REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université Mouloud MAMMERI de Tizi-Ouzou
Faculté de Génie Electrique et d'Informatique
Département d'Informatique

Mémoire de Fin d'étude
En vue de l'obtention du diplôme de Master en informatique

Option : Systèmes Informatiques

Thème :

Reformulation de la requête en recherche d'information en intégrant le profil utilisateur

Proposé & dirigé par :

Mme ACHMOUKH.F

Réalisé par :

Melle AOUDJ Leila

Promotion 2011 2012

REMERCIEMENTS

C'est à « ALLAH », que j'adresse toute ma gratitude en premier lieu.

Mes remerciements, les plus vifs, ma profonde gratitude et mes respects s'adressent à ma promotrice Madame Farida ACHEMOUKH pour avoir accepté de m'encadrer, pour les conseils et orientations tant précieux qu'elle m'a prodiguée durant ce mémoire.

Mes remerciements vont également à l'honorable jury qui a consenti à juger mon travail.

Je remercie aussi l'ensemble de la famille enseignante du département informatique pour m'avoir formé durant toutes ces années et tous ceux qui ont contribué de près ou de loin pour la réalisation de ce travail modeste.

Dédicaces

Je dédie ce modeste travail à:

Mes très chers parents

Ma grand-mère

Ma grande Famille

Mes amis, en particulier le groupe G7

Leila

Table des matières

Chapitre II: Reformulation de la requête

Chapitre III: Conception

Chapitre IV: Implémentation et tests

Liste des figures

Introduction

Introduction

Une quantité toujours croissante d'information électronique (Internet, CD ROM…) est disponible et mise à la disposition du grand public. Le domaine de la recherche d'information (RI) aspire à fournir des méthodes rapides et efficaces de représentation, de gestion et de présentation de ces informations.

Dans ce contexte, les systèmes de recherche d'information (SRI) sont conçus pour chercher et restituer des documents pertinents à des besoins utilisateurs exprimés au moyen d'une requête.

Cependant, il est difficile de choisir les termes adéquats à l'expression des besoins. La complexité de verbaliser un besoin en information peut augmenter lorsque le besoin est vague, lorsque la collection d'informations est peu familière à l'utilisateur ou lorsqu'il est inexpérimenté avec les SRIs. Par contre, il est plus facile pour un utilisateur de savoir si les documents restitués répondent à ses besoins c'est-à-dire qu'il peut évaluer la pertinence des documents.

Les techniques de reformulation de requête tiennent compte de ce dernier point. En effet, ces techniques tentent de construire automatiquement une représentation de requêtes en se basant sur les documents jugés pertinents par l'utilisateur. La reformulation de requêtes est un cycle d'activité : le système restitue, suite à une requête, des documents que l'utilisateur juge. Cette information (jugement) est utilisée par le système pour produire une nouvelle requête et restituer d'autres documents. Cette activité peut être itérative.

La reformulation de requêtes a fait ses preuves pour améliorer l'efficacité des SRIs pour certains types de recherche.

Notre but dans le cadre de ce mémoire est d'intégrer le profil utilisateur dans la phase de reformulation de requêtes pour le but d'enrichir la requête initiale de l'utilisateur, ensuite de réécrire cette dernière en basant sur les résultats de la première recherche.s

Nous avons structuré ce mémoire de la manière suivante : nous commencerons dans le premier chapitre par présenter l'émergence de la recherche d'information (RI) classique à la RI adaptative.

Le deuxième chapitre est consacré aux méthodes de reformulation de requête dans les différents modèles proposées.

Le troisième chapitre présente une conception détaillé de notre approche de reformulation que nous avons proposée.

Les résultats et expérimentations de notre approche dans la plateforme Terrier seront présentés dans le quatrième chapitre.

Nous concluons notre travail et présentons quelques perspectives à la fin du mémoire.

Chapitre I

Emergence de la recherche classique à la recherche adaptative

Avec l'avènement du web, l'expansion de l'informatique à tous les domaines de la vie courante, a eu pour conséquence directe, l'accessibilité par un large public d'utilisateurs, autre que des documentalistes spécialisés, à des masses d'information volumineuses et hétérogènes. Les efforts continus des chercheurs en RI ont permis jusqu'à présent d'améliorer sans cesse les performances et la qualité des services d'accès à l'information en passant par la RI classique à la RI adaptative puis à la RI personnalisée.

Dans ce chapitre, nous décrivons cette évolution de la RI en commençant par la RI classique où nous présentons ces notions de base ainsi que les différents modèles de recherche d'information classique et les méthodes d'évaluation des SRI. Puis nous décrivons l'orientation vers la RI adaptative.

Introduction

Le concept Recherche d'Information RI (Information Retrieval IR) fut utilisé pour la première fois par Calvin N Mooers en 1948 dans son mémoire de maîtrise **[10]**. Les techniques initiales, comme l'abstraction, l'indexation et l'utilisation des catégories de classification ont marqué la naissance de la Recherche d'Information comme discipline de recherche.

Les systèmes de recherche d'information (SRI), servent d'interface entre une collection contenant des quantités considérables de documents et des utilisateurs cherchant des informations susceptibles de se trouver dans cette collection, en utilisant des requêtes. Ces systèmes intègrent un ensemble de techniques pouvant être résumées en quatre fonctions, qui sont :

- Stockage des informations
- Organisation de ces informations (processus d'indexation)
- Recherche d'informations : en réponse à des requêtes utilisateurs
- Restitution des informations pertinentes pour ces requêtes.

I.1 Recherche d'information (RI) classique

I.1.1 Notions de base de RI

"The user expresses his information need in the form of a request for information. Information retrieval is concerned with retrieving those documents that are likely to be relevant to his information need as expressed by his request" [*2*].

L'objectif principal de la recherche d'information est de fournir des techniques et des outils pour sélectionner les informations pertinentes contenues dans une collection de

documents en réponse aux besoins en information d'un utilisateur représentés à l'aide d'une requête.

I.1.1.1 Collection de documents

IL s'agit du fond documentaire ou corpus. C'est l'ensemble des informations exploitables et accessibles. Généralement, quand elle traite un même domaine on l'appelle collection de domaine (par exemple : collection de la génomique) sinon, c'est-à-dire quand elle ne traite pas un même domaine, on l'appelle collection générique (par exemple : la collection des articles de presse).

I.1.2 Documents

Le document constitue l'information élémentaire d'une collection de documents, il peut être un texte, un morceau de texte, une page web, une image, une vidéo, etc. On parle aussi d'une toute unité qui peut constituer une réponse à un besoin informationnel de l'utilisateur.

I.1.3 Besoin en information

Le besoin en information est souvent assimilé au besoin de l'utilisateur. Trois types de besoins utilisateur ont été définis par Ingwersen **[10]**:

a) Besoin vérificatif : l'utilisateur cherche à vérifier le texte avec les données connues. Il recherche une donnée particulière, et sait souvent comment y accéder (par exemple chercher un document ayant une adresse connue sur le web).

b) Besoin thématique connu : l'utilisateur cherche à clarifier, à revoir ou à trouver de nouvelles informations dans un sujet et un domaine connus (il est possible que le besoin s'affine au cours de la recherche).

c) Besoin thématique inconnu: l'utilisateur cherche de nouveaux concepts ou de nouvelles relations hors des sujets ou domaines qui lui sont familiers.

I.1.4 Requête

C'est l'expression du besoin de l'utilisateur. La requête est l'interface entre le SRI et l'utilisateur. Une requête peut être soit un ensemble de mots clés, ou exprimée en langage naturel, booléen ou graphique.

I.1.5 Pertinence

La pertinence est la notion centrale en recherche d'information. C'est la correspondance entre un document et une requête, une mesure de l'informativité du document à la requête, un degré de relation entre le document et la requête.

La notion de pertinence est le critère primaire pour l'évaluation des systèmes de recherche d'informations. Il existe plusieurs « pertinences » possibles entre un document et un besoin, nous les citons dans ce qui suit :

- ***pertinence algorithmique*** **:** cette pertinence est traduite par une mesure algorithmique basée sur le calcul de la pertinence de l'information par rapport à la requête en utilisant des caractéristiques des requêtes d'une part et des documents d'autre part.

- ***pertinence thématique*** **:** cette pertinence est définie par le degré de couverture du document où se trouve le sujet de la requête.

- ***pertinence cognitive*** : c'est la pertinence liée au thème de la requête selon la perception ou les connaissances de l'utilisateur sur ce même thème. Elle est caractérisée par une dynamique qui permet d'améliorer la connaissance de l'utilisateur via l'information renvoyée au cours de sa recherche.

- ***pertinence situationnelle :*** cette pertinence est définie par l'utilisateur où l'utilisabilité de l'information jugée relativement à la tâche de recherche exprimée par le besoin en information selon la perception de l'utilisateur. C'est une pertinence potentiellement dépendante du contexte de recherche et vue comme une pertinence dynamique.

I.2 Processus de recherche d'information

L'objectif fondamental d'un processus de RI est de sélectionner les documents les plus proches du besoin en information de l'utilisateur décrit par une requête.

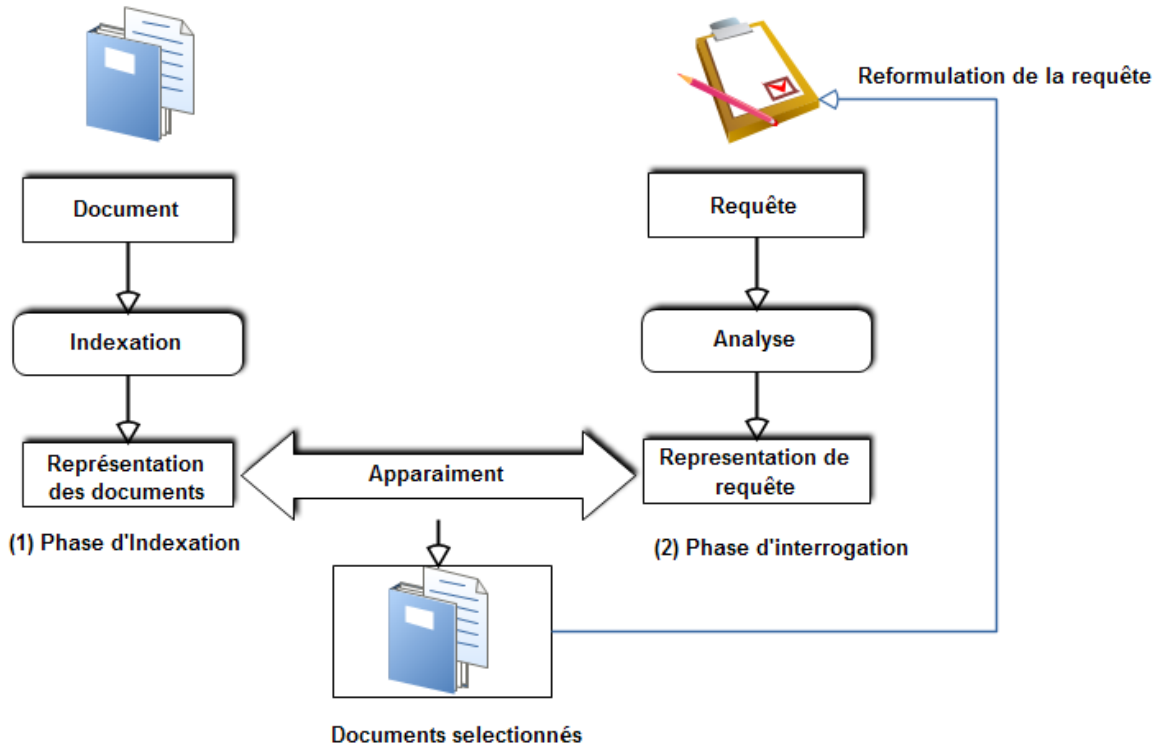


Figure I.1 : Processus en U de recherche d'information [11].

Le déroulement de ce processus induit deux principales phases : indexation et l'interrogation.

I.2.1 L'indexation

L'indexation recouvre un ensemble de techniques visant à transformer les documents (ou requêtes) en substituts ou descripteurs capables de représenter leur contenu [11]. Ces descripteurs forment le langage d'indexation représenté selon une structure souvent basée sur un ensemble de mots clés ou groupes de mots représentant le contenu textuel du document.
L'indexation peut être manuelle, automatique ou semi-automatique :

- *manuelle* : chaque document est analysé par un spécialiste du domaine correspondant ou par un documentaliste,
- *automatique* : chaque document est analysé à l'aide d'un processus entièrement automatisé,
- *semi-automatique* : le choix final reste au spécialiste du domaine correspondant ou documentaliste, qui intervient souvent pour établir des relations sémantiques entre mots-clés et choisir les termes significatifs.

L'indexation automatique est un processus composé des étapes suivantes :

I.2.1.1 Analyse lexicale

L'étape de l'analyse lexicale permet d'extraire l'ensemble des termes appartenant à un document. Cette extraction est effectuée en tenant compte des espaces de séparation entre mots, chiffres et ponctuations. Un terme peut être un mot simple (pomme) ou un mot composé (pomme de terre) mais en RI, on utilise souvent les mots simples.

I.2.1.2 Elimination des mots vides

Les mots vides sont des mots peu significatifs et porteurs de peu de sens, augmentant ainsi la taille de l'index et rendant la recherche plus lente. De ce fait, leur élimination est impérative.

Ces mots vides sont généralement des mots dits « grammaticaux » (comme les prépositions **à, de**, les articles **le, la, un, des**, les pronoms **ce, lui** ou encore les auxiliaires **être, avoir** ...), ou des mots très fréquents au sein d'une collection de texte donné.

On distingue deux techniques pour éliminer les mots vides :

- L'utilisation d'une liste de mots vides.
- L'élimination des mots dépassant un certain nombre d'occurrences dans la collection.

I.2.1.3 Lemmatisation

La lemmatisation consiste à remplacer les différentes formes d'un mot par leur lemme (racine grammaticale) car un mot peut avoir plusieurs formes dans un texte dont le sens est presque similaire. Exemple : porte et le verbe portera qui possèdent la même racine port.

I.2.1.4 Pondération des termes

Cette étape permet de mesurer l'importance d'un terme dans un document. Elle a pour but de trouver les meilleurs termes représentant le contenu d'un document.

Le pouvoir de discrimination des termes pour décrire le contenu des documents n'est pas identique pour tous les termes. Pour trouver les termes du document qui représentent le mieux son contenu sémantique, on a défini la fonction de pondération d'un terme dans un

document connue sous la forme de **Tf.Idf**, qui est reprise dans différentes versions par la majorité des SRI [11].

On y distingue :

- Tf (term frequency) : cette mesure est proportionnelle à la fréquence du terme dans le document. L'idée sous-jacente est que plus un terme est fréquent dans un document, plus il est important dans la description de ce document.

Le Tf est souvent exprimé selon l'une des déclinaisons suivantes :

1. Tf : utilisation brute,
2. $0.5 + 0.5\ \frac{Tf}{Max(Tf)}$

- *Idf* (Inverse of Document Frequency) : mesure l'importance d'un terme dans toute la collection. L'idée sous-jacente est que les termes qui apparaissent dans peu de documents de la collection sont plus représentatifs du contenu de ces documents que ceux qui apparaissent dans tous les documents de la collection. Cette mesure est exprimée selon l'une des déclinaisons suivantes :

1. $Idf = \log(\frac{N}{df})$.
2. $Idf = \log(\frac{N-df}{df})$.

Où *df* est la proportion de documents contenant le terme et *N* le nombre total de documents dans la collection.

La fonction de pondération de la forme *Tf.Idf* consiste à multiplier les deux mesures *Tf* et *Idf*. Une formule largement utilisée est la suivante [13]:

$$Tf.Idf = \left(0.5 + 0.5\frac{Tf}{Max(Tf)}\right) * \log\left(\frac{N}{df}\right)$$

Une normalisation de la mesure du Tf.Idf par rapport à la longueur des documents a été proposée comme suit [11]:

$$Tf.Idf = \frac{Tf + \log\left(\frac{N - df + 0.5}{df + 0.5}\right)}{2.\left(0.25 + 0.75.\frac{dl}{\Delta d}\right)}$$

Idf est la longueur du document en nombre de termes et Δd la longueur moyenne des documents de la collection.

En effet, lors des compagnes d'évaluation internationales, la mesure a eu des performances très limitées dans des corpus de taille très variable. Le problème posé est que les termes appartenant aux documents longs apparaissent très fréquemment et emportent le poids sur les termes appartenant à des documents moins longs. Les documents longs auront alors plus de chance d'être sélectionnés [13].

I.2.1.5 Création de l'index

L'index, défini comme étant la structure de stockage utilisée pour mémoriser les informations sélectionnées lors du processus d'indexation, cette structure permet de sélectionner pour n'importe quel terme tous les documents où il apparait. Plusieurs solutions de stockage ont été proposées parmi lesquelles : les fichiers inverses *(inverted files),* les fichiers de signatures *(signature files)* et les tableaux de suffixes *(suffix arrays).*

La solution la plus utilisée est celle des fichiers inverses. Ces fichiers sont composés de deux éléments principaux : Le dictionnaire et le fichier posting. Le vocabulaire consiste en la liste de tous les mots distincts extraits du texte. Pour chaque mot est assigné l'ensemble des positions dans lesquelles ce dernier apparait.

I.2.2 L'Interrogation

I.2.2.1 Appariement requête-document

Le processus d'appariement document-requête permet de mesurer la pertinence d'un document vis-à-vis d'une requête. De manière générale, à chaque réception d'une requête, le SRI calcule un score de pertinence (similarité vectorielle, probabiliste, etc.). Le processus d'appariement est étroitement lié au processus d'indexation et de pondération des termes, Il existe deux types d'appariement :

- *Appariement exact « exact match retrieval » :*

Le résultat est une liste de documents respectant exactement la requête spécifiée avec des critères précis. Les documents retournés ne sont pas triés.

- *Appariement approché « best match retrieval » :*

Le résultat est une liste de documents censés être pertinents pour la requête. Les documents retournés sont triés selon un ordre de mesure. Cet ordre reflète le degré de pertinence document/requête.

I.2.2.2 Reformulation de la requête

Parfois l'utilisateur est confronté à une situation difficile, il est incapable de trouver les mots précis pour exprimer son besoin en information. Alors, certains pour ne pas dire la majorité des documents qui lui sont retournés l'intéressent moins que d'autres.

Outre cela, certaines requêtes sont courtes, du coup, elles ne sont pas sémantiquement riches, pour que le SRI puisse retourner des documents pertinents. Afin de pallier ces problèmes, les chercheurs en RI se sont orientés vers l'intégration d'une étape supplémentaire dans le processus de recherche : *la reformulation ou expansion de requêtes*. Elle consiste à modifier la requête initiale de l'utilisateur par l'ajout de termes significatifs et/ou la ré-estimation de leur poids.

Si les termes rajoutés proviennent des documents de la collection, on parle de réinjection de pertinence (relevance feedback). En revanche, s'ils sont issus d'une ressource conceptuelle externe (ontologie, thésaurus ou dictionnaire), on parle dans ce cas de reformulation de requêtes directe [14]. Dans le deuxième chapitre on donne les différentes techniques et approches de reformulation de requêtes.

I.3 Taxonomie des modèles de RI

Les travaux de recherche dans le domaine de la RI ont conduit à la proposition de nombreux modèles. Un modèle de RI a pour rôle de fournir une formalisation du processus de recherche d'information et un cadre théorique pour la modélisation de la mesure de pertinence.

Cependant de nombreux modèles ont été proposés en RI, ils sont généralement regroupés autour des trois familles suivantes :

- les modèles ensemblistes qui considèrent le processus de recherche comme une succession d'opérations à effectuer sur des ensembles de mots contenus dans les documents,
- les modèles algébriques au sein desquels la pertinence d'un document par rapport à une requête est envisagée à partir de mesures de distance dans un espace vectoriel,

- les modèles probabilistes qui représentent la RI comme un processus incertain et imprécis où la notion de pertinence peut être vue comme une probabilité de pertinence.

I.3.1 Modèles ensemblistes

Nous nous intéressons ici uniquement au principal représentant des modèles inspirés de la logique booléenne et de la théorie des ensembles pour modéliser l'appariement entre une requête et les documents de la collection.

I.3.1.1 Le model booléen de base

Le premier modèle utilisé en recherche d'information fut le modèle booléen de base. Il est basé sur la théorie des ensembles et sur l'algèbre de Boole.

Dans le cadre de ce modèle, un document D est représenté par conjonction de terme t_i (simple ou composés) indépendant que l'on représente sous la forme D=t_1,t_2, …….t_t.

Une requête Q est représentée par une expression logique de termes connectés par les opérateurs logiques ET, OU, et NON.

A titre d'exemple, la requête (t_1 ET (t_2 OU t_8)), le système retournera les documents indexés avec t_2, t_3 et ceux possédant les termes t_2, t_8.

Pour qu'un document D corresponde à une requête Q, il faut que l'implication suivante soit valide : D=>Q. Dans le cas contraire, le document sera jugé comme sans intérêt et ne sera donc pas présenté à l'usager.

