 La recherche simple:

Concernant la recherche, voici un test simple effectué à la réalisation de XFIRM en utilisant la classe permettant de lancer les requêtes à la ligne de commandes *xfirm/test/TestQuery.java* où la recherche est lancée par la commande :

> java xfirm/test/TestQuery p1 p2 p3 p4 p5 p6 p7 [p8 p9 p10 | p8 p9]

- *p1*: fichier de configuration de la BD pour Oracle
- *p2* : requête, entre "" (CO ou CAS)
- *p3* : nom de la collection utilisée pour la création des tables dans la base Oracle
- *p4* : paramètre max :
  - dans le cas de requête CO il représente le nombre maximum de documents utilisés pour récolter les nœuds feuilles (250 ou 500);
  - dans le cas de requête CAS il représente le nombre maximum d'éléments de la structure à rechercher par sous-requête
- *p5* : formule de calcul du poids des nœuds feuilles:
  - 1 = *F1 = tf*idf* ;
  - 2 = *F2 = tf*ief*;
  - 3 = *F3 = tf*idf*ief* ;
- *p6* : type de propagation utilisée
  - 1 : propagation avec *overlap* ;
  - 2: propagation sans *overlap*
- *p7*: type des requêtes
  - CO
  - CAS

Si le type des requêtes est CO

- *p8* : valeur du paramètre alpha (entre 0 et 1)

- *p9* : valeur du paramètre seuil (nombre de termes minimum dans les résultats)
- *p10* : valeur du paramètre pivot (entre 0 et 1)

Si le type des requêtes est CAS

- *p8* : valeur du paramètre *alpha* (entre 0 et 1)
- *p9* : valeur du paramètre *strict* :
  - 1 : indique que les conditions de structure sur les éléments support de la requête doivent être traitées de manière stricte (c'est-à-dire qu'un élément cible n'est pas renvoyé si les conditions sur les éléments supports ancêtres ou descendants ne sont pas respectées)
  - 0 : indique que les conditions de structure sur les éléments support de la requête doivent être traitées de manière vague

Examples de lancement:

> java xfirm/test/TestQuery config/connect/OracleConfig.txt "topics maps –web" I3E 250 3 2 CO 0.6 0 0.9

Pour une requête de type CO sur la collection I3E.

> java xfirm/test/TestQuery config/connect/OracleConfig.txt "article[] AND p[object database] // ec: p[version management]" I3E 250 3 2 CAS 0.6 0

Pour requête de type CAS sur la même collection.

5.2.5 Recherche dans le cadre d'INEX:

Pendant la compagne d'évaluation INEX, la recherche en utilisant le système XFIRM s'est effectué en deux étapes principales :

- Lancer l'exécution du run
- Formater le run

5.2.5.1 Lancement de l'exécution du run

Il faut utiliser la classe *xfirm/inex/TestInexRun.java.*

Lancement :

> nohup java xfirm/inex/TestInexRun p1 p2 p3 p4 p4 p5 p6 p7 p8

- *p1* : fichier contenant les paramètres de connexion à la base
- *p2* : nom du fichier contenant les requêtes (la premier colonne du fichier indique le numéro de la requête, séparé de la requête par une virgule)
- *p3* : chemin jusqu'aux fichiers résultats + premières lettres du fichier résultat
- *p4* : nom de la collection utilisé pour la création des tables dans la base Oracle
- *p5* : paramètre max :
  - dans le cas de requête CO il représente le nombre maximum de documents utilisés pour récolter les nœuds feuilles (250 ou 500);
  - dans le cas de requête CAS il représente le nombre maximum d'éléments de la structure à rechercher par sous-requête
- *p6* : formule de calcul du poids des nœuds feuilles:
  - 1=*F1=tf*idf* ;

- 2=*F2= tf*ief*;
- 3=*F3=tf*idf*ief*

- *p7* : type de propagation utilisée
  - 1 : propagation avec *overlap* ;
  - 2 : propagation sans *overlap*
- *p8*: type des requêtes
  - CO
  - CAS