L'évaluation de la fonction de correspondance R entre un document D et les différentes formes de requêtes est donné par les équations suivantes : [14]

R (D, t_1 ET t_2) = R (D, t_1) . R (D, t_2)

R (D, t_1 OU t_2) = R (D, t_1) + R (D, t_2) – (R (D, t_1) . R (D, t_2))

R (D, t_1 NON t_2) = R (D, t_1) . (1 - R (D, t_2))

R (D, t) = 1 Si t ∈ D

R (D, t) = 0 si t∈ pas D

Parmi les avantages de ce modèle nous pouvons citer son efficacité, car le temps de réponse demeure assez faible même pour des collections volumineuses, de plus qu'il est simple à mettre en œuvre. En revanche, il présente beaucoup de limites :

- La formulation des requêtes est complexe, et non accessible à un grand public.
- Le non ordonnancement des documents sélectionnés.
- Tous les termes ont la même importance.
- L'efficacité de la recherche dépend d'une stricte égalité entre les termes de la requête et ceux du document.
- Pour une requête qu'est une longue conjonction, un document qui satisfait la majorité des termes est aussi mouvais qu'un document qui ne satisfait aucun terme.
- Pour une requête qu'est une longue disjonction, un document qui satisfait un terme est aussi bon qu'un document qui satisfait tous les termes.

I.3.1.2 Le modèle booléen étendu

Le modèle booléen étendu, appelé aussi modèle *P_Norm*[19] a été proposé en 1983 par Salton et M.J. Mac Gille [21], ils ont combiné entre les deux modèles : booléen et vectoriel pour permettre l'utilisation des opérateurs logiques tout en proposant une pertinence graduée (Utilisation de la pondération des termes dans un document).

Rappelons une des limites du modèle booléen. Si q = t1∧t2 est une requête alors si un document contient t1 et pas t2 ou si un document ne contient ni t1 ni t2, ces deux documents sont traités de la même manière ils ne sont pertinents ni l'un ni l'autre.

Avec une telle requête q = t1∧t2, on peut considérer que plus la distance est grande entre d (description dans le carré unitaire avec en abscisse t1 unitaire et en ordonnée t2 unitaire) et le point de coordonnées (1,1) correspondant donc à la représentation de t1∧t2 moins q est proche de d.

Pour une requête disjonctive binaire q = t1Vt2, le point à éviter est le point (0, 0), donc plus on est loin (plus d est loin) de (0, 0), plus la requête est satisfaite par d. (figure 1.3)

Figure I.2 : présentation du modèle booléen étendu.

La mesure de similarité requête-document est donnée comme suit :

Operateur OU

$$Sim(D_i, Q) = \left[\frac{\sum_{j=1}^{N} wq_j^p * wq_{ij}^p}{\sum_{i=1}^{N} wq_i^p}\right]^{1/p}$$

Operateur ET

$$Sim(D_i, Q) = 1 - \left[\frac{\sum_{j=1}^{N} wq_j^p * \left(1 - wq_{ij}^p\right)}{\sum_{i=1}^{N} wq_i^p}\right]^{1/p}$$

Avec : wq_j *le poids du terme* t_j dans la requête Q.

P est une constante : $0 \leq P \leq \infty$.

En variant le paramètre **P** entre 1 et∞, il est possible d'obtenir un système intermédiaire entre un modèle vectoriel (P=1) et un modèle booléen (P = ∞). En d'autres termes, le comportement du modèle P-Norm varie entre le modèle vectoriel et le modèle booléen comme illustré dans la figure suivante:

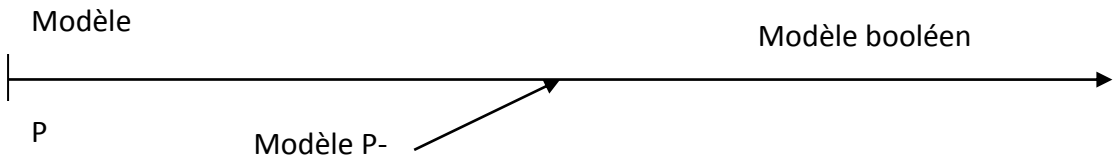


Figure I.3 : comportement du P-Norm

Le modèle P-Norm est intéressant non pas pour sa performance en pratique, mais aussi pour son cadre unificateur. Néanmoins, l'évaluation dans ce modèle est quelque peu complexe et l'opération de distributivité n'est pas retenue lors de l'évaluation.

3.2 Modèles algébriques

Les modèles algébriques rassemblent les modèles de RI qui proposent une représentation vectorielle des documents et requêtes. À partir de ces représentations, la mise en correspondance d'un document et d'une requête revient à appliquer un calcul algébrique de similarité entre vecteurs. Parmi les nombreux modèles qui s'appuient sur ce cadre théorique émergent les modèles cités ci-dessous.

I.3.2.1 Le modèle vectoriel

Le modèle vectoriel est introduit au début des années 70 par Gérard Salton et son équipe dans le système de recherche d'information SMART [22]. Basé sur une intention géométrique, le modèle vectoriel représente les documents sous la forme d'un vecteur de termes à **t** dimensions, dans laquelle **t** indique un nombre de termes d'indexation définit par le système. Un document $D_i = (w_{di\,1}, w_{di\,2}, \ldots\ldots, w_{di\,t}, \ldots\ldots w_{di\,t})$, dans lequel chaque valeur w_{dij} indique la pondération associée au termes d'indexation $\mathbf{t_j}$ dans le document D_i.

La requête Q est représentée suivant le même formalisme.

Soit $Q= (w_{q1}, w_{q2}, \ldots\ldots\ldots w_{qj,}\ldots\ldots\ldots\ldots w_{qt})$,

L'appariement s'effectue ainsi avec des fonctions de calcul de similarité entre vecteurs, tel qu'un calcul de distance.

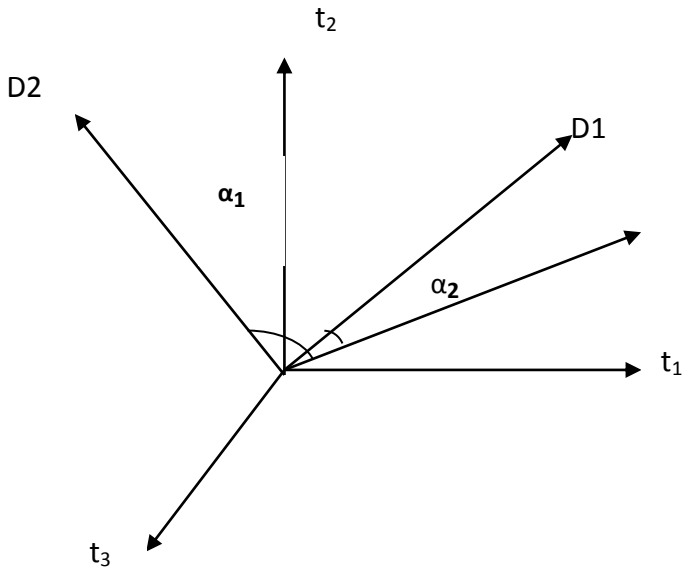
La mesure de similarité peut être exprimée par un produit scalaire :

$$Sim(D_i, Q) = \sum_{j=1}^{t} \left(wd_{ij} * wq_j\right)$$

La méthode la plus utilisée pour le calcul de similarité dans ce modèle est la mesure cosinus, qui est donnée par la formule suivante :

$$Sim(D_i, Q) = \frac{\sum_{j=1}^{t}\left(wd_{ij} * wq_j\right)}{\left[\sum_{i=1}^{t}(wqj)^2\right]^{1/2}\left[\sum_{j=1}^{t}(wqij)^2\right]^{1/2}}$$

Plus deux vecteurs sont similaires, plus l'angle formé est petit, et plus le cosinus de cet angle est grand. Dans l'exemple de la figure 2, le document D1 est plus similaire à la requête que le document D2.



FigureI.4 : Illustration de la représentation d'une requête (Q) et de deux documents (D1 et D2) avec le modèle vectoriel dans un espace à trois démenions

Le modèle vectoriel est le plus populaire en RI du fait qu'il dispose de plusieurs avantages à savoir :

- Le langage de requête est le plus simple (liste de mots clés).
- Les performances sont meilleures grâce à la pondération des termes.
- Le renvoi de documents à pertinence partielle, est possible.
- La fonction d'appariement permet de trier les documents.

Ceci malgré ses avantages, le modèle vectoriel est loin d'être parfait car, il présente des inconvénients :

- Le modèle considère que tous les termes sont indépendants.
- Le langage de requête est moins expressif.
- L'utilisateur voit moins pourquoi un document lui est renvoyé.

Pour résoudre le problème majeur de ce modèle, qui est l'indépendance des termes. On a proposé *le modèle vectoriel généralisé* ***GVSM (G****eneralised* ***V****ector* ***S****pace* ***M****odel)*.

I.3.2.2 Le modèle vectoriel généralisé GVSM

Dans ce modèle, les vecteurs définissant l'espace d'indexation ne sont pas orthogonaux. Ils sont représentés à partir des composants, les mini-termes, en tenant compte de la co-occurrence des termes dans les documents en supposant que la co-occurrence des deux termes dans un document implique leur inter-dépendance. Les mini-termes sont donc des expressions atomiques ou des concepts, représentés à partir de conjonctions de termes. Ainsi, le mini-terme m_k est défini par :

$$m_k = x_1 \wedge x_2 \wedge \ldots \wedge$$

Où x_i est t_i ou $>t_i$ tel que les ti sont des termes de la collection : Ces termes apparaissent une et une seule fois dans les *mk*.

Les ti peuvent également être exprimés en fonction des *mk* sous forme de disjonction.

$$t_i = m_{i1} \vee \ldots \vee m_{ir}$$

Où les m_i sont des mini-termes pour lesquels t_i n'est pas nié.

Les documents sont donc représentés dans un espace tenant compte des dépendances entre les termes. [21]

I.3.2.3 Le modèle LSI (*latent semantic indexing*)

La critique souvent énoncée pour le modèle vectoriel, est qu'il considère chaque terme d'un document indépendamment des autres. Le modèle LSI en est une réponse. Ce système étend le modèle vectoriel en utilisant des matrices et non plus des vecteurs pour représenter les documents [I.8]. Il propose, en s'appuyant sur une décomposition en valeurs singulières de la matrice pondérée classique d'occurrences des termes d'indexation dans les documents de la collection, de créer un espace vectoriel plus petit où les dimensions ne sont plus représentées par les termes mais par une combinaison linéaire de ces termes.

Ces combinaisons sont susceptibles de mieux faire ressortir les affinités *sémantiques* latentes entre les mots et, par conséquent, de mieux exprimer les concepts contenus dans les documents. [12]

Un élément de la matrice correspond au poids associé à un terme donné pour un document donné. Après décomposition en valeurs singulières, les axes factoriels correspondant aux plus grandes valeurs propres sont conservés. Cela montre qu'il s'agit des axes permettant un ajustement dans un espace de dimension réduite qui minimise la perte d'information. [22]Cette phase de réduction peut néanmoins s'avérer coûteuse en termes de calculs pour des matrices d'occurrences de grande dimension.

I.3.2.4 Le modèle connexionniste

Ce modèle repose sur la théorie des réseaux de neurones [19]. Un réseau de neurones est constitué d'un ensemble de couches : une couche d'entrée qui désigne la requête (chaque neurone correspond à un de ses termes), une couche qui représente l'ensemble des termes de la collection (chaque neurone équivaut à un terme) et une couche documents où un nœud représente un document de la collection (figure 1.4). Ces couches sont reliées entre elles, dont l'objectif est de représenter les différentes relations et associations qui existent entre les termes et les documents. À partir de cette représentation, le mécanisme d'appariement de la requête et des documents est relativement simple.

L'activation initiale des neurones représentant la requête se propage vers les nœuds « termes » de la collection, qui, à leur tour, envoient des signaux aux nœuds documents, à travers les différentes connexions pondérées du réseau. Les documents qui ont reçu le plus de signaux sont considérés comme les plus pertinents.

Ce modèle possède l'avantage de supporter des mécanismes d'apprentissage lui permettant de s'adapter aux points de vue des utilisateurs. Il est en particulier possible d'ajuster les pondérations associées aux différents éléments du modèle (termes d'indexation, documents) pour mieux répondre aux besoins des utilisateurs [22]. Il permet également de prendre en compte la dépendance entre les éléments d'informations. En revanche, la limite de ce modèle réside dans la complexité des calculs.

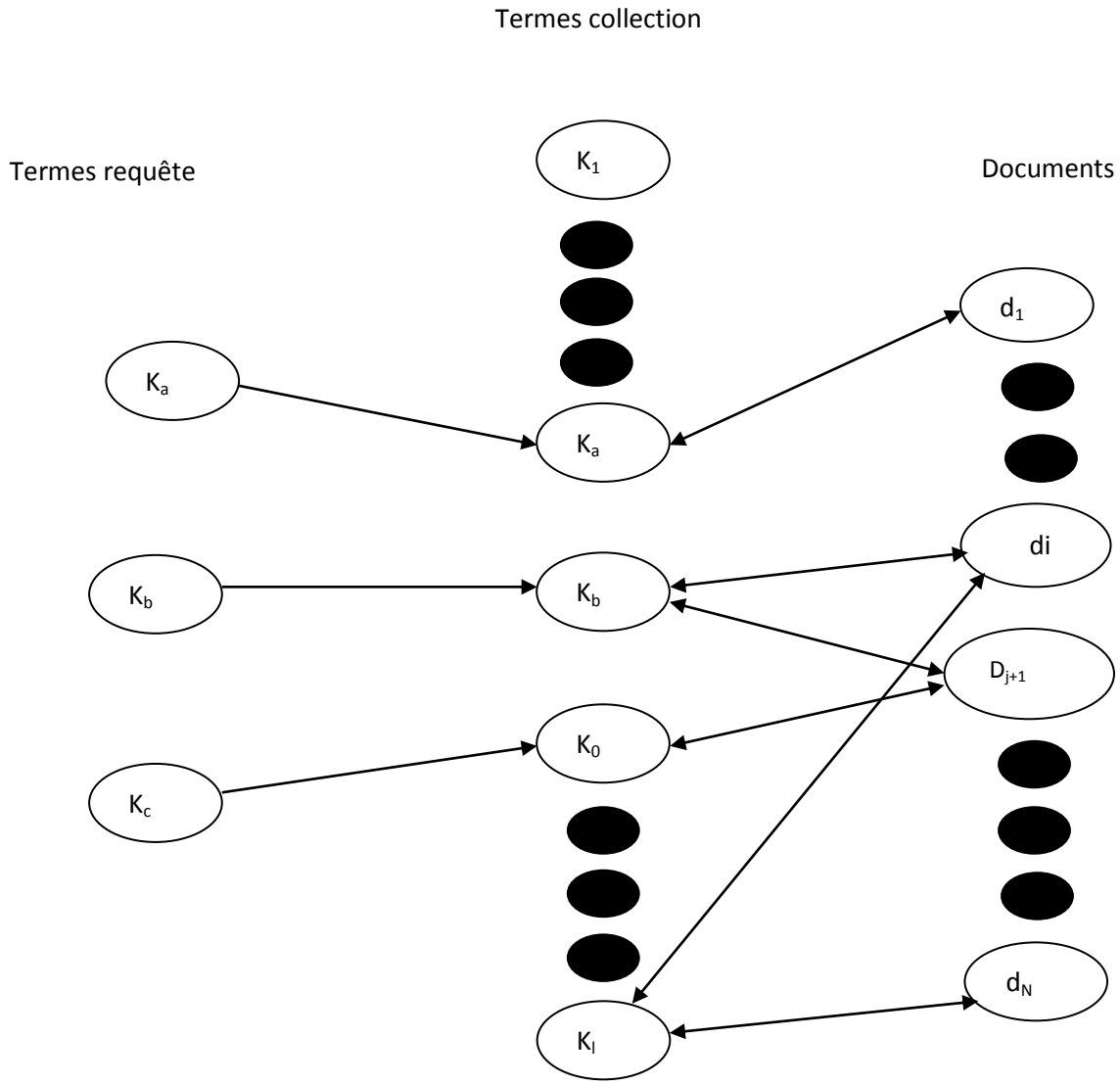


Figure I.4 Modèle de réseau de neurones pour la RI [22]

I.3.3 Les modèles probabilistes

La particularité des modèles probabilistes, qui constituent l'un des formalismes les plus utilisés aujourd'hui en RI, est de représenter la similarité d'un document vis-à-vis d'une requête par une probabilité de pertinence.

On peut distinguer deux grandes classes de modèles probabilistes : les modèles probabilistes de la pertinence, les modèles probabilistes d'inférences (la RI est un processus d'inférences incertaines).

I.3.3.1 Le modèle probabiliste de base

Le modèle probabiliste se base comme son nom l'indique sur l'utilisation de la théorie des probabilités et plus particulièrement sur le théorème de Bayes [23]. On cherche à savoir si un document est pertinent, c'est à dire qu'il appartient à l'ensemble des documents pertinents, noté R (pour relevant) ou non pertinent, noté NR (pour not-relevant).
On cherche donc à déterminer les deux probabilités suivantes :

- ✓ P(R|D), qui correspond à savoir pour un document D s'il appartient à l'ensemble R, et donc qu'il contient une information pertinente.
- ✓ P(NR|D), qui indique que le document D est non pertinent.

On travaille, en théorie des probabilités, sur des éléments observables qui sont ici l'absence ou la présence d'un terme dans le document. Cela veut dire que dans ce modèle, on ne tient pas compte de la pondération des termes, et donc que les termes sont soit présents et prennent une valeur de 1, soit absents et prennent une valeur de 0.

On cherche donc maintenant à déterminer la caractéristique de R et NR pour une requête donnée, c'est à dire que l'on cherche P(R|D) et P(NR|D).

En calculant, la valeur de ces deux probabilités, on peut donc classer les documents par ordre de pertinence à l'aide de la fonction O(D) définie comme le rapport de ces deux probabilités[23]

$$O(D) = \frac{P(R|D)}{P(NR|D)}$$

Plus la valeur de O(D) est importante, plus le document obtient un meilleur classement.
Ces deux probabilités ne pouvant pas être directement calculées, on utilise le théorème de Bayes pour reformuler les deux probabilités, soit [23]:

$$P(R|D) = \frac{P(D|R)P(R)}{P(D)}$$

$$P(NR|D) = \frac{P(D|NR)P(NR)}{P(D)}$$

Avec les significations suivantes :

- ✓ P(D|R) étant la probabilité au document D d'appartenir à l'ensemble des documents pertinents.
- ✓ P(D|NR) étant la probabilité au document D d'appartenir à l'ensemble des documents non pertinents.
- ✓ P(R), la probabilité d'un document choisi au hasard d'être pertinent.
- ✓ P(D), la probabilité que le document D soit tiré au hasard dans le corpus.

On obtient alors, en reportant ces valeurs dans O(D), la formule suivante :

$$O(D) = \frac{P(D|R)P(R)/P(D)}{P(D|NR)P(NR)/P(D)} = \frac{P(D|R)P(R)}{P(D|NR)P(NR)}$$

Or pour une même requête, P(R) et P(NR) sont des constantes. Comme on cherche uniquement à connaître l'ordre des documents pertinents, on peut redéfinir O(D) de la manière suivante :

$$O(D) = \frac{P(D|R)}{P(D|NR)}$$

On cherche donc à calculer la valeur de P(D|R) et P(D|NR). Or l'événement observé ici, est la « présence » ou « l'absence » d'un terme, c'est à dire que le terme t_i vaut x_i, avec x_i valant 1 ou 0, qui est représenté par $t_i = x_i$.

$$P(D|R) = P(t_1 = x_1, t_2 = x_2, \ldots, t_n = x_n|R)$$

Comme nous venons de le voir, cette approche est particulièrement intéressante dans le cadre du bouclage de pertinence.

En effet, la pertinence et la non-pertinence des documents étant connues, puisque données par l'utilisateur, la probabilité de pertinence de chaque terme de ces documents peut être calculée plus précisément et exploitée dans la reformulation des requêtes.

En outre, le modèle probabiliste présente quelques inconvénients :

- Difficulté de calcul de probabilités conditionnelles.
- Les jugements de pertinence étant propres à un utilisateur particulier, l'apprentissage effectué à partir de ces données n'est valable que pour cet utilisateur.
- Le modèle ne prend pas en compte les dépendances entre les termes d'indexation**.**

I.3.3.2 Modèle de réseau d'inférence bayésiens

Les réseaux d'inférence bayésiens ont été introduits dans la RI pour remédier aux défauts du modèle probabiliste.

Un réseau d'inférence est représenté par un graphe acyclique orienté où les nœuds représentent les événements (ou variables) et les arcs, les relations (différentielles) éventuellement pondérées entre ces nœuds. Appliqué à la RI, un réseau d'inférence (dont un exemple est donné en figure 1.6) se divise en deux sous-réseaux. Le sous-réseau documents se décompose en plusieurs niveaux hiérarchiques. Le premier niveau représente les documents de la collection. Chaque nœud *dj* correspond à la probabilité d'observer un document de la collection. Le second niveau correspond à la représentation des concepts, caractérisés le plus souvent par les termes des documents. Chaque nœud *rk* désigne la probabilité qu'un concept (un terme) ait été observé étant donné l'ensemble de ses nœuds parents (notée $P(rk\,|dj)$). Cette probabilité prend en compte la fréquence des termes dans les documents, dans la collection ou d'autres formes de pondération.