Si le type des requêtes est CO :

- *p9* : valeur du paramètre *alpha* (entre 0 et 1)
- *p10* : valeur du paramètre *seuil* (nombre de termes minimum dans les résultats)
- *p11* : valeur du paramètre *pivot* (entre 0 et 1)

Si le type des requêtes est CAS :

- *p9* : valeur du paramètre *alpha* (entre 0 et 1)
- *p10* : valeur du paramètre *strict* :
  - 1 : indique que les conditions de structure sur les éléments support de la requête doivent être traitées de manière stricte (c'est-à-dire qu'un élément cible n'est pas renvoyé si les conditions sur les éléments supports ancêtres ou descendants ne sont pas respectées)
  - 0 : indique que les conditions de structure sur les éléments support de la requête doivent être traitées de manière vague

Exemples de lancement :

java xfirm/inex/TestInexRun config/connect/OracleConfig.txt examples/queries/CO2004.q examples/results/co/co I3E 250 3 2 CO 0.6 0 0.9

Pour une requête de type CO.

java xfirm/inex/TestInexRun config/connect/OracleConfig.txt examples/queries/CAS2004.q examples/results/cos/cos I3E 5000 3 2 CAS 0.6 1

Pour une requête de type CAS.

5.2.5.2 Lancement du formatage du run

Il faut utiliser la classe xfirm/inex/TestInexFormat.java

Lancement :

java xfirm/inex/TestInexFormat p1 p2 p3 p4 p5 p6 p7

- *p1* : type de la requête : CO / CAS
- *p2* : année de la campagnie : 2003 / 2004 / 2005 / 2006 / 2007 / 2008
- *p3* : chemin jusqu'aux fichiers résultats + premières lettres des fichiers résultat
- *p4* : nom de la collection utilisée pour la création des tables dans la base Oracle
- *p5* : fichier contenant les paramètres de connexion à la base
- *p6* : *rsv* : 1 si affichage du *rsv*, 0 si non
- p7 : *rank* : 1 si affichage du *rank*, 0 si non

Annexe

Exemples de lancement :
java xfirm/inex/TestInexFormat CO 2004 examples/results/co/co I3E config/connect/OracleConfig.txt 1 0
Pour une requête de type CO, dans la compagnie 2004 dont le résultat est ciblé au fichier *examples/results/co/co* sur la collection I3E avec affichage du *rsv*.

5.3 Les packages du répertoire Xfirm

Le répertoire *xfirm* qui représente le code source de modèle contient les packages suivants :

- **analysis** : ce package contient toutes les classes qui vont permettre de parcourir les documents, et d'extraire les unités d'indexation ainsi que les attributs associés. Il est principalement utilisé lors de l'indexation.
- **analysis.standard** : ce package propose une implémentation standard du package analysis. La grammaire *StandardTokenizer.jj* définit ce qu'est une unité d'indexation.
- **document** : ce package contient la définition des principaux objets composant un document XML (*Document, Node*, …). Il est principalement utilisé lors de l'indexation
- **index** : les objets de ce package sont en fait des éléments qui vont aller dans les index (c'est-à-dire les tables de la BD Oracle). Les instanciations de la classe IndexWriter permettent en outre de construire ces index.
- **inex** : Ce package contient les exécutables spécifiques à INEX.
- **queryParser** : ce package contient la grammaire des requêtes orientées contenu (fichier *QueryParser.jj*).
- **search** : ce package fournit des classes permettant de traiter des requêtes orientées contenu (composées de simples mots-clés).
- **store** : les classes de ce package permettent de créer ou lire les index (*BaseWriter* et *BaseReader*), grâce à une liaison directe avec la BD Oracle.
- **structureQueryParser** : ce package contient la grammaire des requêtes orientées contenu et structure (fichier *StructureQueryparser.jj*).
- **structureSearch** : Ce package fournit des classes permettant de traiter des requêtes orientées contenu et structure (mots_clés+ conditions de structure).
- **test** : Ce package contient des classes de test pour l'indexation et l'interrogation.
- **tree** : Ce package permet de charger les documents indexés en mémoire et de reconstruire leur structure arborescente. Il est principalement utilisé pendant la recherche.
- **util** : Ce package contient des classes "outils" qui peuvent servir dans les autres packages.