Le sous-réseau requête (reconstruit pour chaque nouvelle demande d'information) peut comporter également deux niveaux : le niveau « requête » (noté *q*) formé de nœuds toujours vrais représentant la proposition qu'un besoin d'information de l'utilisateur soit satisfait ; le niveau « concept » dans lequel les nœuds *ci*, vrais ou faux, représentent celle qu'un concept (terme) ait été observé ou non dans un document. Les deux sous-réseaux sont reliés par des arcs entre les nœuds concepts de la requête (*ci*) et les nœuds concepts des documents (*rk*). L'arc *rk* - *ci* porte une valeur égale à la croyance qu'un nœud *ci* ait été observé dans un document. Pour effectuer l'appariement entre une requête et un document, le réseau calcule, en activant un document à la fois et en propageant les valeurs de croyance de nœud en nœud, la probabilité que le document considéré réponde au besoin d'information de l'utilisateur. Les documents sont ensuite ordonnés selon cette probabilité [12].

Les principaux avantages de l'approche basée sur les réseaux d'inférence résident dans le fait qu'elle tient compte de la dépendance entre les termes, tout en permettant l'ordonnancement des documents en fonction de la probabilité trouvée. Son principal inconvénient réside dans le calcul des probabilités, nécessitant un temps exponentiel par rapport au nombre de termes de la requête.

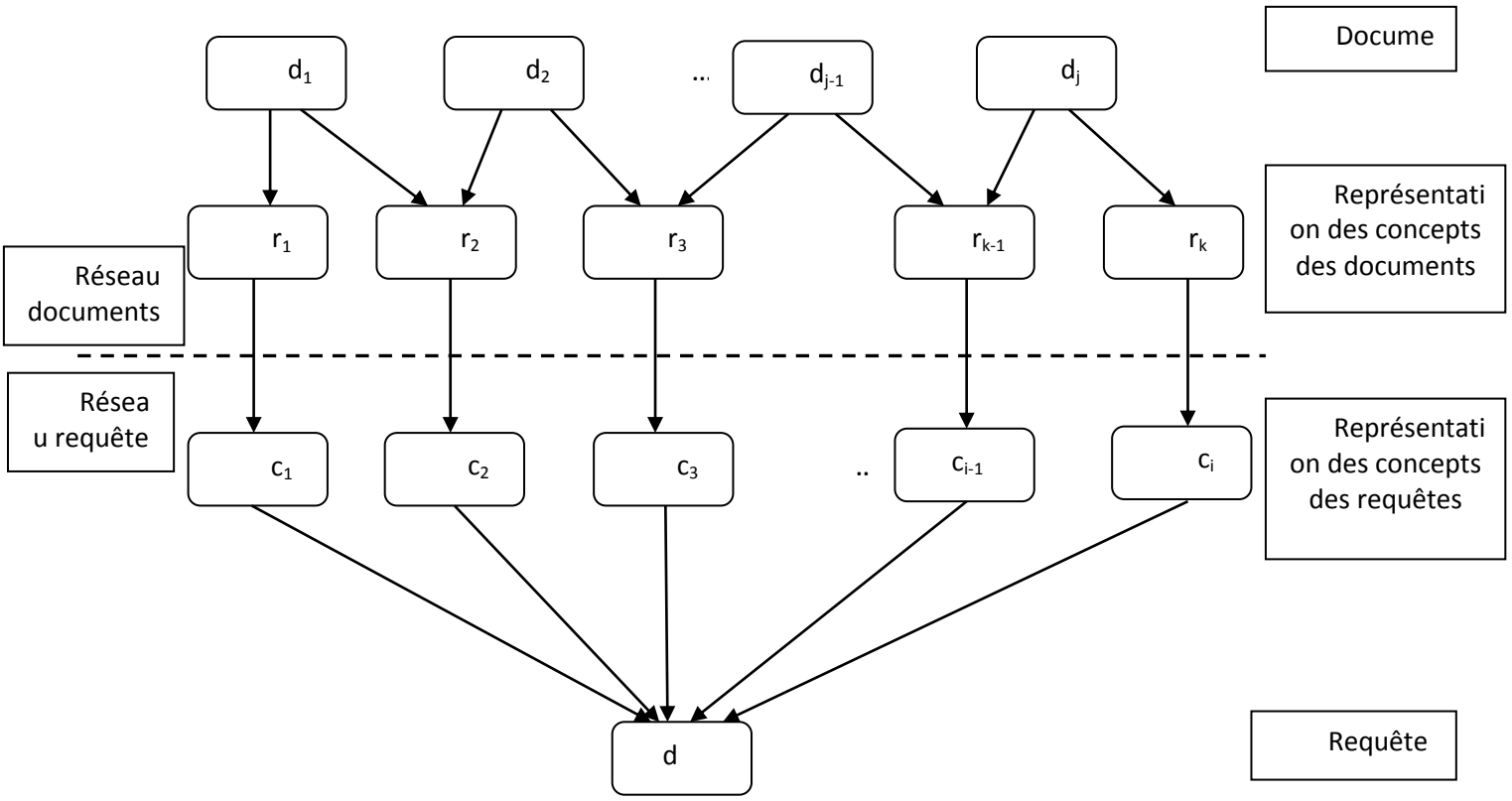


Figure I.5 : Exemple de réseau d'inférence pour la RI [12]

I.3.3.3 Le modèle de langue

Le modèle de langue est emprunté de la linguistique informatique. L'objectif d'un modèle de langue est de capter les régularités linguistiques d'une langue, en observant la distribution des mots, succession mots, dans la langue donnée.

Le modèle de langue désigne une fonction de probabilité qui assigne à chaque séquence de mots une probabilité. Les SRI utilisant les modèles de langage suivent une approche différente des autres modèles. En effet, dans la plupart des modèles, on cherche à comparer une représentation de la requête de l'utilisateur avec une représentation du document recherché pour évaluer la pertinence. Dans ce modèle, on part de l'observation que

l'utilisateur crée la requête à partir d'une représentation hypothétique qu'il se fait du document recherché. La requête est donc « générée » à partir des documents voulus.

La pertinence d'un document est ainsi estimée en calculant la probabilité que la requête utilisateur soit inférée part celui-ci. En conséquent on a [10] :

$$P(Q|D) = \prod_{k=1}^{n} P(Q_k|D)$$

Avec : Q_k sont les termes de la requête.

I.4 Evaluation des performances des SRI

Le domaine de la RI n'est pas une science exacte, car les approches font en sorte que la pertinence du système soit la plus proche possible de celle de l'utilisateur. Même si le temps de réponse et l'espace utilisé pour le stockage d'informations sont plus ou moins importants dans l'évaluation des SRI, la qualité des résultats renvoyés par un système reste le critère le plus important. Cette évaluation permet de comparer les SRI entre eux.

Les mesures d'évaluation doivent être effectuées pour les différents SRI sur les mêmes bases documentaires afin de rendre valable cette comparaison. Pour cela, plusieurs campagnes d'évaluation ont été créées. L'évaluation des SRI repose généralement sur trois éléments principaux :

- Une collection de documents de test ;
- Des requêtes de test ;
- Une liste des documents pertinents pour chaque requête.

I.4.1 Mesure d'évaluation des SRI

La démarche de validation en RI se base sur l'évaluation expérimentale des performances du modèle ou du système proposé. L'évaluation des performances d'un modèle de RI, permet de paramétrer le modèle, d'estimer l'impact de chacune de ses caractéristiques et de fournir des éléments de comparaison entre modèles.

Cette évaluation peut porter sur plusieurs critères : le temps de réponse, la pertinence, la qualité et la présentation des résultats, etc. Le critère le plus important est celui qui mesure la capacité du système à satisfaire le besoin en information de l'utilisateur, c'est à dire la pertinence. Deux facteurs permettent d'évaluer ce critère. Le premier est le taux de rappel, il mesure la capacité du système à sélectionner tous les documents pertinents. Le second est le

taux de précision, il mesure la capacité du système à rejeter tous les documents non pertinents. On calcule :

$$Taux - rappel = R_d = \frac{Nombre\ de\ documents\ pertinents\ restitués}{Nombre\ de\ documents\ pertinents}$$

$$Taux - précision = P_d = \frac{Nombre\ de\ documents\ pertinents\ restitués}{Nombre\ de\ documents\ restitués}$$

Ce type de mesure est effectué sur des collections de tests. Une collection de tests est composée d'un ensemble de documents, d'un ensemble de requêtes et d'un ensemble de jugements de pertinence.

I.4.2 Compagne d'évaluation TREC

Actuellement l'initiative la plus importante pour la construction de collections de tests est sans conteste TREC (Text REtrieval Conference, http://trec.nist.gov) [13]. TREC est plus qu'une collection de tests, c'est un programme d'évaluation des SRI, initié par le NIST (National Institute of Standards and Technology) aux USA.

TREC fournit une plate-forme comportant des collections de tests, des tâches spécifiques et des protocoles d'évaluation pour chaque tâche, pour l'évaluation et la comparaison d'expérimentations sur des collections volumineuses de textes. Les collections TREC représentent aujourd'hui un référentiel incontournable en RI [13].

I.5 Limitations de la RI classique

La RI classique a une vision orientée système, en ce sens où la recherche des informations pertinentes se base uniquement sur l'appariement des documents avec la requête soumise par l'utilisateur. Toutefois, cette vision de l'accès à l'information suppose que l'utilisateur est extérieur au système de recherche.

Néanmoins, dans la pratique la majorité des requêtes exprimées par les utilisateurs sont courtes et ambiguës (3 à 4 mots), ce qui donne des spécifications inachevées sur leur besoin en informations. En outre, cette liste de termes ne correspond pas forcément à ceux utilisés pour indexer les documents pertinents de la collection et manque souvent de termes significatifs pouvant exprimer effectivement le besoin en information de l'utilisateur. La

différence de vocabulaire entre les termes choisis par l'utilisateur pour formuler sa requête et les termes utilisés pour représenter les documents engendrent un défaut d'appariement. Ce défaut d'appariement est à l'origine d'une dégradation des performances de recherche.
L'avènement de la RI adaptative inscrit alors l'amélioration de l'efficacité du processus de recherche d'information.

I.6 La RI adaptative

Les performances d'un SRI ne dépendent pas uniquement de l'efficacité et la qualité des mécanismes d'indexation et d'appariement, mais de façon non négligeable de la capacité du SRI de prendre en considération les besoins de l'utilisateur pour mieux répondre à leurs attentes. De ce constat est apparu un nouvel axe de recherche, celui de la RI adaptative.

Les travaux de la RI adaptative se sont particulièrement axés sur l'amélioration de l'efficacité du processus de recherche notamment lors de la phase d'exécution de la requête. Les techniques développées ont eu pour but de désambiguïser le sens des mots de la requête utilisateur afin de mieux cerner le but de sa recherche. Les techniques développées en RI adaptative se focalisent principalement sur l'assistance à la formulation des requêtes des utilisateurs.

I.6.1 Reformulation de requêtes

Le but des techniques de reformulation de requêtes par réinjection de pertinence est de générer une nouvelle requête mieux adaptée au besoin en information de l'utilisateur. La reformulation de requêtes consiste à ajouter de nouveaux termes à une requête initiale ou alors repondérer les poids de ses termes dans le but de cibler la recherche vers les documents pertinents. La génération de la nouvelle requête s'effectue soit par reformulation automatique ou par reformulation interactive [4].

I.6.1.1 La reformulation automatique de requêtes

La reformulation automatique de requêtes est l'une des premières techniques ayant produit des améliorations notables dans ce domaine. L'idée de base est d'ajouter à la requête initiale des termes issus de ressources linguistiques existantes ou bien de ressources construites à partir des collections.

Il s'agit principalement de chercher des associations inter-termes extraites à partir des ontologies linguistiques ou à partir de thésaurus. Il existe aussi d'autres méthodes entièrement automatiques telles que le calcul des liens contextuels entre termes et la classification automatique de documents. Dans ce dernier cas, ces liens peuvent être construits à partir des documents retrouvés par le système, on parle alors de reformulation par contexte local, ou à partir de la collection entière de documents, on parle alors de reformulation par contexte global. Ces associations sont créées automatiquement et généralement basées sur la cooccurrence des termes dans les documents. Par ailleurs, cette approche ne prend en compte que les documents de l'ensemble de la collection qui ne sont pas forcément pertinents pour la requête. De ce fait, l'approche globale ne fournit seulement qu'une solution partielle aux problèmes d'ambiguïté des requêtes.

I.6.1.2. La reformulation interactive de requêtes

A la différence de la reformulation automatique, l'approche interactive (ou par réinjection de pertinence et/ou non-pertinence) exploite uniquement un sous-ensemble de documents sélectionnés parmi les premiers résultats obtenus de l'exécution de la requête initiale. Son principe fondamental est d'utiliser cette requête pour amorcer la recherche, puis modifier celle-ci à partir des jugements de pertinence et/ou de non pertinence de l'utilisateur, soit pour repondérer les termes de la requête initiale, soit pour y ajouter (resp. supprimer) d'autres termes contenus dans les documents pertinents (resp. non pertinents) [4]. La nouvelle requête ainsi obtenue à chaque itération de feedback, permet de corriger la direction de la recherche dans le sens des documents pertinents.

I.6.2 Le filtrage d'information

Le filtrage est un processus qui consiste à extraire les informations pertinentes et de qualité à partir d'une imposante masse d'informations. Une des approches pour adapter le processus de recherche à l'utilisateur est de filtrer l'information en fonction de son profil.

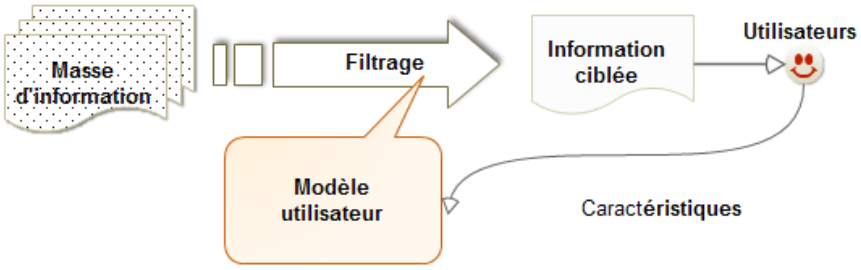


Figure I.6: Filtrage d'information [7].

I.6.2.1 Principe du filtrage

Le filtrage d'informations consiste à concevoir des mécanismes destinés à faire parvenir à l'utilisateur l'information qui l'intéresse directement. Un SRI a pour fonction « d'amener à l'utilisateur les documents qui vont lui permettre de satisfaire ses besoins en information » [7].

Un système de filtrage d'informations (SFI) peut être assimilé à un assistant personnel et doit être capable d'identifier les documents qui correspondent ou pas aux besoins en information des utilisateurs.

Un SFI extrait et achemine au sein d'un grand nombre d'informations, provenant de sources dynamiques, les seuls documents susceptibles de répondre aux besoins et intérêts de l'utilisateur après que celui-ci ait dressé un profil de requête, c'est-à-dire défini ses centres d'intérêts [21].

Une des premières approches à mettre en œuvre est la notion de filtrage d'information qui a été la DSI (Dissémination Sélective de l'Information) vers les années 60. Elle consiste à filtrer l'information produite par des scientifiques, dans le but de les maintenir continuellement informés des nouveautés relatives à leurs domaines de spécialisation [7].

Le processus de filtrage d'informations débute avec des individus ou groupes d'individus qui ont des intérêts relativement stables à long terme et décrits à travers des profils. La source d'informations provient des producteurs de textes (exemple : journaux, Internet, CD-ROM, etc.). Ces derniers doivent distribuer ces informations aux personnes

intéressées. Cette opération est réalisée en comparant les textes aux profils des différents individus.

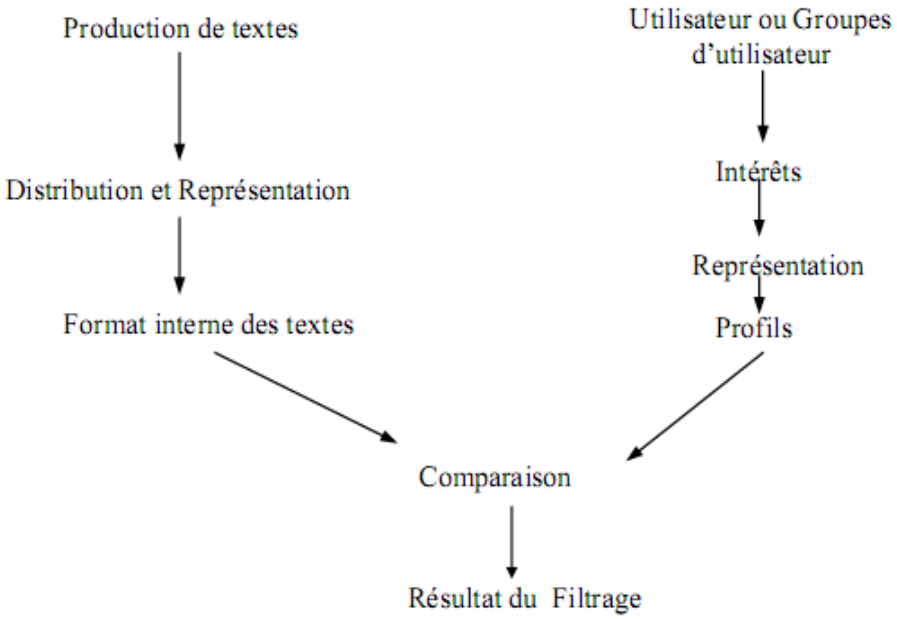


Figure I.7 : Processus du filtrage d'information [7].

I.6.2.2 Les modes de filtrage

On distingue principalement trois modes de filtrage : le filtrage collaboratif, le filtrage explicite et le filtrage implicite.

I.6.2.2.1 Le filtrage collaboratif

Le terme de filtrage collaboratif (collaborative filtering) décrit les techniques d'un groupe d'utilisateurs pour prédire la préférence inconnue d'un nouvel utilisateur; les recommandations pour le nouvel utilisateur sont basées sur ces prédictions [21].

Les utilisateurs d'un système d'informations participent activement à l'alimentation d'une base de données gérée par le filtre, contenant des informations sur eux-mêmes, et sur les documents qu'ils ont consultés. L'utilisateur donne son avis sur les documents lus et ces réactions peuvent être annotées et consultées par d'autres. Ainsi des relations document-document et document-utilisateur sont établites. Le filtre dispose donc d'informations variées sur un document et peut par exemple proposer à un utilisateur la liste des personnes travaillant sur le même sujet et lui sélectionner les documents qu'elles ont consultés. Si parmi ces

documents certains sont pertinents, le filtre pourra trouver l'ensemble des documents ayant les mêmes annotations [19].

I.6.2.2.2 Le filtrage explicite

Dans le cas du filtrage explicite, le contenu du profil est ciblé en fonction des informations données volontairement par l'utilisateur lors de son enregistrement sur le site. Par exemple, de nombreux sites demandent à l'utilisateur de préciser ses centres d'intérêts. Il recevra ensuite des offres qui lui correspondent. Il permet la spécification des centres d'intérêts d'un utilisateur par une liste de mots clés. Ils proposent également à l'utilisateur un menu où il peut choisir le contenu de la page en cochant les catégories jugées intéressantes parmi une liste donnée (actualités, horoscopes, programmes télé, météo, cinéma, etc.). Une fois les catégories intéressantes choisies, le système offre à l'utilisateur une page personnelle qui contient des informations dont les thèmes sont les catégories et des liens vers des données supplémentaires [22].

Une autre technique de filtrage explicite est l'utilisation du feedback de l'utilisateur [7]. En fonction des pages et documents jugés pertinents par l'utilisateur, le système va extraire une liste de termes qui représente le mieux la sémantique des documents. En fonction des données recueillies, le système va pouvoir proposer à l'utilisateur des pages ou des liens vers des pages qui concernent son profil.

I.6.2.2.3 Le filtrage implicite

Dans le cas du filtrage implicite, le contenu du profil est ciblé en fonction des données extraites automatiquement à partir du comportement de l'utilisateur. Par exemple: une page ouverte pendant un long moment peut signifier l'intérêt de l'utilisateur pour les informations qui y sont présentes. On prend ainsi en compte toutes sortes d'opérations, réalisées à l'aide de la souris, qui vont permettre d'en savoir un peu plus sur l'utilisateur : sélection du texte, passage sur un lien, clic sur un lien, vitesse de défilement de la page, etc.

La technique de l'enregistrement Web (Web recording) participe également au filtrage implicite. Elle consiste à mémoriser l'opération de navigation effectuée par un ou plusieurs clients, afin de la reproduire automatiquement pour un autre client. Puisque l'opération est connue, il devient possible de la reproduire à plus grande vitesse. On anticipe sur ce que va faire l'utilisateur pour lui faire gagner du temps [22].

Conclusion

Les travaux en RI adaptative ont certes apporté des solutions en particulier au défaut d'appariement requête-document qui ont conduit à l'amélioration des performances du processus de recherche d'information. Cependant une analyse des travaux dans ce domaine montre que ces performances dépendent de nombreux facteurs à priori non contrôlés par le processus de réécriture adaptative de la requête. Ces facteurs, principalement liés aux approches de reformulation de requête, qui sont ainsi problématiques, peuvent être catégorisés selon trois principales dimensions : l'utilisateur, l'information portée par la requête et/ou document et l'interaction entre l'utilisateur et le SRI.

Chapitre II

Reformulation de la requête

Nous avons présenté dans le chapitre précédent les notions de bases de la RI, dans ce chapitre, nous présenterons quelques techniques utilisées pour l'amélioration des performances des SRI et nous nous baseront sur la réinjection de pertinence ainsi que ses différentes applications.