6. Evaluation des requêtes orientées contenu :

L'objectif du traitement des requêtes orientées contenu (requêtes de type P1) est de retrouver des sous arbres de taille minimale qui répondent de manière exhaustive à la requête. Ce traitement est effectué comme suit :

1. en premier lieu : l'évaluation de la similarité des nœuds feuilles de l'index à la requête (i.e. calcul du score des nœuds feuilles),

2. En second lieu : la recherche des sous arbres pertinents et informatifs. Pour cela, la dimension d'informativité des sous arbres est évaluée :

– en propageant vers le haut le score des feuilles dans l'arbre du document, et ce en privilégiant les nœuds les plus porteurs d'informations,

– et en propageant vers le bas le score du document dans sa globalité, afin de tenir compte du contexte du sous arbre dans l'évaluation de sa pertinence.

7. Calcul du score des nœuds feuilles :

XFIRM représente les requêtes indépendamment de leurs contenus, par la représentation suivante :

$$q = \{(t_1, w_1^q), (t_2, w_2^q), \dots (t_k, w_k^q)\}$$

$_{(1,k)}$ Étant les termes de la requête q et w_i^q leurs poids.
Afin d'évaluer la similarité requête-nœuds feuilles, on calcule les scores des nœuds feuilles identifiés dans l'arbre du document par la fonction de similarité *RSV(q , nf)* (Retrieval Status Value) :

$$RSV(q, nf) = \sum_{i=1}^{k} w_i^q * w_i^{nf}$$

Où : w_i^q et w_i^{nf} sont respectivement le poids du terme t_i dans la requête q et le poids du terme t_i dans le nœud feuille nf, calculés par les formules de pondération décrite plus haut.

Propagation de pertinence :

Après l'étape de calcule des valeurs de similarité pour les nœuds feuilles, on arrive à l'étape de calcule de valeur de pertinence pour chaque nœud de l'arbre du document en question, et selon ce modèle **[Sauvagnat, 2005]**, plus la distance est grande entre un nœud et son ancêtre, moins qu'il tend vers la pertinence recherchée, cette intuition est modélisée par *dist(n , nf_k)* qui décrit la distance entre un nœud feuille nf_k et son ancêtre *n*. Il parait aussi intuitif que plus un nœud possède de nœuds feuilles pertinents, plus il est pertinent d'où l'introduction du paramètre $|F_n^p|$ qui le représente. Donc, la valeur de pertinence *Pn* d'un nœud feuille est calculée selon Sauvagnat par :

$$Pn = \left|F_n^p\right|. \sum_{nf_k \epsilon Fn} \alpha^{dist(n,nf_k)-1} * (RSVm(q, nf_k))$$

Où :
$|F_n^p|$ Nombre de nœuds feuilles pertinents du nœud *n*.
$dist\ n, nf_k$ nombre d'arcs séparant *n* et nf_k
Fn ensemble des *nf* descendants de *n*
]0,1] paramètre de quantification de l'importance de la distance inter-nœuds dans la formule.
Les nœuds sont ensuite renvoyés à l'utilisateur par ordre décroissant de *Pn*

Annexe

Annexe 4 : Exemple de calcul SVD

1.1 Exemple de décomposition en valeurs singulières d'une matrice :

La décomposition en valeurs singulière (Sigular value decomposition « SVD » repose sur un théorème d'algèbre linéaire qui dit qu'une matrice rectangulaire A peut être exprimée comme le produit de 3 matrices : une matrice orthogonale U, Une matrice diagonale S et la transposé d'une matrice orthogonale Où :

$$A_{mm} = U_{mm} S_{mn} V_{nn}^{t}$$

- Les colonnes de U sont vecteurs propres orthogonaux de AA^t .
- les colonnes de V sont les vecteurs propres orthogonaux de A^tA.
- S est une matrice diagonale qui contient les racines carrée des valeurs propres de U et V en ordre décroissant.