Introduction

Les performances d'un SRI, mesurées en général par la double mesure rappel-précision, dépendent d'une part de l'efficacité du modèle de recherche mis en œuvre pour l'appariement entre requête documents, et d'autre part des requêtes formulées par l'utilisateur.

La requête initiale de l'utilisateur est souvent représentée par une liste de termes très réduite. Cette liste manque souvent de termes intéressants pouvant exprimer effectivement le besoin en information de l'utilisateur. Ceci a plusieurs raisons, la plus importante vient de la diversité du vocabulaire de la collection de documents. En effet, l'utilisateur ne connaît pas le vocabulaire utilisé dans les documents qu'il recherche. Pour pallier ces problèmes, les systèmes de recherche d'information proposent des techniques, appelées reformulation de le requête (expansion de la requête ou réinjection de pertinence), pour affiner et améliorer automatiquement la requête initiale de l'utilisateur.

La reformulation de requêtes, est un processus ayant pour objectif de générer une nouvelle requête plus adéquate que celle initialement formulée par l'utilisateur. Elle représente une forme de personnalisation à court terme.

Cette reformulation permet de coordonner le langage de recherche, utilisé par l'utilisateur dans sa requête et le langage d'indexation [15]. Par conséquent, elle limite le bruit et le silence, dus à un mauvais choix des termes d'indexation dans l'expression de la requête.

II.1 Techniques d'amélioration des SRI par reformulation de requêtes

La requête initiale seule est souvent insuffisante pour permettre la sélection de documents répondant aux besoins de l'utilisateur. De ce fait, plusieurs techniques ont été proposées pour améliorer les performances des SRI.

La figure ci-dessous, présente les principales techniques d'amélioration des SRI par reformulation de la requête initiale en y ajoutant de nouveaux termes. La reformulation peut se faire par expansion automatique de la requête, par combinaison de différentes présentations de la requête ou par réinjection de pertinence.

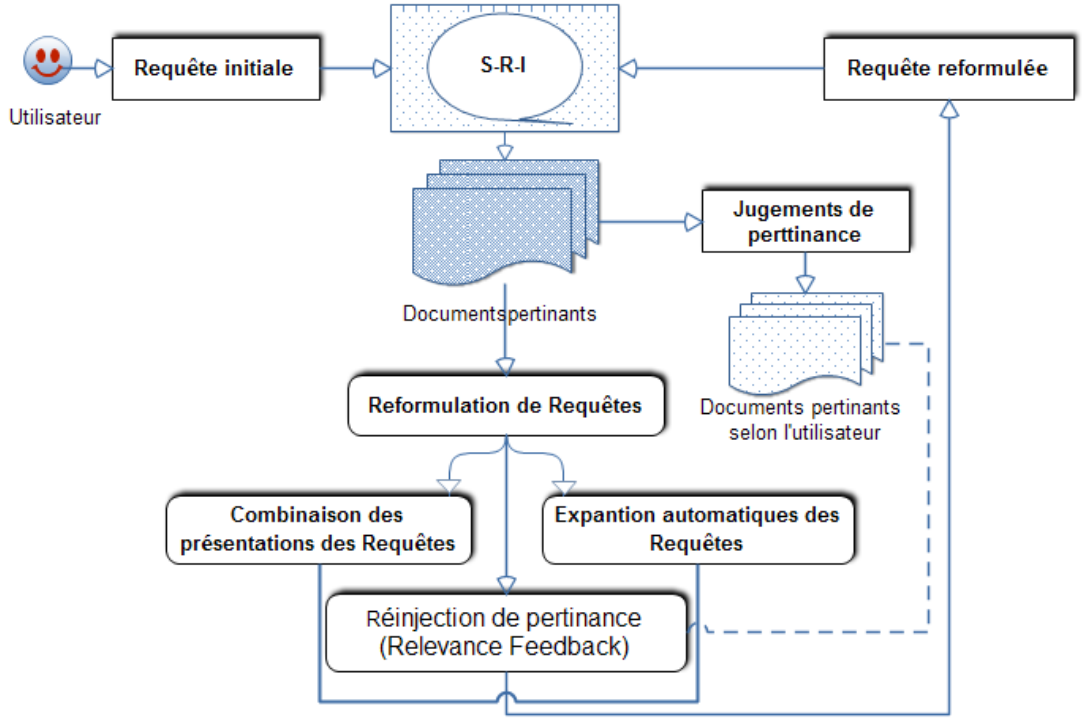


Figure II.1 : Processus d'amélioration des SRI par reformulation de requêtes [24].

II.1.1 Reformulation automatique des requêtes (directe)

La reformulation directe de requêtes consiste à ajouter d'autres termes à ceux choisis par l'utilisateur pour l'interrogation. Les nouveaux termes sont choisis automatiquement, sans l'intervention de l'utilisateur et doivent avoir des sens proches des termes donnés par l'utilisateur.

Cette reformulation consiste à rajouter à la requête initiale des termes issus de ressources linguistiques existantes ou bien de ressources construites à partir des collections. Plus précisément, au niveau des ressources linguistiques, le but est d'utiliser un vocabulaire contrôlé issu de ressources externes. Ce mécanisme assure la restitution des documents indexés par des variantes des termes composant la requête.

Les associations établies manuellement traduisent généralement des relations de synonymie et de hiérarchie. Les thésaurus construits manuellement sont un moyen efficace pour l'expansion de la requête.

En ce qui concerne la seconde catégorie de ressources (ressources construites à partir des collections), elles sont construites en s'appuyant sur une analyse statistique des collections. Il s'agit de chercher des associations de termes afin d'ajouter des termes voisins à la requête. Les associations créées automatiquement sont généralement basées sur la

cooccurrence des termes dans les documents. Les liens inter-termes renforcent la notion de pertinence des documents par rapport aux requêtes.

Le problème avec la reformulation automatique est l'estimation des « bons » termes qui peuvent conduire effectivement à une amélioration du processus de recherche car l'introduction des termes inappropriés peut entraîner un silence ou au contraire augmenter un bruit.

II.1.1.1 Expansion de requêtes avec les thésaurus

Le thésaurus est un outil classique en recherche d'informations : il consiste en la définition d'un certain nombre de termes du domaine, généralement appelés *concepts* et la représentation de *relations sémantiques.* Un thésaurus joue un rôle important dans un SRI. L'utilisation la plus courante est celle de l'expansion de requêtes, cela consiste à étendre la requête de l'utilisateur en ajoutant ou en remplaçant des termes par ceux du thésaurus pour permettre de faire correspondre les thèmes exprimés dans une requête et l'indexation des documents qui a été réalisée avec d'autres termes. En effet, le thésaurus définit les relations entre les différents termes de l'index et permet de sélectionner de nouveaux termes à ajouter à la requête initiale. Plus les termes ajoutés sont pertinents, plus la performance du SRI est élevée.

Le thesaurus regroupe plusieurs informations de type linguistique (équivalence, association, hiérarchie) et statistique (pondération des termes).

Il existe deux manières de construire un thésaurus : manuelle et automatique.

A. ***Construction manuelle de thésaurus***

Le principe fondamental des thésaurus construits manuellement est d'ajouter à la requête initiale les termes voisins définis dans le thesaurus. En effet, le système présente le thésaurus à l'utilisateur et lui permet de naviguer parmi les termes. La polysémie y est traduite par la possibilité d'associer à chaque mot, *n* catégories différentes représentant ses différents sens. Ce mode de construction est généralement adapté à des collections de petites tailles, à domaine spécifique [25].

Cependant, les thésaurus manuels sont peu utilisés par les SRI car leur construction et la maintenance des informations sémantiques qu'ils contiennent sont coûteuses en temps et nécessitent le recours aux experts des domaines considérés [25].

B. ***Construction automatique de thésaurus***

La construction automatique de thésaurus est typiquement basée sur la cooccurrence des termes. On prend l'exemple de thésaurus de similarité.

- ***Thésaurus de similarité***

Les méthodes d'expansion de la requête basées sur un thésaurus de similarité consistent à rajouter à la requête des termes issus d'un thésaurus de similarité. Ceci revient à calculer des valeurs de similarité entre les termes de la requête et ceux d'un thésaurus donné. Les k meilleurs termes de similarité la plus élevée sont rajoutés à la requête initiale.

Un thésaurus de similarité est une matrice de similarité terme-terme [6]. Pour le construire, au lieu de représenter un document par un vecteur de termes, on représente chaque terme t_i par un vecteur de documents dans un espace de vecteurs de documents, par exemple, $\vec{t_i} = (d_{i1}, \dots, d_{in})$. Avec, d_{ik} signifie le poids du document D_k par rapport au terme t_i et n est le nombre de documents dans la collection.

La formule de pondération utilisée par Qui et Frei [6] pour calculer le poids d_{ik} est la suivante:

$$d_{ik} = \frac{\left(0.5 + 0.5\frac{ff(d_k, t_i)}{\max ff(t_i)}\right) \times iif(d_k)}{\sqrt{\sum_{j=1}^{n}\left(\left(0.5 + 0.5\frac{ff(d_j, t_i)}{\max ff(t_i)}\right) \times iif(d_j)\right)^2}}$$

Avec :

$ff(d_k, t_i)$: la fréquence du terme t_i dans le document d_k,

$iif(d_k) = \log (m/|d_k|)$: la fréquence inverse des d_k ; avec m le nombre de termes dans la collection et $|d_k|$ est le nombre de termes dans le document d_k, maxf $f(t_i)$: la fréquence maximale du terme ti dans toute la collection.

Le thésaurus de similarité est construit en calculant la similarité entre toutes les paires de termes (t_i, t_j). La similarité entre deux termes est exprimée par le produit scalaire suivant :

$$Sim(t_i, t_j) = \vec{t_i}.\vec{t_j} = \sum_{k=1}^{n} d_{ik} \, . \, d_{jk}$$

Grâce à ce thésaurus, l'expansion d'une requête Q consiste à calculer une similarité, notée $Sim_{Qt}(Q, t_j)$, entre chaque terme du thésaurus et la requête Q, et non pas avec chaque terme de la requête Q.

La formule utilisée pour calculer cette similarité est la suivante :

$$Sim_{qt}(Q, t_j) = \sum_{t_i \in Q} q_i . Sim(t_i, t_j)$$

Où,

q_i est le poids du terme t_i dans la requête Q.

Ainsi, tous les termes de la matrice du thesaurus de similarité, associés aux termes de la requête, peuvent être ordonnés par ordre décroissant de leur valeur de similarité Sim_{qt}.

Les poids des termes sélectionnés sont donnés par la formule de pondération suivante :

$$wq_i = \frac{Sim_{qt}(Q, t_j)}{\sum_{t_i \in Q} q_i}$$

Le nombre de termes à rajouter à la requête est un paramètre important en recherche d'information (RI) [6].

Cette approche peut construire automatiquement les liens entre les termes en se basant sur la cooccurrence de ces termes. Ces liens sont mesurés en se basant sur la cooccurrence entre termes et sont construits, soit à partir des documents retrouvés par le système, on parle alors de reformulation par contexte local, soit à partir d'une collection de documents existante, on parle alors de reformulation par contexte global.

II.2 Combinaison des présentations des requêtes

Plusieurs approches de RI utilisent une seule représentation de requêtes comparée à plusieurs représentations de documents (algorithmes multiples de recherche) [15]. Il a été montré qu'une recherche plus efficace peut être atteinte en exploitant des représentations multiples de requêtes ou des algorithmes de recherche différents ou encore en utilisant différentes techniques de réinjection.

Une combinaison des représentations de requêtes peut augmenter le rappel d'une requête, tandis que la combinaison des algorithmes de recherche peut augmenter la précision. La base théorique de la combinaison des évidences (termes) a été présentée par Ingwersen[24]. Il a, en particulier, montré que des représentations multiples d'un même objet,

par exemple une requête, permettent une meilleure perception de l'objet qu'une seule bonne représentation.

Cependant, il est important que chacune des sources d'évidences utilisées fournisse non seulement un point de vue différent sur l'objet, mais que ces points de vue aient différentes bases cognitives. Les représentations multiples d'une requête peuvent donner différentes interprétations du besoin en information.

Une des approches de combinaison de multiples représentations de requêtes consiste à calculer les scores des documents directement depuis la fonction d'appariement document-requête en utilisant le même système de recherche mais différentes versions de la requête. Ensuite, les résultats obtenus par chacune des versions sont combinés pour avoir une seule liste finale. Ces versions sont issues soit des expressions d'une même requête par des chercheurs différents, soit des présentations d'une même requête dans des langages différents [24].

II.3 Réinjection de pertinence (relevance feedback) RF

La réinjection de pertinence, une des techniques de la reformulation de requêtes, connue aussi sous le nom de Relevance Feedback (RF), permet d'enrichir une requête initiale selon des informations extraites des éléments jugés pertinents par l'utilisateur [15].

Le processus de réinjection de pertinence, comporte principalement trois étapes: l'échantillonnage, l'extraction des évidences et la réécriture de la requête.

- *L'échantillonnage :* est une étape qui permet de construire un échantillon de documents à partir des éléments jugés par l'utilisateur. Cet échantillon est caractérisé par le nombre d'éléments jugés non pertinents et le nombre d'éléments jugés pertinents.
- *L'extraction des évidences* est l'étape la plus importante, elle consiste en général à extraire les termes pertinents qui serviront à l'enrichissement de la requête initiale. Plusieurs approches ont été développées, la plus reconnue est celle de Rocchio[24] adaptée au modèle vectoriel.
- *La réécriture de la requête* consiste à construire une nouvelle requête en combinant la requête initiale avec les informations extraites dans l'étape précédente.

Le processus général de la réinjection de pertinence peut être renouvelé plusieurs fois pour une même séance de recherche : on parle alors de la réinjection de pertinence à itérations multiples [15].

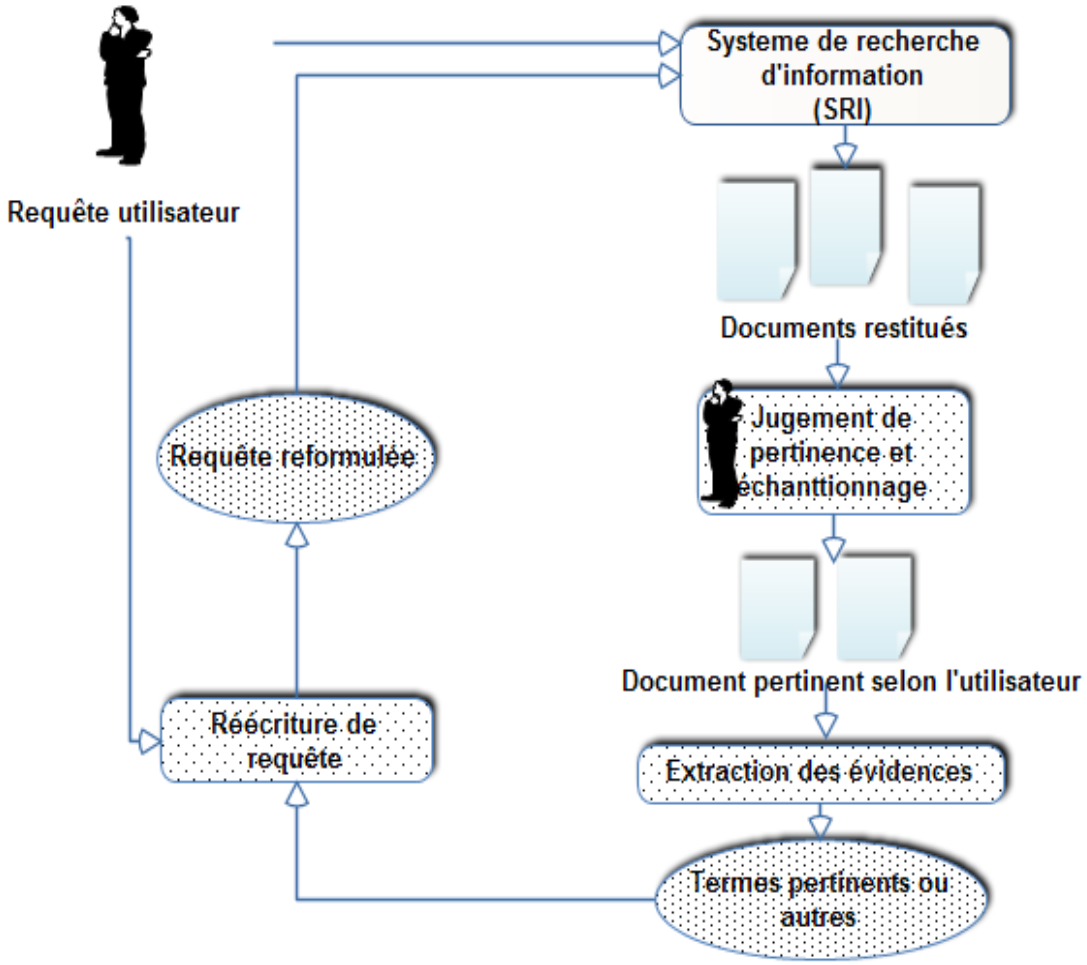


Figure II.2: Processus général de la réinjection de pertinence [15].

D'une manière générale, la phase d'échantillonnage ne présente pas de problématique spécifique. Le seul point abordé à ce niveau concerne le nombre d'éléments à évaluer pour pouvoir effectivement constituer un échantillon représentatif [15].

La problématique principale de la réinjection de pertinence réside dans les deux autres phases: l'extraction des termes (ils sont alors pondérés pour sélectionner les éléments les plus pertinents) et la réécriture de la requête avec repondération des termes [15], [24].

Dans la plupart des approches de la littérature, les deux phases sont effectuées avec des méthodes de pondération des termes similaires. Cependant certaines méthodes et particulièrement celles basées sur le modèle probabiliste, utilisent des méthodes de pondération différentes [24].

II.3.1 Méthodes d'extraction des termes

La reformulation de la requête telle qu'elle a été initialement utilisée par Wu et Salton[24] consistait à ajouter tous les termes des documents pertinents retrouvés en réponse à la requête lors du processus de recherche.

Cette méthode de sélection des termes peut être à l'origine de beaucoup de bruit (restitution de documents non pertinents). En effet, les termes dans les premiers documents pertinents restitués ne sont pas tous significatifs. L'idée d'utiliser seulement une sélection de termes a été proposée par Harman [15]. La question est de savoir quels termes utiliser pour étendre la requête initiale de façon à améliorer le rappel et la précision du système.

II.3.1.1 Approche de Harman

L'approche présentée par Harman consiste à sélectionner les dix premiers documents et à identifier parmi ceux-ci les documents pertinents. Harman a utilisé différentes techniques pour ordonner les termes afin de choisir les vingt meilleurs termes de la liste. Il a été démontré que la technique utilisée pour le tri des termes pertinents a un large impact sur la performance. Dans plusieurs techniques de tri, il utilise une mesure de bruit n_k calculée comme suit [15] :

$$n_k = \sum_{i=1}^{N} \frac{tf_{ik}}{f_k} log_2 \frac{f_k}{tf_{ik}}$$

Avec :

Tf_{ik} le nombre d'apparition du terme *k* dans le document *i*,

F_k le nombre d'apparition du terme *k* dans la collection et

N le nombre de termes dans la collection.

La technique a été étendue pour tenir compte du nombre de documents dans l'ensemble des documents pertinents contenant le terme *k* (p_k) et du nombre d'apparition du terme k dans l'ensemble des documents pertinents (rtf_k).

Harman a défini ainsi une autre mesure de bruit par rapport à l'ensemble des documents pertinents. Cette mesure est calculée comme suit [15]:

$$rn_k = \sum_{i=1}^{N} p_k \frac{tf_{ik}}{rtf_k} log_2 \frac{f_k}{tf_{ik}}$$

Harman a défini d'autres techniques de tri des termes. La technique qui conduit à de meilleurs résultats est basée sur une formule de pondération définie par Sparck-Jones et Robertson [15] :

$$w_{ij} = log_2 \frac{p_{ij}(1 - q_{ij})}{q_{ij}(1 - p_{ij})}$$

Avec :

W_{ij} poids du terme i dans la requête j,

p_{ij} la probabilité que le terme i apparaisse dans les documents pertinents pour la requête j,

q_{ij} la probabilité que le terme i apparaisse dans les documents non pertinents pour la requête j.

La sélection des termes ayant une valeur de poids importante revient à sélectionner les termes caractéristiques des documents pertinents avec une faible probabilité d'apparition dans les documents non pertinents.

Harman a également démontré que la meilleure méthode de sélection des termes issus des documents pertinents devient inefficace au-delà de 20 à 40 termes ajoutés [15].

II.3.1.2 L'approche de Robertson

Croft et al. et Robertson et al. [15] ont adopté une méthode de sélection de nouveaux termes sur la base d'une fonction qui consiste à attribuer à chaque terme un nombre traduisant sa valeur. Robertson propose la formule suivante pour calculer la valeur de sélection d'un terme :

$$selValue(i) = w_{ij} \times (p_i - U_i)$$

Avec :

W_{ij} poids du terme i dans la requête j,

P_i la probabilité $(di = 1/D\ est\ pertinent)$; et U_i la probabilité

$(di = 0/D\ est\ non\ pertinent)$.

Les termes sont alors triés en fonction de leurs valeurs de pertinence puis sélectionnés en utilisant un seuil prédéfini.