Exemple :

Soit une matrice A :

$$A = \begin{bmatrix} 3 & 1 & 1 \\ -1 & 3 & 1 \end{bmatrix}$$

Pour trouver U, on calcule d'abord AA^T :

$$AA^T = \begin{bmatrix} 3 & 1 & 1 \\ -1 & 3 & 1 \end{bmatrix} \begin{bmatrix} 3 & -1 \\ 1 & 3 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 11 & 1 \\ 1 & 11 \end{bmatrix}$$

On cherche maintenant les valeurs propres et les vecteurs propres correspondants de AA^T.
L'équation aux valeurs propres s'écrit :

$$\begin{bmatrix} 11 & 1 \\ 1 & 11 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \lambda \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

On peut écrire ces équations :

$$(11 - \lambda)x_1 + x_2 = 0$$

$$x_1 + (11 - \lambda)x_2 = 0$$

On cherche les valeurs propres en égalant le déterminant de la matrice des coefficients à 0 :

$$\begin{vmatrix} 11 - \lambda & 1 \\ 1 & 11 - \lambda \end{vmatrix} = 0$$

Qui s'écrit :

$$(11 - \lambda)(11 - \lambda) - 1 * 1 = 0$$

Qu'on peut développer comme suit :

$$\lambda^2 - 22\lambda + 120 = 0$$

Annexe

$$\text{Et } (\lambda - 10)(\lambda - 12) = 0$$

Donc λ=10 et λ=12 sont les valeurs propres de A

En remplaçant les valeurs propres dans les équations de départ, on peut trouver les vecteurs propres associés. Pour la valeur propre λ=10 on peut écrire :

$$(11 - 10)x_1 + x_2 = 0$$

Donc $x_1 = -x_2$

Qui est vraie pour un grand nombre de valeurs. On choisit $x_1 = 1$ et $x_2 = -1$

On a alors : λ_1=10 et $v_1 = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$

Pour la valeur propre λ_2=12 on a :

$$(11 - 12)x_1 + x_2 = 0$$

$$x_1 = x_2$$

Et donc en prenant $x_1 = 1$, on a $x_2 = 1$ et

$$\lambda_2\text{=12 et } v_2 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

Les vecteurs v_1 et v_2 deviennent les colonnes d'une matrice ordonnée en fonction des valeurs propres décroissantes, on a donc :

$$\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

Il faut finalement convertir cette matrice en matrice orthogonale. On accomplit cette opération par une procédure d'ortho normalisation de Gam-Schmidt.

Cette approche consiste à normaliser le premier vecteur et de normaliser ensuite les autres en termes d'eux-mêmes auxquels on soustrait une multiplication avec les vecteurs déjà normalisés. Plus formellement, soient un ensemble de vecteurs orthogonaux $u_1, u_2, \dots, u_{\beta-1}$, alors on peut écrire:

$$w_\beta = v_\beta - \sum_{i=1}^{\beta-1} u_i \,.\, v_\beta * u_i$$

Et $u_\beta = \frac{w_\beta}{\|w_\beta\|}$

On a :

$$v_1 = \frac{v_1}{\|v_1\|} = \frac{[1 \quad 1]^T}{\sqrt{1^2 + 1^2}} = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix}^T$$