II.3.1.3 L'approche de Lundquist

Lundquist et al. ont étudié dans une autre technique de tri des termes, pour un terme *k*, ils ont associé une valeur :

$$p_k \times nidf$$

Où p_k est le nombre de documents dans l'ensemble des documents pertinents contenant le terme *k*, et *nidf* est une fréquence absolue inverse normalisée.
En utilisant la collection TIPSTER, Lundquist et al. ont démontré que cette formule conduit à de bonnes performances [15]. Par ailleurs, ils ont aussi démontré que l'utilisation des dix premiers termes (termes simples ou expressions) conduit à une amélioration de la précision moyenne de 31% par rapport à l'utilisation des cinquante premiers termes et vingt premières expressions.

II.3.1.4 L'approche de Boughanem

Boughanem et al. ont quant à eux étudié la reformulation de la requête sur un SRI basé sur l'approche connexionniste fondée sur les réseaux de neurones. Les termes ajoutés à la requête sont sélectionnés sur la base d'un seuil de cooccurrence avec les termes de la requête initiale. Ils ont conclu que la valeur idéale du seuil (c'est à dire la valeur permettant d'améliorer les résultats) varie de façon inversement proportionnelle à la taille de la base (base de cooccurrence avec les termes de la requête) et à la taille moyenne des documents [15].

II.3.1.5 L'approche de Buckley

Buckley et al. ont démontré que le taux de performance (Rappel- Précision) est davantage corrélé avec le nombre de termes ajoutés à la requêté qu'avec le nombre de documents initialement retrouvés. Cette idée est traduite par l'équation suivante :

$$RP(N) = A.log(N) + B.log(X) + C$$

Avec :
RP(N) la performance du système pour *N* documents restitués,
N le nombre de documents restitués, et *X* le nombre de termes ajoutés à la requête.
A, *B*, et *C* sont des constantes telles que *B*>>*A*>*C* [15].

II.3.2 Relevance feedback et modèles de recherche

Plusieurs techniques ont été proposées pour améliorer les performances des SRI. Ces méthodes apportent des solutions aux deux principales questions :

1. Comment peut-on retrouver plus de documents pertinents vis-à-vis d'une requête donnée?
2. Comment peut-on mieux exprimer la requête de l'utilisateur de manière à mieux répondre à son besoin?

II.3.2.1 Réinjection de pertinence dans le modèle vectoriel (Algorithme de Rocchio)

L'algorithme de reformulation de requêtes développé par Rocchio au milieu des années 60 [6], est une excellente illustration de la réinjection de pertinence adaptée initialement au modèle vectoriel. Bien que simple, cette méthode est largement utilisée en recherche d'information.

Rocchio considère que la restitution des documents pertinents est liée à la notion de "requête optimale". Cette dernière est censée maximiser la différence entre le vecteur des documents pertinents et celui des documents non-pertinents.

Comme l'utilisateur n'est pas en mesure de soumettre une requête optimale, la réinjection de pertinence doit permettre de rapprocher le vecteur de la requête initiale du vecteur moyen des documents pertinents et de l'éloigner du vecteur moyen des documents non pertinents. Ceci est mis en œuvre par repondération des termes initiaux et ajout de nouveaux termes pondérés à la requête initiale. Les poids servent à la discrimination des documents pertinents des documents non pertinents.

La forme standard de l'algorithme de Rocchio est donnée comme suit :

$$Q_{nlle} = \alpha Q_{init} + \frac{\beta}{|D_r|}\sum_{D_j \in D_r} D_J - \frac{\gamma}{|D_n|}\sum_{D_j \in D_r} D_J$$

Où:

|.| désigne le cardinal d'un ensemble,

Q_{nlle} : le vecteur de la nouvelle requête,

Q_{init} : le vecteur de la requête initiale,

D_r : l'ensemble des documents restitués et jugés pertinents par l'utilisateur,

D_n: l'ensemble des documents restitués et jugés non pertinents par l'utilisateur,

D_j : le j ème document d'un ensemble,

α, β et γ : des paramètres constants,

Le nouveau vecteur de requête est le vecteur de la requête initiale plus les termes qui différencient au mieux les documents pertinents des documents non- pertinents. Une requête reformulée contient de nouveaux termes (extraits des documents jugés pertinents) associés à de nouveaux poids. Si le poids d'un terme de la requête décroit vers zéro ou au-dessous de zéro, il est éliminé de l'ensemble des termes de la requête.

II.3.2.2 Réinjection de pertinence dans le modèle probabiliste

Dans le modèle probabiliste développé par Robertson, Sparck Jones et Van Rijsbergen[15], les documents et les requêtes questions sont également vus comme des vecteurs mais la mesure vectorielle de similarité est remplacée par une fonction probabiliste. On rappelle que le modèle probabiliste est basé sur la probabilité qu'un document soit pertinent à un utilisateur pour une requête donnée. Ce modèle est par essence même lié à la réinjection de pertinence, puisque ses paramètres sont estimés sur la base de la présence/absence des termes dans les documents pertinents et non pertinents.

Robertson et Sparck-Jones utilisent la formule de pondération des termes suivante :

$$W_i = log \frac{p_i\ (1 - q_i)}{q_i(1 - p_i\ )}$$

W_i le poids du terme *i*, avec

$$p_i = p(t_i = 1/D\ \ est\ pertinent) = \frac{r_i}{R}, q_i = p(t_i = 1/D\ \ est\ non\ pertinent) = \frac{n_i - r_i}{N - n_i}$$

Où $t_i = 1$ si le terme *i* indexe le document, $t_i = 1$ sinon

r_i le nombre de documents pertinents contenant le terme t_i,

R le nombre de documents pertinents pour la requête,

n_i le nombre de documents contenant le terme t_i et

N le nombre de documents dans la collection.

Les poids des termes ajoutés à la requête sont alors calculés selon la formule suivante:

$$w_i = log \frac{r_i/R - r_i}{n_i - r_i/(N - n_i) - (R - r_i)}$$

Harman a montré que l'utilisation de la formule de Sparck-Jones pour la repondération des termes, permet une augmentation de la précision de 25% sur la base Cranfield[15].

Croft a défini une méthodologie de repondération en utilisant une version révisée de la formule de pondération de Sparck-Jones. Plus précisément, la recherche initiale suit la fonction de pondération des termes suivante :

$$w_{ijk} = (c + idf_i).f_{ik}$$

Avec :

C une constante, f_{ik} la fréquence du terme ti dans le document

k, idf_i la fréquence absolue du terme ti dans la collection et *j* la requête.

Pour repondérer des termes par réinjection de pertinence, Croft se base sur la formule de Robertson. La formule de repondération est la suivante :

$$w_{ijk} = \left(c + log\frac{p_{ij}\left(1-q_{ij}\right)}{q_{ij}\left(1-p_{ij}\right)}\right).f_{ik}$$

Avec :

W_{ijk} le poids du terme ti dans la requête *j* et le document *k*,

$$p_{ij} = \frac{r_i + 0.5}{R + 1.0}\ si\ r_i > 0, p_{ij} = 0.01\ si\ r_i = 0,$$

$$p_{ij} = \frac{n_i - r_i + 0.5}{N - R + 1.0}\ si\ r_i > 0, p_{ij} = 0.01\ si\ r_i = 0,$$

$$f_{ik} = K + (1-K).\frac{freq_{ik}}{max(freq_k)}$$

Où $freq_{ik}$ est la fréquence du terme ti dans le document k, $max(freq_k)$ est le maximum des fréquences des termes dans le document *k* et *C*, *K* sont des constantes.

II.3.2.3 Réinjection de pertinence dans Okapi

La méthode de reformulation de requêtes dans Okapi se base sur la formule de poids proposée par Robertson et Sparck-Jones [6]. L'idée de base de la méthode consiste à extraire tous les termes des documents restitués et jugés pertinents. Ces termes sont ensuite ordonnés en fonction de leurs valeurs obtenues par des méthodes spécifiques.

Deux méthodes spécifiques ont été proposées par Robertson et Walker [6] pour la sélection des termes composant la nouvelle requête :

(1) la première méthode consiste à sélectionner un nombre limité de termes (généralement une valeur fixe) dans un ordre relatif à la valeur de sélection du terme (TSV pour Term Selection Value).

(2) la deuxième méthode est basée sur les statistiques des nouveaux termes significatifs, et nécessite un seuil absolu pour extraire les termes constituant la nouvelle requête.

Ces deux méthodes sont adaptées pour l'apprentissage des besoins en information des utilisateurs dans le cadre des systèmes de filtrage d'information.

Ce modèle de filtrage, développé par le laboratoire de recherche Microsoft de Cambridge, est basé sur le modèle probabiliste OKAPI et implanté dans le système Keenbow[6].

Le système de filtrage utilise la fonction de pondération BM25 pour calculer le score de similarité d'un document:

$$\sum_{t_i \in p} w_i^{(1)} \frac{(k_1+1).tf_i}{k+tf_i} \frac{(k_3+1).qtf_i}{k_3+qtf_i}$$

Avec :

P : un profil contenant les termes *ti*,

tfi: fréquence d'apparition du terme ti dans le document *D*,

qtfi : fréquence d'apparition du terme ti dans le profil*P*,

$K = ((1-b) + b.dl/avdl)$,

k_1, *b* et *k3* : paramètres dépendant de la nature des profils et du corpus.

II.3.2.4 Autre proposition considérant la dépendance des termes

Chacun des modèles vectoriel et probabiliste suppose l'indépendance entre les termes. En d'autres termes, la présence d'un terme dans un document n'influe pas sur la probabilité de l'existence d'un autre terme dans le même document. Bien que cette hypothèse simplifiée, facilite la construction de systèmes de recherche assez performants, l'indépendance des termes n'est pas fondée.

En effet, les mots sont reliés par leurs occurrences dans les documents qui peuvent refléter des relations sémantiques fondamentales entre les termes.

Des auteurs tels que Spiegel et Bennet [15] ont suggéré dès 1964 que cette dépendance de l'information peut être employée pour extraire d'autres termes pour l'extension de la requête. On distingue trois investigations sur la dépendance de l'information :

Van Rijsbergen, et al. ont proposé un arbre composé de nœuds représentant les termes et reliés par des arcs qui représentent les similarités entre deux termes [15]. Cette similarité est estimée selon la mesure d'association basée sur la distribution des probabilités des deux termes.

L'extension de la requête consiste à rajouter tous les termes directement liés aux termes de la requête initiale. L'ensemble des termes sera par la suite pondéré selon la formule de Robertson [15].

Les méthodes qui intègrent la dépendance des termes n'ont pas permis une amélioration des performances des systèmes de recherche [15]. Ceci peut être dû aux limitations informatiques pour calculer et stocker l'information de la dépendance. Bien que les méthodes d'indépendance des termes telles que celles basées sur le modèle probabiliste semblent simplifiées et n'expriment pas explicitement la dépendance des termes pertinents, elles permettent implicitement d'exprimer un certain degré de co-occurrence des termes.

Ceci peut être expliqué par le fait que les bons discriminateurs de pertinence sont les termes qui apparaissent plus fréquemment dans les documents pertinents que dans les documents non pertinents [15].

II.3.3 Autres formes de Réinjection de pertinence

II.3.3.1 Réinjection de pertinence négative

La majorité des techniques proposées en RF est basée sur la différence entre le contenu des documents pertinents et celui des documents non pertinents. Ces derniers se rapportent à deux groupes de documents :

1. ceux qui ont été jugés non pertinents explicitement par l'utilisateur ;
2. ceux qui n'ont pas été jugés par l'utilisateur. Ces documents sont soit non sélectionnés, l'utilisateur ne les a pas jugés, soit l'utilisateur les a rejetés implicitement sans fournir une évaluation de pertinence.

La différence entre ces deux groupes de documents non pertinents n'est pas exprimée dans les modèles probabiliste et vectoriel.

La RF utilisant le groupe des documents jugés explicitement non pertinents est appelée RF négative.

Le but de la réinjection de pertinence négative abordé par Sumner et al. était la suppression des documents non pertinents

Il est difficile de spécifier les conditions dans lesquelles un utilisateur doit considérer un document non pertinent. En effet, un document est considéré non pertinent s'il ne contient absolument aucune information pertinente, s'il ne contient aucune information liée aux besoins de l'utilisateur, s'il contient l'information liée au thème en question mais pas l'information pertinente, si le document n'est pas assez pertinent, etc. La question est quand un utilisateur devrait-il juger un document non pertinent ?

Dans le cas de la réinjection de pertinence positive, le genre de documents recherchés, ainsi que les changements effectués par le système apparaissent avec plus de clarté, contrairement à la réinjection négative pour laquelle l'utilisateur ne peut pas voir quels documents ont été supprimés.

Dans la pratique, la pertinence et la non pertinence ne sont pas des notions opposées. En général, un utilisateur qui juge un document pertinent donne souvent des raisons détaillées, mais les raisons de la non-pertinence sont susceptibles d'être basées sur ce qui manque dans le document, plutôt que sur ce qui est présent.

Plus les modalités d'évaluation sont compliquées moins les utilisateurs évaluent la pertinence, ce qui est le cas de l'évaluation de la non pertinence.

II.3.3.2 pseudo-réinjection de la pertinence

La pseudo-réinjection ou *blind Relevance Feedback*, utilise des techniques de réinjection automatique à l'aveugle pour construire une nouvelle requête. Plus précisément, le système de recherche restitue un ensemble de documents répondant à la requête initiale. Ainsi au lieu de juger explicitement les documents, on suppose que les k premiers documents comme étant pertinents (documents pseudo-pertinents). On peut également considérer les documents qui sont restitués en fin de liste comme non pertinents. L'idée de base derrière la pseudo réinjection de pertinence est qu'une itération de réinjection basée sur les documents les plus similaires à la requête initiale de l'utilisateur pourrait donner une meilleure restitution des documents.

Cette technique a été développée la première fois par Croft& Harper [6], en tant qu'un moyen d'estimation des probabilités dans le modèle probabiliste pour une première recherche. Depuis, cette technique a été largement étudiée pour améliorer les classements des documents en particulier dans le cadre de TREC [15].

Croft et Harper ont également indiqué que cette méthode peut avoir des impacts négatifs. En effet, si les documents considérés pour la réinjection contiennent peu d'informations pertinentes ou aucune, la réinjection ajoutera des termes à la requête initiale. Il est prouvé dans la majorité des travaux que la réinjection automatique présente une solution pratique pour l'amélioration des performances de la recherche en ligne sous un certain nombre de conditions. En particulier, c'est une technique très utile pour améliorer la recherche quand il s'agit de requêtes courtes ou de requêtes qui ne permettent pas de restituer assez de documents pertinents [15].

II.3.3.3 Réinjection de pertinence à itérations multiples

Campbell a abordé la notion du besoin dynamique à travers la notion de la «pertinence ostensive ». L'idée derrière la pertinence ostensive est que des documents jugés pertinents dans une itération courante de RF présentent des indicateurs plus intéressants que ceux retrouvés dans des itérations précédentes.

Cependant, les documents pertinents ne sont pas considérés d'égale importance mais d'importance variable. Campbell et Van Rijsbergen ont étendu le modèle probabiliste en intégrant un terme de " *vieillissement* " pour la pondération des termes pertinents. Ce concept permet de savoir si le document auquel appartient le terme est récemment jugé pertinent ou jugé dans des itérations antérieures. Des expérimentations préliminaires de cette approche ont montré que la pondération ostensive peut améliorer les résultats en moins d'itérations de recherche que les approches non-ostensives.

Ruthven et al. ont montré également que la pondération ostensive est bénéfique pour l'extension de la requête [15].

II.3.3.4 Combinaison d'algorithmes de réinjection de pertinence

Une autre application de la réinjection de pertinence est la combinaison des résultats de différentes méthodes de réinjection. Ceci pourrait impliquer de combiner les classements donnés par les différentes méthodes de réinjection sur les mêmes évaluations originales de requête et de pertinence, ou la combinaison des requêtes modifiées selon plusieurs méthodes de réinjection.

D'après Ruthven et Lalmas, la combinaison des évidences est une technique puissante pour la réinjection de pertinence, cependant, la majorité de techniques évaluées ont prouvé que cette combinaison est une technique très variable pour la recherche initiale, elle permet

d'améliorer la performance pour quelques requêtes mais aussi de la dégrader pour d'autres [15]. En outre, il est également très difficile de prévoir quelles sont les évidences à combiner pour différentes collections ou requêtes.

Conclusion

Dans ce chapitre, nous avons présenté les techniques de reformulation de la requête, les différentes ressources utilisées pour choisir les termes à ajouter soit externes/internes ou bien à partir des jugements de pertinence de l'utilisateur sur les documents restitués par le système. On a abordé la reformulation de la requête dans le cadre des différents modèles de recherche d'informations.

Le but commun à ces techniques est principalement l'amélioration de la qualité des résultats de recherche lorsque la seule évaluation de la similarité entre les requêtes et les documents n'est plus suffisante.

Nous décrivons dans le chapitre suivant notre approche de reformulation de requêtes.

Chapitre III

Conception

Nous avons présenté dans le chapitre précédent les différentes techniques de reformulation de la requête. Dans ce chapitre nous allons présenter une nouvelle approche dont l'objectif est d'améliorer les résultats de recherche, en se basant sur le jugement des utilisateurs afin de répondre efficacement à leurs besoins.

Dans ce qui suit, nous allons donner les fondements de base de notre approche qui est implémentée en utilisant la plateforme Terrier, ainsi que les outils de développement. En fin, nous présenterons quelques résultats expérimentaux obtenus.

Introduction

Il est évident qu'une requête construite à partir d'un ou deux mots clés permet de ne transcrire qu'une partie du besoin utilisateur. Si la formulation du besoin est une tâche complexe pour l'utilisateur de SRI, elle l'est d'autant plus lorsque ce besoin est imprécis et lorsque le corpus ne lui est pas familier. De ce fait, la nécessité d'approfondir la compréhension du besoin de l'utilisateur dans son contexte devient un point essentiel à l'amélioration des SRI [3]. Dans le but d'aider ce dernier à répondre le plus pertinemment possible au besoin de l'utilisateur, la réinjection automatique peut être bénéfique si les requêtes initiales permettent de retrouver des documents pertinents. Dans le cas contraire, elle provoque une dégradation des performances.

Des chercheurs comme Mitra et al. et Buckley et al. [3], ont essayé avec un certain succès de surmonter ce problème en améliorant le taux de précision dans les k meilleurs documents, c'est ce qu'on nomme habituellement la "haute précision". D'autres groupes de recherche comme Biron, Lee, Robertson et al., [3] se sont concentrés sur l'amélioration des techniques de réinjection afin de détecter les meilleurs termes à ajouter ainsi que sur le calcul de leurs poids.

Pour répondre aux limites de cette technique, il est nécessaire de faire intervenir l'utilisateur dans le processus de réinjection de pertinence. Dans ce chapitre, nous détaillerons une approche qui intègrera l'utilisateur dans la reformulation de la requête.

III.1 Architecture générale de l'approche suivie

La figure suivante illustre l'architecture de notre approche :

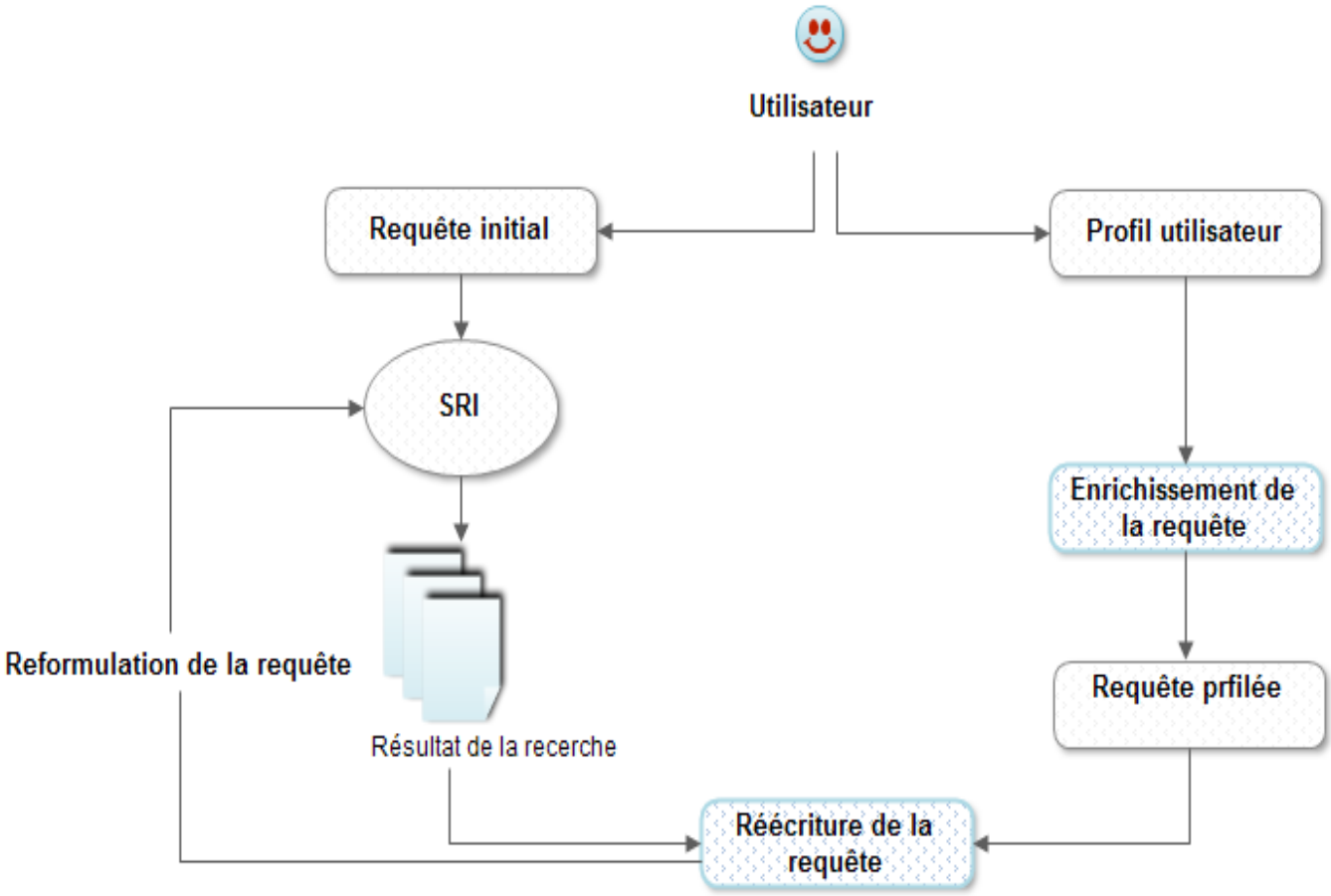


Figure III.1: Architecture générale de l'approche suivie.