Et

$$w_2 = v_2 - u_1 v_2 * u_1 = [1 \quad -1]^T - \left[\frac{1}{\sqrt{2}} \quad \frac{1}{\sqrt{2}}\right][1 \quad -1]^T * \left[\frac{1}{\sqrt{2}} \quad \frac{1}{\sqrt{2}}\right]$$
$$= [1 \quad -1]^T - 0 * \left[\frac{1}{\sqrt{2}} \quad \frac{1}{\sqrt{2}}\right] = [1 \quad -1]^T$$

$$u_2 = \frac{w_2}{\|w_2\|} = \left[\frac{1}{\sqrt{2}} \quad \frac{-1}{\sqrt{2}}\right]^T$$

On a donc pour U :

$$U = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} \end{bmatrix}$$

Le calcul de V est similaire et se fait sur A^TA :

$$A^TA = \begin{bmatrix} 3 & -1 \\ 1 & 3 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 3 & 1 & 1 \\ -1 & 3 & 1 \end{bmatrix} = \begin{bmatrix} 10 & 0 & 2 \\ 0 & 10 & 4 \\ 2 & 4 & 2 \end{bmatrix}$$

Les vecteurs propres de A^TA se calculent avec :

$$10x_1 + 2x_3 = \lambda x_1$$

$$10x_2 + 4x_3 = \lambda x_2$$

$$2x_1 + 4x_2 + 2x_3 = \lambda x_1$$

Donc :

$$(10 - \lambda)x_1 + 0x_2 + 2x_3 = 0$$

$$0x_1 + (10 - \lambda)x_2 + 4x_3 = 0$$

$$2x_1 + 4x_1 + (2 - \lambda)x_3$$

Si on fixe le déterminant à 0 on a :

$$\lambda(\lambda - 10)(\lambda - 12) = 0$$

Et $\lambda_3 = 0, \lambda_2 = 10 \; et \; \lambda_1 = 12$ sont les valeurs propres de A^TA

Pour $\lambda_1 = 12$:

$$(10 - 12)x_1 + 2x_3 = 0$$

$$-2x_1 + 2x_3 = 0$$

En choisissant $x_1 = 1$ et $x_3 = 1$ on trouve $x_2 = 2$

Et donc : $\lambda_1 = 12 \;\; v_1 = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix}$

On procède de même pour λ_2 et λ_3 et on trouve :

$\lambda_2 = 10 \;\; v_2 = \begin{bmatrix} 2 \\ -1 \\ 0 \end{bmatrix}$ et $\lambda_3 = 0 \;\; v_3 = \begin{bmatrix} 1 \\ 2 \\ -5 \end{bmatrix}$

On utilise les v_i comme les vecteurs colonnes d'une matrice (ordonnés par λ_i décroissant) :

$$\begin{bmatrix} 1 & 2 & 1 \\ 2 & -1 & 2 \\ 1 & 0 & -5 \end{bmatrix}$$

On normalise par Gam-Schmidt :

$$u_1 = \frac{v_1}{\|v_1\|} = \begin{bmatrix} \frac{1}{\sqrt{6}} & \frac{2}{\sqrt{6}} & \frac{1}{\sqrt{6}} \end{bmatrix}^T$$

$$w_2 = v_2 - u_1 v_2 * u_1 = \begin{bmatrix} \frac{2}{\sqrt{5}} & \frac{-1}{\sqrt{5}} & 0 \end{bmatrix}^T,$$

$$w_2 = v_3 - u_1 v_3 * u_1 - u_2 v_3 * u_2 = [1 \quad 2 \quad 5]^T$$

$$u_3 = \begin{bmatrix} \frac{1}{\sqrt{30}} & \frac{2}{\sqrt{30}} & \frac{-5}{\sqrt{30}} \end{bmatrix}$$

Et donc :

$$V = \begin{bmatrix} \frac{1}{\sqrt{6}} & \frac{2}{\sqrt{5}} & \frac{1}{\sqrt{30}} \\ \frac{2}{\sqrt{6}} & \frac{-1}{\sqrt{5}} & \frac{2}{\sqrt{30}} \\ \frac{1}{\sqrt{6}} & 0 & \frac{-5}{\sqrt{30}} \end{bmatrix}$$

Et :

$$V^T = \begin{bmatrix} \frac{1}{\sqrt{6}} & \frac{2}{\sqrt{6}} & \frac{1}{\sqrt{6}} \\ \frac{2}{\sqrt{5}} & \frac{-1}{\sqrt{5}} & 0 \\ \frac{1}{\sqrt{30}} & \frac{2}{\sqrt{30}} & \frac{-5}{\sqrt{30}} \end{bmatrix}$$

Annexe

Pour la matrice S, on prend la racine carré des valeurs propres non nulles et on les place sur la diagonale.

$$S = \begin{bmatrix} \sqrt{12} & 0 & 0 \\ 0 & \sqrt{12} & 0 \end{bmatrix}$$

On peut vérifier que :

$$A = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} \sqrt{12} & 0 & 0 \\ 0 & \sqrt{10} & 0 \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{6}} & \frac{2}{\sqrt{6}} & \frac{1}{\sqrt{6}} \\ \frac{2}{\sqrt{5}} & \frac{-1}{\sqrt{5}} & 0 \\ \frac{1}{\sqrt{30}} & \frac{2}{\sqrt{30}} & \frac{-5}{\sqrt{30}} \end{bmatrix} = \begin{bmatrix} 3 & 1 & 1 \\ -1 & 3 & 1 \end{bmatrix}$$

Bibliographie

Bibliographie

Bibliographie :

[Azzoug , 2013] : W. Azzoug:Contribution à la definition d'une approche d'indexation sémantique de documents textuels. Mémoire de Magister, Université Boumerdes, 2013

[Baziz M, 2005] M. Baziz : Indexation Conceptuelle Guidée par Ontologie pour la Recherche d'Information .Thèse de Doctorat de l'université Paul Sabatier, 2005.

[Besançon, 2002] R. Besançon. Intégration de connaissances syntaxiques et sémantiques dans les représentations vectorielles de textes. Application au calcul de similarités sémantiques dans le cadre du modèle DSIR. PhD thesis, Ecole polytechnique Fédérale de Lausanne, 2002.

[Boubekeur, 2008] F. Boubekeur-Amirouche : Contribution à la définition de modèles de recherche d'information flexibles basés sur les CP-Nets, thèse en informatique, Université Toulouse III - Paul Sabatier.

[Boughanem.M et al., 2008] Boughanem Mohand et Jacques Savoy : RI état des lieux et perspectives

[Chiaramella Y, 2000] Y. Chiaramella: Information Retrieval and Structured Documents. In Proceedings of the Third European Summer-School on Lectures on Information Retrieval (ESSIR 2000)

[Fellag, 2006] S. Berchiche-Fellag: Recherche d'information dans les documents semi-structurés XML. Thèse de magister, UMMTO, 2006.

[Fuhr, 2001] N.Fuhr XIRQL: a query language for information retrieval in XML documents, University of Dormund, Germany 2001.

[Fuller et al. ,1993] M. Fuller, E. Mackie, R. Sacks-Davis, and R. Wilkinson. Structural answers for a large structured document collection, 1993.

[Gruber, 1993] T.R. Gruber, A translation approach to portable ontology specifications, Knowledge Acquisition, 1993.

[Hammache, 2013] Hammache A: Recherche d'Information: un modèle de langue combinant mots simple et mots composés. Thèse doctorat, UMMTO, 2013.

[Harris et al., 1989] Z. Harris, M. Gottfried, T. Ryckman, P. Mattick, A. Daladier, T.N. Harris, S. Harris: The form of Information in Science: Analysis of an immunology sublanguage. Dordrecht : Kluwer Adademic Publishers (1989)

[Harrathi et al., 2010] R. Harrathi et S. Calabretto. Une approche de recherche sémantique dans les documents semi-structurés.