III.2 Présentation de notre approche

Nous proposons un processus de reformulation de requêtes, en se basant sur l'utilisation d'une méthode de reformulation de requêtes, afin de fournir de meilleurs résultats de recherche.

La personnalisation d'accès à l'information consiste à intégrer le profil utilisateur dans au moins l'une des phases du processus de recherche. Son intégration dans la phase de la reformulation de requêtes consiste à augmenter la requête initiale par des termes issus du profil utilisateur, défini dans notre approche, par son centre d'intérêt.

L'objectif principal de notre approche de reformulation, consiste alors à identifier des sources de données pertinentes selon une requête initiale profilée, ensuite de reformuler les requêtes utilisateurs en fonction de ces sources de données.

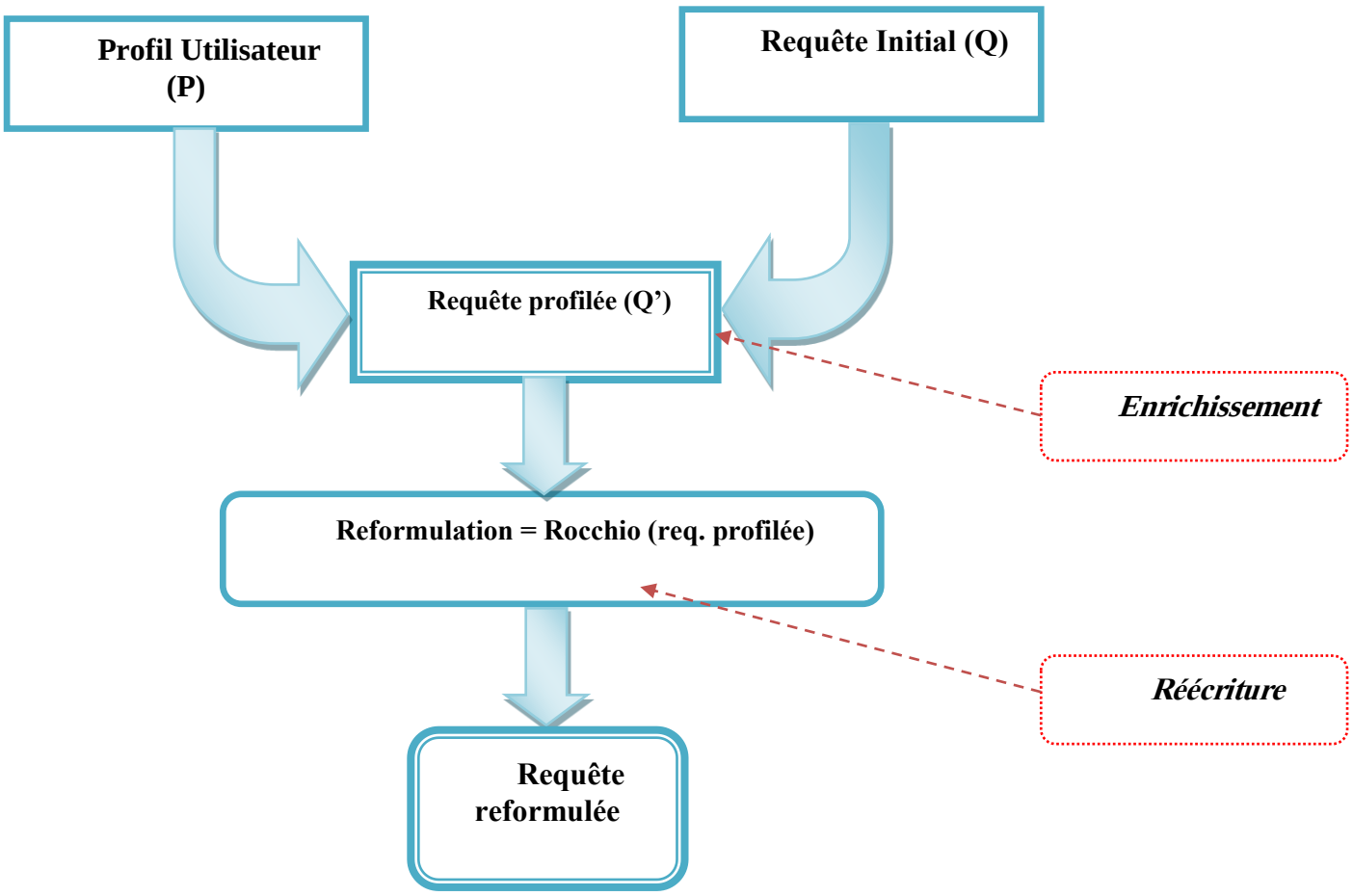


Figure III.2 : Reformulation de la requête en deux étapes.

III.2.1 Enrichissement de la requête

L'utilisateur ne sait pas choisir les bons termes qui expriment le mieux ses besoins d'information. L'enrichissement des requêtes est un processus ayant pour objectif de générer une nouvelle requête plus adéquate que celle initialement formulée par l'utilisateur en rajoutant de nouveaux termes du profil utilisateur et/ou supprimant des termes inutiles.

- ***Profil utilisateur :***

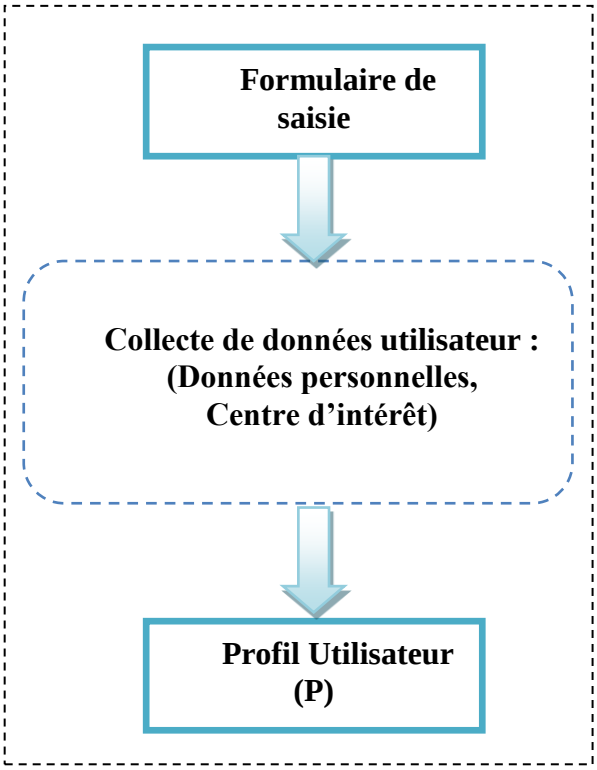


Figure III.3: Profil utilisateur.

On appelle profil utilisateur toute structure qui permet de modéliser et de stocker les données caractérisant l'utilisateur. Ces données représentent les centres d'intérêt, les préférences et les besoins en informations de l'utilisateur. L'utilisateur peut les formuler à travers l'introduction d'un ensemble de mots clés, pour décrire ses centres d'intérêt. Sur la base de ces mots clés, le système pourra lui présenter des informations qui répondent au mieux à ses attentes.

Le but de ce profil est d'utiliser ces informations pour personnaliser la requête, autrement dit à profiler la requête.

Dans notre approche, nous avons construit le profil utilisateur d'une manière explicite, en remplissant manuellement les cases des paramètres associés.

III.2.2. Réécriture de la requête

L'ajout d'informations complémentaires aux requêtes est un processus très délicat à cause du risque d'insertion d'informations incorrectes qui provoquerait le retour de résultats sans aucun intérêt. Donc nous avons ajouté une 2eme étape qui s'agit d'utiliser le résultat de la première recherche avec la formule de rocchio.

En utilisant la formule de rocchio classique, au lieu d'utiliser la requête initiale formulée par l'utilisateur, nous allons intégrer la requête profilée. En effet, la formule de rocchio classique peut être bénéfique sauf si la requête initiale permet de retrouver des documents pertinents.

La réinjection de pertinence de la formule de rocchio dans notre cas doit permettre de rapprocher le vecteur de la requête profilée du vecteur moyen des documents pertinents et de l'éloigner du vecteur moyen des documents non pertinents.

Une modification apportée à la formule de Rocchio, qui permet d'utiliser le centre d'intérêt d'un utilisateur est décrite dans l'équation suivante :

$$Q_{nlle} = \alpha Q_{profilé} + \frac{\beta}{|D_r|}\sum_{D_j \in D_r} D_J - \frac{\gamma}{|D_n|}\sum_{D_j \in D_r} D_J$$

Où:

|.| désigne le cardinal d'un ensemble,

Q_{nlle} : vecteur de la nouvelle requête,

$Q_{profilé}$: vecteur de la requête profilée obtenu par l'enrichissement de la requête initiale en utilisant le profil utilisateur,

D_r : ensemble des documents restitués et jugés pertinents par l'utilisateur,

D_n: ensemble des documents restitués et jugés non pertinents par l'utilisateur,

D_j : le j ème document d'un ensemble,

α, β et γ : paramètres constants,

III.3 Intégration de notre approche dans la plate-forme terrier

III.3.1 La plate-forme Terrier

Terrier, ***TeRabyteRetrIEveR***, est un moteur de recherche robuste et efficace, utilisé avec succès pour la recherche Ad-hoc, la recherche web et la recherche multilingue dans des environnements centralisés et distribués. Terrier offre une plate-forme idéale destinée à l'indexation de volumes importants de documents: jusqu'à 25 millions de documents. Il est développé par le département informatique de l'université *Glasgow*. C'est un logiciel open source écrit en Java.

Comme tout moteur de recherche, terrier permet :

- L'indexation classique : extraction des mots clés des documents appartenant à une collection et les stocker dans un index.
- Recherche des documents pertinents pour répondre aux requêtes formulées par l'utilisateur.
- Evaluation des résultats de la recherche.

Nous présentons une description détaillée de ce système dans Annexe.

III.3.1.1 Le processus d'indexation de Terrier

L'indexation dans Terrier est divisée en quatre procédures et à chaque procédure, des classes java peuvent être ajoutées pour la personnalisation du système.

Les quatre procédures sont :

1. Splitter la collection de documents : consiste à parcourir l'ensemble du corpus reçu en entrée par Terrier et envoyer chaque document à l'étape suivante.
2. Extraction des termes (Tokenize Document) : qui consiste à parser chaque document reçu et en extraire les différents termes.
3. Traitement des termes extrait avec TermPipelines : consiste en l'élimination des mots vides et la lemmatisation des termes.
4. La construction de l'index.

(Toutes ces étapes, les modules en charge de leur exécution ainsi que les fichiers résultants de cette indexation sont présentés de façon plus détaillée dans l'annexe1.)

Les différentes classes associées au processus d'indexation sont organisées dans un ensemble de packages dont on trouve :

Org.terrier.indexing : ce package contient les différentes classes permettant de réaliser un ensemble d'opérations sur la collection des documents, dans le but d'extraire les termes de tous les documents de la collection.

Org.terrier.terms : les classes qui se trouvent dans ce package permettent d'effectuer un ensemble de traitements sur les termes extraits. Parmi ces traitement, l'élimination des mots vides, lemmatisation des termes, …etc.

Org.terrier.structures : les classes de ce packages permettent la construction d'un ensemble de structures où un ensemble de données est stocké, parmi ces structure on a :

- ✓ Lexicon : qui stocke les informations de chaque terme de la collection ;
- ✓ Inverted Index : où le fichier inverse est stocké ;
- ✓ Document Index : qui contient des informations sur les différents documents de la collection ;
- ✓ …etc.

La figure ci-dessous donne un aperçu de l'interaction des principaux composants impliqués dans le processus d'indexation.

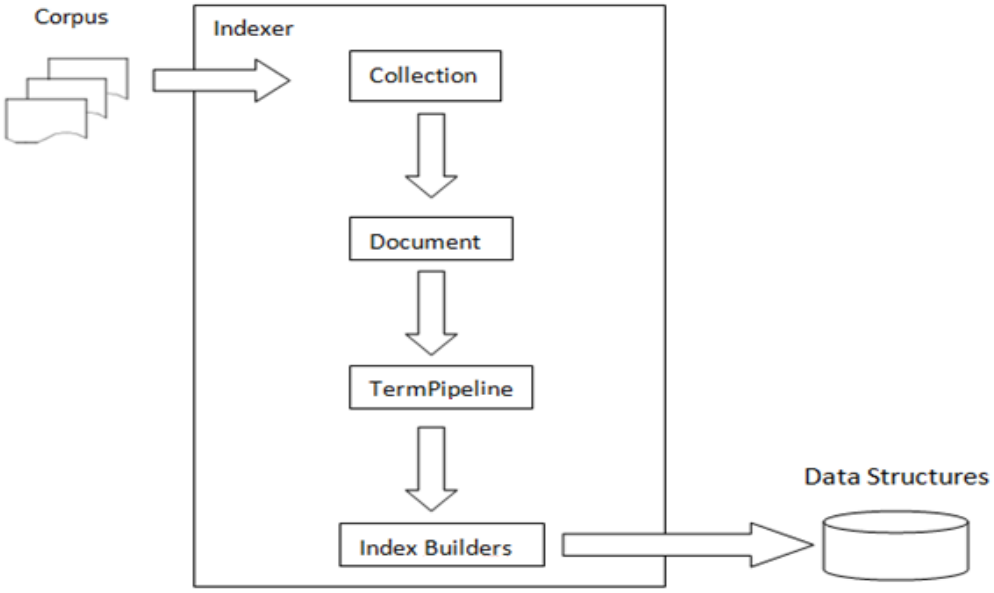


Figure III.3: processus d'indexation de Terrier [28].

III.3.1.2 Le processus de recherche de Terrier

Durant le processus de recherche, chaque requête doit passer par les étapes suivantes :

1. Parsing : qui se charge de tokenizer la requête.

2. Pré-processing : qui applique le TermPipeline à la requête. Elimine les mots vides et les lemmatise.

3. Matching : responsable de l'initialisation du Weighting Model et du calcul des scores entre la requête et les documents.

4. post-filtrage : filtrage des résultats.

5. post-traitement : peut modifier le ResultSet, par exemple, par un procédé QueryExpansion afin de générer un meilleur classement de documents.

(Toutes ces étapes et les modules en charge de leur exécution seront détaillés en Annexe1.)
Parmi les packages où se trouve les classes associées au processus de recherche on a :

Org.terrier.Matching : contient les classes qui permettent de déterminer les documents répondant à la requête.

Org.terrier.matching.models : on trouve dans ce package les différents modèles de pondération dont TF-IDF, BM25, …etc ;

La figure III.4 donne un aperçu de l'interaction des composants Terrier dans la phase de recherche :

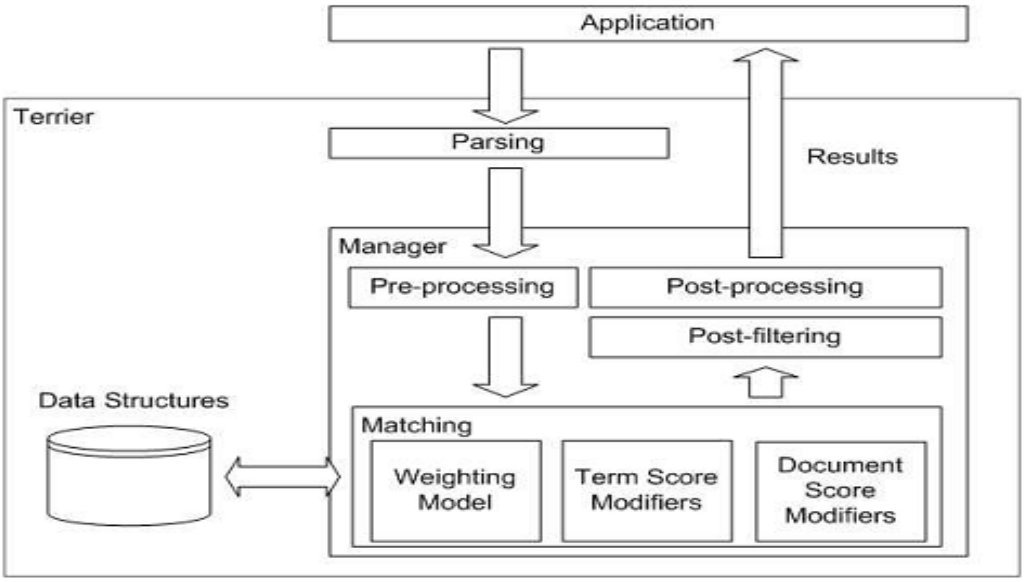


Figure III.4 : processus de recherche de Terrier [28].

III.3.2. Le processus de reformulation de la requête (Query Expansion)

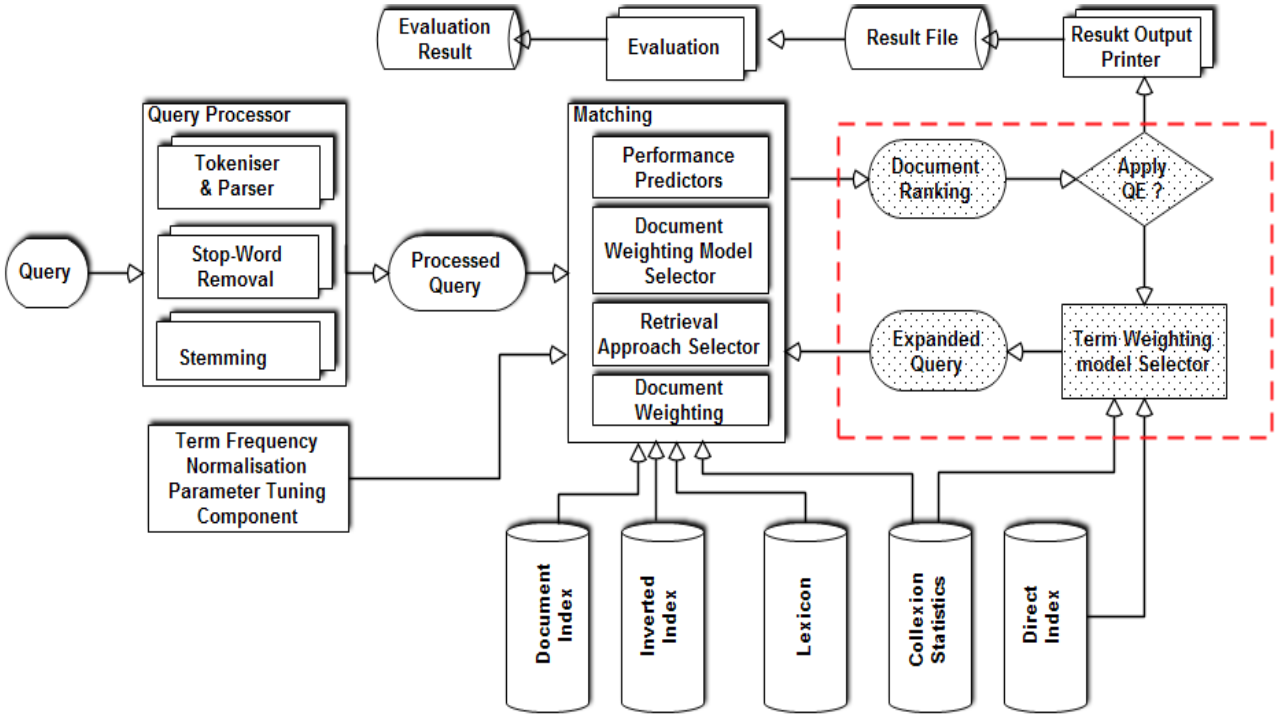


Figure III.5 : Architecture générale de recherche et de reformulation [29].

Après l'indexation, les termes sont stockés en structures de données suivantes:

- ✓ Lexicon : qui stocke les informations de chaque terme de la collection ;

- ✓ Inverted Index : où le fichier inverse est stocké ;
- ✓ Document Index : qui contient des informations sur les différents documents de la collection ;
- ✓ …etc.

Index direct :

L'index direct enregistre pour un document les termes qui apparaissent dans ce dernier. Il est souvent utilisé pour la reformulation de la requête, la classification et comparaison des documents.