[Hatano et al, 02] K. Hatano, H. Kinutani, M. Yoshikawa, and S. Uemura. Information Retrieval System for XML Documents. 2002.

Bibliographie

[Jiang et al., 1997] Jiang J. J and Conrath D. W.:Semantic similarity based on corpus statistics and lexical taxonomy. In International Conference Research on Computational Linguistics (ROCLING X) (1997).

[Krovetz, 1997] Krovetz R. Homonymy and polysemy in information retrieval, 1997.

[Kim et al., 2005] Kim M. S. and. Kong Y.-H: Ontology-DTD Matching Algorithm for Efficient XML Query, in FSKD (2), ser. Lecture Notes in Computer Science, L. Wang and Y. Jin, Eds.(2005)

[Lalmas et al., 2005] Lalmas M. and Piwowarski B. INEX 2005 Relevance Assessment Guide, 2005. http://inex.is.informatik.uniduisburg.de/2005/internal/pdf/Relevance Assessment2005.pdf.

[Maisonnasse, 2008] Maisonnasse L: Les supports de vocabulaires pour les systèmes de recherche d'information orientés précision : application aux graphes pour la recherche d'information médicale. Thèse, Université Joseph Fourier – Grenoble I (2008)

[Ogden et al, 1923] Ogden, C. K., Richards, I. A. 1923. "The Meaning of Meaning." 1923.

[Philippe, 2007] Philippe,M: Recherche d'information "Fondements", 2007.

[Resnik P.1995] P. Resnik: Using information content to evaluate semantic similarity. In: Proc. 14th Int. Joint Conf. Artificial Intelligence, Montreal (1995).

[Resnik P.1999] P. Resnik: Semantic similarity in a taxonomy: An information-based measure and its application to problems of ambiguity in natural language, Journal of Artificial Intelligence Research (1999).

[Robertson, 1977] S. Robertson: The probability ranking principale in IR. Journal of Documentation, 1977.

[Sadi, 2012] S. Sadi: Prise en compte des liens dans la recherche d'information dans les documents XML. Mémoire Master, UMMTO 2012.

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

[Schenkel et al., 2003] G. Weikum, M. Theobald and R. Schenkel: Exploiting structure, annotation and ontological knowledge for automatic classification of xml data, In WebDB, San Diego, CA. 2003.

[Schenkel et al., 2005] R. Schenkel, A. Theobald , and Weikum G, : Semantic similarity search on semistructured data with the XXL search engine, Information Retrieval (2005)

Bibliographie

[Schlieder et al., 2002] T. Schlieder and H. Meuss: Querying and Ranking XML Documents. Journal of American Society for Information Science and Tecnology (JASIST), Special Topic Issue on XML and Information Retrieval 2002.

[Slimani et al. 2006] T. Slimani, B. Yaghlane , and K. Mellouli: A new similarity measure based on edge counting. In Proceedings of world academy of science, engineering and technology (December 2006).

[**Taha et al., 2008**] Taha K. and Elmasri R: CXLEngine: A Comprehensive XML Loosely Structured Search Engine, In Proceedings of the EDBT workshop, Nantes, France 2008.

[Tversky, 1977] A. Tversky. Features of similarity. *Psychological Review*, 1977.

[Veillard, 2000] D. Veillard. Les technologies associées à xml. 2000.

[Wu-Palmer, 1994] Z. Wu and M. Palmer: Verb semantics and lexical selection. In 32nd. Annual Meeting of the Association for Computational Linguistics (1994).

[Xavier, 2006] T. Xavier : La recherche d'information dans les documents XML. Rapport de recherche 2006.

[Zargayouna, 2005] H. Zargayouna : Indexation sémantique des documents XML. Thèse de doctorat, université Paris XI Orsay, 2005.

[Zargayouna et al., 2004] H. Zargayouna, S. Salotti: Mesure de similarité dans une ontologie pour l'indexation sémantique de documents XML. Actes de la conférence IC'2004 (2004)