Index inversé :

L'index inversé contrairement à l'index direct enregistre pour un terme les documents dans lesquels il apparait, il contient aussi la position de chaque terme et sa fréquence dans ces documents.

Comme il est illustré dans la **figure III.5,** la reformulation de la requête se fait après avoir des résultats de la première recherche. Telle qu'une autre requête générée de la requête précédente.

Comme nous avons dit précédemment, notre objectif à travers cette proposition est d'intégrer le profil utilisateur dans la phase de la reformulation de la requête.
Donc on ajoute un terme qui présente le centre d'intérêt de l'utilisateur à la requête initiale pour construire une requête profilée et on reformule cette dernière avec l'algorithme de rocchio en se basant sur les documents restitués dans la première recherche. Ce terme ajouté va orienter la reformulation (extraction des termes) vers les documents qui sont réellement pertinents.

❖ **Présentation du profil utilisateur dans terrier :**

Dans la plate-forme terrier, on utilise un corpus de test prédéfini qui se compose de:

- o un ensemble de documents (collection AP88);
- o un ensemble de requêtes (topics);
- o la liste de documents pertinents pour chaque requête (qrels).

Donc au lieu d'ajouter à la requête initiale un terme extrait dans le centre d'intérêt qui est construit à partir du profil utilisateur ; on prend ce terme automatiquement dans les documents pertinents par rapport à la requête utilisée.

Conclusion

Nous avons présenté dans ce chapitre la description détaillée de l'approche suivie pour la reformulation de la requête, puis une architecture d'intégration de cette approche dans la plate-forme Terrier. Nous avons en particulier décrit l'implémentation du module de Query Expansion dans Terrier.

Nous consacrons le prochain chapitre à l'implémentation et l'évaluation de notre approche en utilisant la plate-forme Terrier.

Chapitre IV

Implémentation et tests

Nous avons décrit dans le chapitre précédent la présentation de notre approche et son intégration dans la plate-forme Terrier.

Dans ce chapitre, nous présenterons l'environnement technologique de développement de notre approche, puis nous donnerons quelques aspects de son implémentation, ainsi que le détail des tests et évaluations.

Introduction

Les expérimentations et les évaluations sont articulés autour de la comparaison de notre modèle proposé pour la reformulation de requêtes et celui de Rocchio classique, Les expérimentations que nous décrivons dans ce chapitre ont été effectuées sur une collection standard de RI à savoir la collection de test AP88 sous la plate forme terrier.

IV.1 Environnement de développement

Dans nos expérimentations, nous avons utilisé la plate-forme Terrier qui est un open source écrit en java donc ce dernier est imposé à l'utilisation. Dans la section suivante, nous présentons les outils utilisés.

IV.1.1 Présentation de l'environnement du travail

Le travail que nous avons réalisé a été développé sur un microordinateur ayant les principales caractéristiques suivantes:

- Un microprocesseur Intel(R) Core(TM) i3 ;
- Microprocesseur de fréquence : 2.40 GHZ ;
- RAM de capacité : 3Go ;
- Disque dur de capacité : 500 Go.

Nous avons travaillé sous le système d'exploitation Microsoft Windows XP et utilisé le langage de programmation orienté objet Java.

IV.1.2 Le langage java

Java est un langage de programmation à usage général, évolué et orienté objet dont la syntaxe est proche du celle du langage C. apparu en 1995 début 1996 est développé par SUN Microsystems [26]. Il permet d'exécuter des programmes au travers d'une machine virtuelle (JVM).

Il possède un certain nombre de caractéristiques qui ont largement contribué à son énorme succès.

IV.1.2.1 Caractéristique de Java : [26]

- **Orienté objets**

La programmation orientée objets présente l'immense intérêt de faciliter la réutilisabilité des composants développés. Java est un langage orienté objet.Ecrire un programme revient à écrire une classe d'objets avec ses données et les méthodes qui permettent d'y accéder.

- **Simple**

Java hérite une grande partie de la syntaxe du langage C++, et dans le but d'écrire des codes facilement et sans erreurs, il en a été dépouillé de tous les mécanismes complexes, redondants ou devenus inutiles, tels que la gestion des pointeurs, la gestion de la mémoire (la libération de la mémoire en particulier n'est plus à la charge du développeur mais est gérée de manière automatique par ramasse-miettes intégré), l'héritage multiple,...etc.

- **Robuste**

Plusieurs raisons font que le code généré est effectivement plus robuste qu'avec d'autres langages, et risque donc moins de générer des erreurs :

- Le développeur n'a pas la possibilité d'accéder aux pointeurs, réduisant ainsi le risque d'écraser des données par erreur dans une zone mémoire.

- Le mécanisme de gestion des exceptions qui permet une meilleure maîtrise des erreurs. Ce mécanisme permet en effet de gérer des événements non souhaités (ex : division par zéro) en imposant un traitement adapté (ex : arrêt du programme) évitant ainsi un résultat non maîtrisé (ex : arrêt brutal du programme)…

- **Sécurité**

Java est utilisé en réseaux, de nombreux mécanismes ont été mis en œuvre, pour renforcer la sécurité de ce dernier, afin d'éviter par exemple qu'un programme malfaisant en provenance d'un serveur distant, ne vienne modifier ou supprimer les données d'un fichier local.

- **Doté d'une bibliothèque de classes très complète**

Java est livré avec de nombreuses API (Application Programming Interface) allégeant la tâche du développeur, en lui évitant de réécrire des composants couramment utilisés, tels que ceux permettant la manipulation de types de données ou d'éléments d'interfaces graphiques, la gestion de fichiers, de réseaux ou de bases de données...etc.

- **Multithread**

Il est nécessaire de mettre en œuvre des threads (processus légers) qui permettent de gérer simultanément plusieurs tâches au sein d'une même application.
Java permet d'adapter ce mécanisme via la classe Thread qui offre une palette de méthodes, permettant non seulement de lancer et d'arrêter un thread mais aussi, de vérifier les statuts de plusieurs threads, et de synchroniser ceux-ci en fonction par exemple de leur état.

- **Dynamique**

La modification d'une classe ne nécessite pas une recompilation pour prendre en compte les changements effectués (à l'inverse du langage C++), puisque les liens entre les objets sont résolus lors de l'exécution, cela apporte sans aucun doute plus de flexibilité dans un environnement de composants.

- **Multi plates-formes**

Les programmes tournent sans modification sur tous les environnements (Windows, UNIX).

IV.1.3 Présentation d'Eclipse

Pour le développement de notre approche, nous avons choisi l'environnement de programmation « éclipse GALILEO » sous Windows XP.
Eclipse est un environnement de développement intégré, libre, extensible, universel et polyvalent, permettant de créer des projets de développement mettant en œuvre n'importe quel langage de programmation. Eclipse IDE est principalement écrit en Java (à l'aide de la

bibliothèque graphique SWT, d'IBM), ce langage, grâce à des bibliothèques spécifiques, est également utilisé pour écrire des extensions.

La spécificité d'Eclipse IDE vient du fait que son architecture est totalement développée autour de la notion de plugin (en conformité avec la norme OSGi) : toutes les fonctionnalités de cet atelier logiciel sont développées en tant que plug-in [27].

La figure ci-dessous illustre l'interface d'Eclipse :

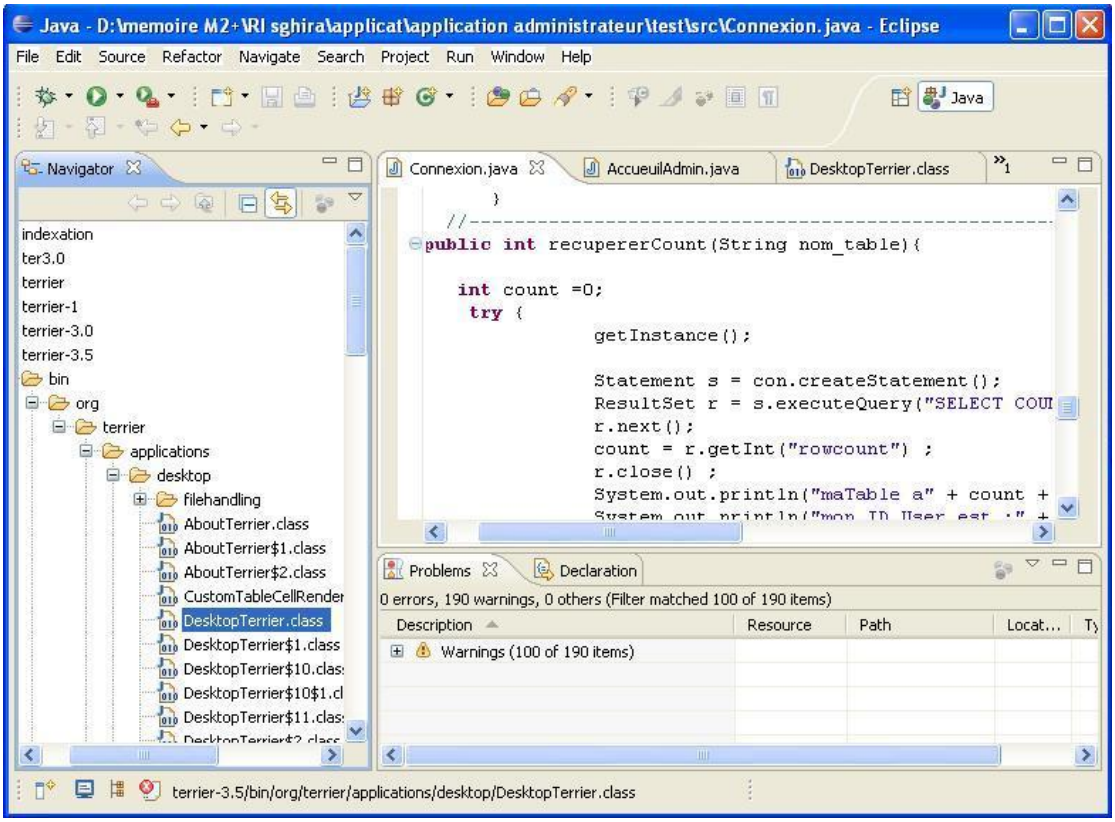


Figure IV.1: Interface d'Eclipse.

Les avantages d'Eclipse

Eclipse possède de nombreux points forts qui sont à l'origine de son énorme succès dont les principaux sont :

- Une plate-forme ouverte pour le développement d'applications et extensible grâce à un mécanisme de plug-ins
- Plusieurs versions d'un même plug-in peuvent cohabiter sur une même plate-forme.
- Un support multi langage grâce à des plug-ins dédiés : Cobol, C, PHP, C# , ...
- Support de plusieurs plates-formes d'exécution : Windows, Linux, Mac OS X, ...

- Malgré son écriture en Java, Eclipse est très rapide à l'exécution grâce à l'utilisation de la bibliothèque SWT

- Les nombreuses fonctionnalités de développement proposées par le JDT (refactoring très puissant, complétion de code, nombreux assistants, ...)

- Une ergonomie entièrement configurable qui propose selon les activités à réaliser différentes « perspectives ».

- Un historique local des dernières modifications

- La construction incrémentale des projets Java grâce à son propre compilateur qui permet en plus de compiler le code même avec des erreurs, de générer des messages d'erreurs personnalisés.

- Une exécution des applications dans une JVM dédiée sélectionnable avec possibilité d'utiliser un débogueur complet (points d'arrêts conditionnels, visualiser et modifier des variables, évaluation d'expression dans le contexte d'exécution, ...)

- Propose le nécessaire pour développer de nouveaux plug-ins.

- Possibilité d'utiliser des outils open source : CVS, Ant, Junit

- La plate-forme est entièrement internationalisée dans une dizaine de langue sous la forme d'un plug-in téléchargeable séparément.

IV.2 Implémentation

Nous présentons dans ce qui suit les différentes étapes de reformulation de la requête selon notre approche.

A. Récupération du centre d'intérêt

Après avoir récupéré les documents pertinents par rapport à la requête en cours du traitement dans le fichier qrels, on indexe cette nouvelle collection et on récupère les termes avec leurs Nt (Nt : nombre de document qui contient le terme t_i), avec la commande « *--printlexicon* ».

Algorithme1

Début

Récupération du terme avec la plus grande valeur ;

Enregistrer ce terme dans la table centre d'intérêt ;

Fin.

B. Reformulation de la requête :

Algorithme2

// L'ajout du terme (centre d'intérêt) à la requête initiale

Début

Chargement de la requête initiale ;

//construction de la requête profilée

Ajouter le terme (centre d'intérêt) ;

Lancer la reformulation avec la modèle de Rocchio ;

Fin.

IV.3 Evaluation expérimentale de notre approche

L'objectif de cette évaluation et de mesurer les performances et la viabilité de notre approche. Nous présentons dans ce qui suit le cadre d'évaluation (collection de tests et protocole d'évaluation) ainsi que les résultats expérimentaux.

IV.3.1 Collection de tests

Les collections de tests ont été traditionnellement utilisées en RI pour évaluer les stratégies de recherche. Plus particulièrement, une collection de référence doit traduire la subjectivité de pertinence des utilisateurs. D'autre part, elle doit contenir une masse d'informations assez importante et variée pour constituer un environnement standard d'interrogation.

Différentes collections de tests sont utilisées en RI. Parmi elles, la collection AP88 que nous avons utilisé pour nos expérimentations. Nous allons dans ce qui suit décrire cette collection.

La collection de test TREC, AP88 (AssociatedPress 1988) qui est de taille moyenne et qui contient des documents plats. Nous avons utilisé un ensemble de requêtes qui se trouve dans le fichier Topics251-300.

Le tableau ci-dessous montre quelques statistiques sur la collection AP88

Collection	Taille	Documents	Topics
AP88	237 Mo	79 919	251-300

Un document est identifié comme suit

<DOC>

<DOCNO> AP880212-0001 </DOCNO>

<FILEID>AP-NR-02-12-88 2344EST</FILEID>

<FIRST>u i AM-Vietnam-Amnesty 02-12 0398</FIRST>

<SECOND>AM-Vietnam-Amnesty,0411</SECOND>

<HEAD>Reports Former Saigon Officials Released from Re-education Camp</HEAD>

<DATELINE>BANGKOK, Thailand (AP) </DATELINE>

<TEXT>

More than 150 former officers of the overthrown South Vietnamese government have been released from a re-education camp after 13 years of detention, the official Vietnam

News Agency reported Saturday.

The report from Hanoi, monitored in Bangkok, did not give specific figures, but said those freed Friday included an ex-Cabinet minister, a deputy minister, 10 generals, 115 field-grade officers and 25 chaplains.

It quoted Col. Luu Van Ham, director of the Nam Ha camp south of

Hanoi, as saying all 700 former South Vietnamese officials who had been held at the camp now has been released…

…

... He said many of the former inmates would return to their families in Ho Chi Minh City, formerly the South Vietnamese capital of Saigon.

The amnesties apparently are part of efforts by Communist Party chief Nguyen Van Linh to heal internal divisions and improveVietnam's image abroad.

</TEXT>

</DOC>

Exemple de requête : Dans le fichier / etctrec.topics.list, nous devons spécifier quel fichier contient les requêtes à traiter.

<top>

<num> Number: 251

<title> Exportation of Industry

<desc> Description:

Documents will report the exportation of some part of U.S. Industry to another country.

<narr> Narrative:

Relevant documents will identify the type of industry being exported, the country to which it is exported; and as well will reveal the number of jobs lost as a result of that exportation.

</top>

IV.3.2 Protocole d'évaluation

Nous avons effectué notre évaluation en utilisant Terrier-3.5 selon le Protocol d'évaluation TREC :

Pour chaque requête, les documents restitués par le système sont examinés et des précisions sont calculées à différents points (à 1, 2, 3, 4, 5, 10, … 200 premiers documents restitués). La précision à x (exemple précision à 5) définie le taux de documents pertinents parmi les x premiers documents retrouvés.

Une précision moyenne est ensuite calculée pour chaque requête. Il s'agit de la moyenne des précisions de chaque document pertinent pour cette requête. La précision d'un document est la précision à x, tel que x est le rang de ce document dans l'ensemble des documents pertinents retrouvés. Finalement, la précision moyenne pour l'ensemble des requêtes est calculée permettant d'obtenir une mesure de la performance globale du système.

IV.3.3 Résultats expérimentaux

Pour évaluer la performance de l'intégration du profil utilisateur dans la reformulation de la requête, nous avons réalisé une série d'expérimentations dans le but de les comparer avec l'expansion de la requête classique de terrier en utilisant le modèle de rocchio.

- **Evaluation de notre approche de reformulation de requêtes**

Les valeurs de précision moyenne pour chaque requête, avec la reformulation classique et notre reformulation sont montrées dans le tableau et la figure ci-dessous.

Requêtes	*Reformulation Classique*	*Reformulation Proposée*
251	0.0052	0.0070
252	0.0101	0.0139
253	1.0000	1.0000
254	0.1401	0.2511
255	0.0552	0.4368
256	0.0355	0.0453
257	0.2951	0.3306
258	0.1322	0.7562
259	0.7341	0.7342
260	0.0022	0.0118

Tableau IV.1: Valeurs de moyenne précision.

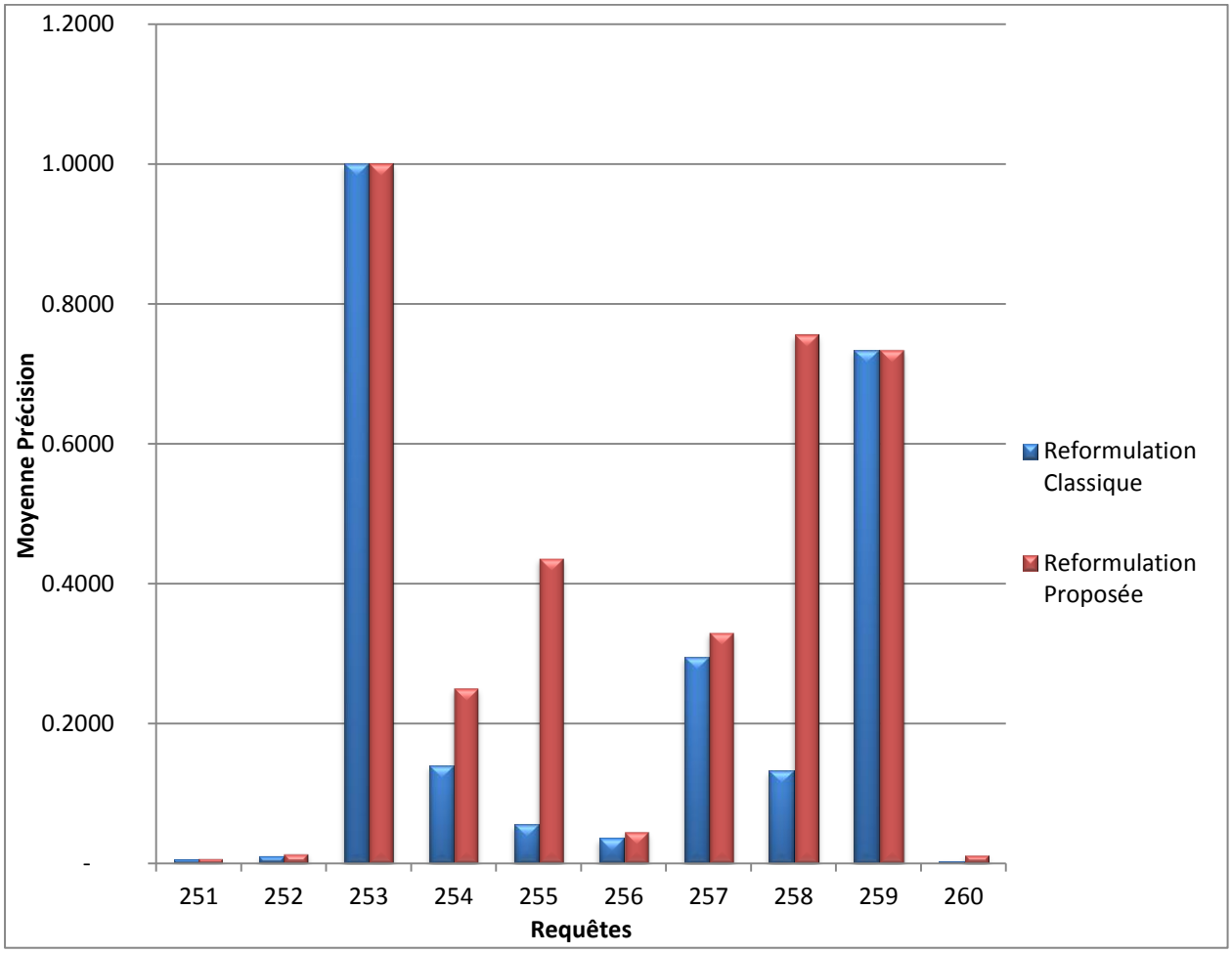


Figure IV.1 : L'apport de notre reformulation

Pour évaluer la performance de notre approche de reformulation de requête, nous avons réalisé une série d'expérimentations dans le but de comparer cette approche par rapport à la reformulation classique.

Les résultats obtenus montrent que La précision moyenne est améliorée pour la plupart des 10 requêtes testées.

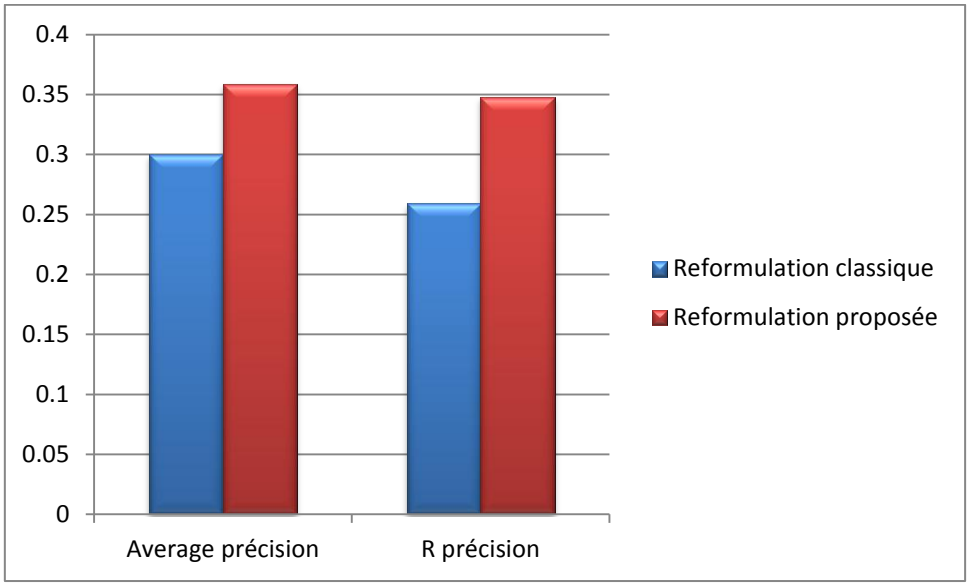


Figure IV.2 : Augmentation de Average précision et R précision

Ces résultats montrent que notre approche de reformulation de requêtes offre une précision moyenne et une précision réelle plus importante que la reformulation classique. En effet, la précision moyenne est passée de 0.2998 pour la reformulation classique à 0.3587 pour la reformulation proposée. La R-Precision est passée de 0.2589 à 0.3478.

Conclusion

Dans ce chapitre, nous avons présenté les détails de l'implémentation ainsi que les expérimentations et résultats associés à notre approche. Selon les résultats obtenus, il résulte une amélioration de la précision moyenne et de la R-précision lorsqu'on utilise notre approche de reformulation.

Conclusion

Conclusion

Les travaux présentés dans ce rapport s'inscrivent dans le cadre des études sur les systèmes de recherche d'information. Ils concernent la reformulation de la requête par réinjection de pertinence.

La reformulation de requêtes est une technique utilisée en recherche interactive d'information pour permettre à l'utilisateur de fournir des informations supplémentaires dans le but d'aider le système à restituer les documents les plus pertinents.

Nous avons proposé un processus de reformulation de requêtes par réinjection de pertinence basé sur l'enrichissement de la requête initiale avant la réécriture de cette dernière dans le but d'orienter le besoin de l'utilisateur vers les documents pertinents.

Nous avons testé notre approche sur la plate-forme terrier 3.5, les résultats de cette approche ont effectivement amélioré les performances du SRI dans la restitution de documents en réponse aux besoins de l'utilisateur. La précision moyenne a augmenté de plus de 10%.

Donc l'intégration du profil utilisateur au niveau de la reformulation de la requête joue un rôle intéressant pour la reformulation de requêtes afin d'affiner la recherche.

Pour clore ce modeste mémoire, nous dirons que ce travail nous a permis d'aborder le domaine de la recherche d'information, d'enrichir nos connaissance et plus précisément :

- ✓ Approfondir nos connaissances sur la recherche d'information ;
- ✓ Mettre l'accent sur la manière dont les SRI fonctionnent dans plate-forme terrier.

Comme perspectives de travail, nous envisageons d'appliquer des méthodes implicites pour construire la requête profilée. En effet, la présentation du profil utilisateur avec un formulaire à saisie est ennuyante pour l'utilisateur.

Références

Références

[1] : **Jean-Pierre Chevallet, Yves Chiaramella**, Vers la recherche d'informations à base de termes Equipe Modélisation et Recherche d'Information Multimédia (MRIM) du laboratoire de Communication Langagière et Interaction Personne-Système (CLIPS) de l'MAG).

[2] : **L. Tamine, M. Boughanem**, Un Algorithme génétique spécifique à une reformulation multi-requêtes dans un système de recherche d'information IRIT-SIG, Université Paul Sabatier septembre 2001.

[3] : **Aurélien Saint Requier, Gérard Dupont, Sébastien Adam, Yves Lecourtier**, Évaluation d'outils de reformulation interactive de requêtes, Université de Rouen, LITIS, France.

[4] : **I.KAMOUN FOURATI, M. TMAR, M. BOUGHANEM, A.BEN HAMADOU**, Reformulation automatique de la requête en recherche d'information structurée, Laboratoire MIRACL.et IRIT-SIG Université Paul Sabatier.

[5] : **H. Aliane, Z. Alimazighi, R.O.Boughacha, T. Djelliout**. Un Système de reformulation de requêtes pour la recherche d'information, Centre de Recherche sur l'Information Scientifique et Technique et Université des Sciences et de la Technologie Houari Boumedienne, Alger, Algérie.

[6] : Thèse, **Hamid TEBRI**, Formalisation et spécification d'un système de filtrage incrémental d'information, l'Université Paul Sabatier de Toulouse.

[7] : **Thierry Urruty,** Optimisation de l'indexation multidimensionnelle : application aux descripteurs multimédia, Année 2007.

[8] : Thèse, **Yannick Loiseau,** Recherche flexible d'information par filtrage flou qualitatif, 2004.

[9] : Thèse, **Mustapha BAZIZ,** indexation conceptuelle guidée par ontologie pour la recherche d'information, 2005.

[10] : Thèse, **SAID L'HADJ Lynda**, Recherche Conceptuelle d'Information et Modèle d'Indexation Mixte : concepts-mots, 2009.

[11] : Thèse, **Mariam DAOUD**, Accès personnalisé à l'information : approche basée sur l'utilisation d'un profil utilisateur sémantique dérivé d'une ontologie de domaines à travers l'historique des sessions de recherche. 2009.

[12] : Thèse, **Fabienne MOREAU,** Revisiter le couplage traitement automatique des langues et recherche d'information, 2006.

[13] : Thèse, **ZemirliW.Nesrine**, Vers le développement d'un système de recherche d'information personnalisé intégrant le profil utilisateur, 2004.

[14] : Thèse, **SAID L'HADJ Lynda**, Recherche Conceptuelle d'Information Modèle d'Indexation Mixte : concepts-mots, 2007.

[15] : **Lobna HLAOUA,** Reformulation de Requêtes par Réinjection de Pertinence dans les Documents Semi-Structurés.

[16] : **Antonio Balvet**, Filtrage d'information par analyse partielle grammaires locales, dictionnaires électronique et lexique- Grammaire pour la recherche d'information, Paris 2001.

[17] : **Madjid IHADJADENE**, les systèmes de recherche d'informations, ed. Hermes science, paris 2004.

[18] : **Lynda TAMINE,** thèse : le système de recherche d'information : reformulation de requêtes et apprentissage basés sur les algorithmes génétiques, Institut d'informatique, 1998.

[19] : **Farida ACHMOUKH**, thèse sur les modèles de langage pour la recherche d'information.

[20] : **Philipe LEFEVRE**, la recherche d'information, du texte intégral au thésaurus, ed. Hermes Science, paris 2004.

[21] : **Mohammed NAFI et Akli ABBAS**, mémoire de fin d'étude sur la Conception et la réalisation d'un système de recherche d'information basé sur le modèle booléen étendu, 2005.

[22] : **Josiane MOTHE** , thèse sur la recherche et l'exploration d'informations, découverte de connaissances pour l'accès à l'information.

[23] : **Fabien SCHWOB**, Mémoire de maîtrise sur la recherche d'informations, 2005.

[24] : **ABDELKRIM BOURAMOUL,** Recherche d'information contextuelle et sémantique sur le web, 2011.

[25] : **Jian-Yun Nie**, Le domaine de recherche d'information, Département d'informatique et recherche opérationnelle, Université de Montréal

[26] : **CLAUDE DELANNOY,**Programmer en Java, paris, 2008.

[27]: **Mark Dexter,** Eclipse And Java For Total Beginners Companion Tutorial Document, 2007.

[28]: **Craig Macdonald & Ben He,** Researching and Building IR applications using Terrier, University of Glasgow.

[29]: **Iadh Ounis, Gianni Amati, Vassilis Plachouras, Ben He, Craig Macdonald, and Douglas Johnson,** Terrier Information Retrieval Platform, University of Glasgow.

[30]: http://www.terrier.org

Annexes

Plate forme Terrier

Introduction

Terrier est l'abréviation de *TerabyteRetrieval* est un logiciel libre et open source classé dans la lignée des SRI, sa création fut par le département d'informatique de l'université de Glasgow de Royaumes-Unis en 2000. Dans le but de fournir une plateforme flexible pour le développement rapide des applications de recherche d'information. Terrier est un produit Open source écrit en Java, qui a été à mis à disposition du public général depuis novembre 2004 sous la licence de MPL. Il a été conçu pour fonctionner sur la plupart des systèmes d'exploitation actuels tels que Windows ou Linux.

Avec sa conception modulaire, il facilite son extension en cas de besoin, il propose aussi une grande palette de configuration et cela dans le but de satisfaire ses utilisateurs et leur fournir une meilleure maniabilité. Ce système est souvant utilisé dans l'innovation et la mise en œuvre de nouvelles méthodes pour la RI.

1. Installation Terrier sous Windows

Pour pouvoir utiliser terrier version 3.5 sous Windows, il suffit simplement d'extraire (décompresser) le contenu du fichier ***terrier-x.x.zip*** téléchargé à partir du site de la plate-forme de Terrier : http://www.terrier.org .

Dans le dossier terrier décompressé, on trouve un ensemble de répertoires structurés comme suit :

- *bin* : ensemble des scripts pour exécuter terrier ;
- *doc* : la documentation relative à terrier ;
- *etc* : les fichiers de configuration de terrier,
- *lib* : les classes compilés de terrier et les différentes librairies externes utilisées par terrier ;
- *share* : la liste des mots vides (stopword-list.txt) et des exemples des documents à tester sur terrier ;
- *scr* : code source de terrier ;
- *var/index* : les structures de données après indexation (fichier inverse, fichier lexicon, index direct, index documents)
- var /result : résultats de la recherche ;
- licenses : les informations sur la licence des différents composants inclus dans terrier.

Ces différents répertoires sont illustrés dans la figure suivante:

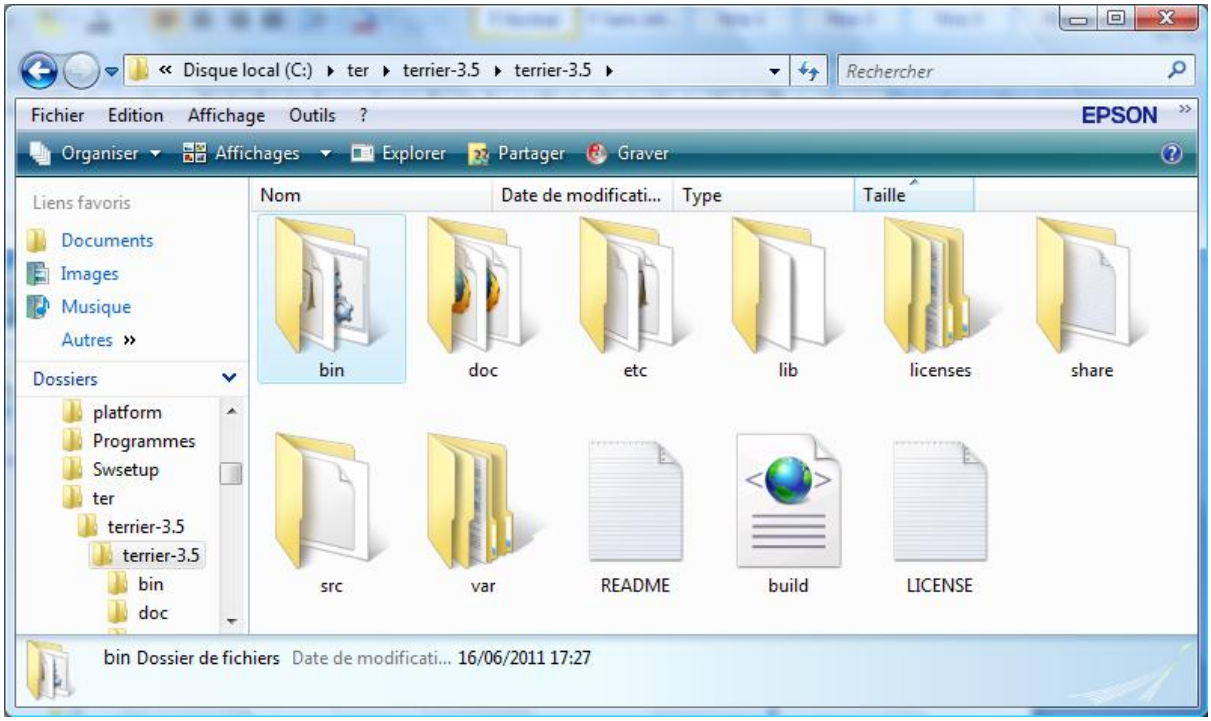


Figure 1 : Structure de Terrier 3.5

L'utilisation de Terrier requiert la version Java 1.6 ou supérieure. Pour cela, il suffit de vérifier si l'exécution des commandes *java* et *javac* dans l'invité de commandes sont reconnus. Si ce n'est pas le cas, installer Java et ajouter une nouvelle variable d'environnement utilisateur *PATH* qui prend comme valeur le chemin vers le dossier *bin* du Java que vous avez installé.

2. Architecture de Terrier

La figure montre l'architecture générale deTerrier

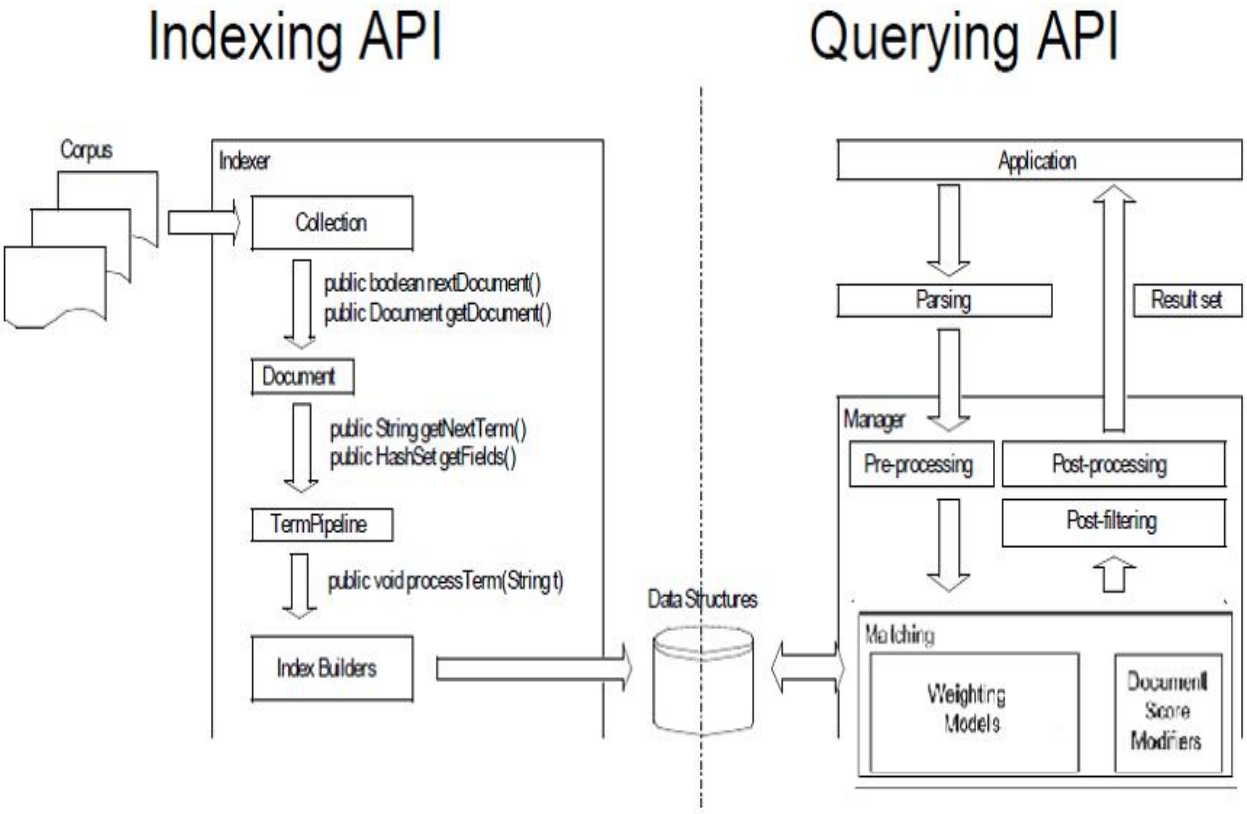


Figure 2 : Vue d'ensemble de l'architecture Terrier.

Indexing et Querying APIs (Application Programming Interface) est facile à étendre, adapte de nouvelles applications, dispose d'une architecture modulaire, facile pour commencer à travailler avec, elle offre beaucoup d'options de configuration.

Nous pouvons différentier les principales étapes qui caractérisent le système terrier :

2.1. L'indexation avec terrier

La figure ci-dessous donne une vue d'ensemble d'interaction des composants principaux impliqués dans le processus d'indexation.

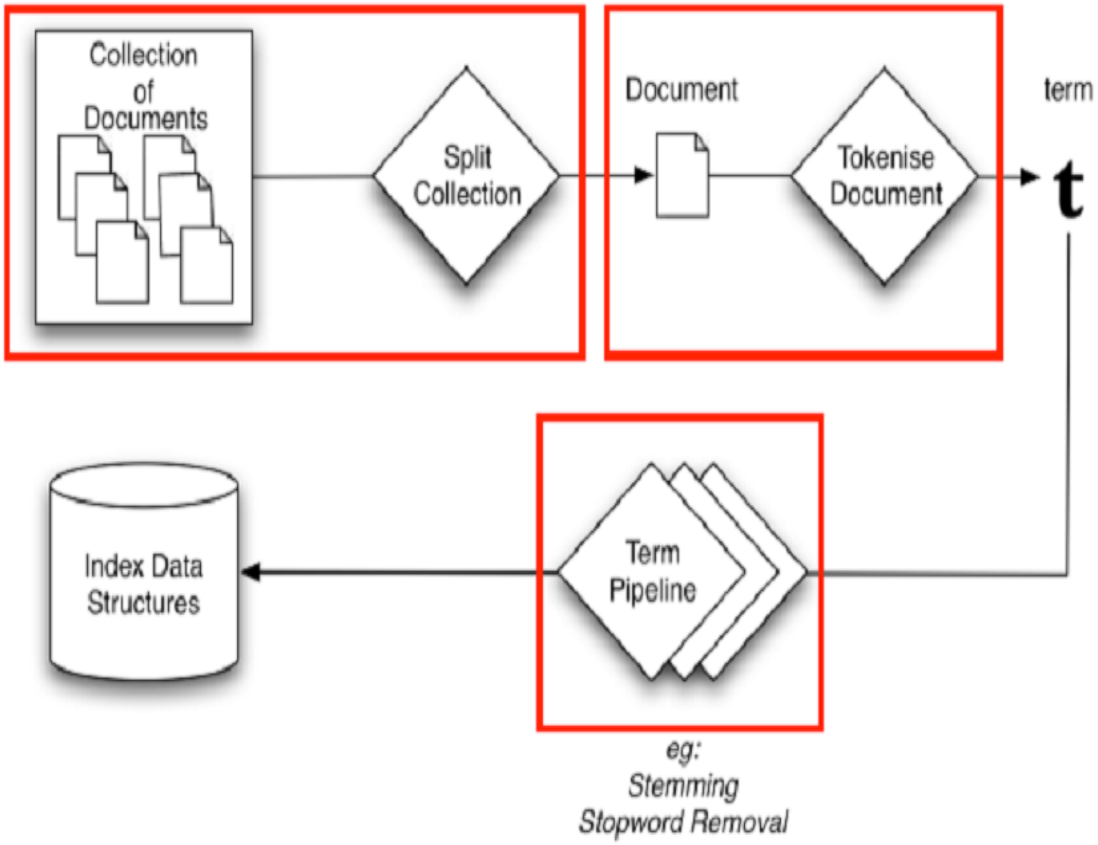


Figure 3: le processus d'Indexation dans Terrier.

Le processus d'indexation avec terrier consiste à analyser tous les documents de la collection spécifiée afin de produire un ensemble de termes d'indexations « index » et de les ranger dans la structure « index data structure ».

D'après la figure, on peut distinguer les unités de traitements suivantes :

2.1.1. Traitement sur la collection

Après la sélection de la collection sur laquelle on fait la recherche, cette collection sera ensuite déviée en documents particuliers pour leur effectuer un autre traitement.

2.1.2. Traitement sur les documents

Pour chaque document de la collection, un processus visant à extraire les termes du document nommé « tokenisation » :

- ✓ ***Tokenisation :***

Le processus de tokenisation fait l'analyse lexicale des documents, il reconnait les mots dans le texte, comme il offre des possibilités de traitement sur les chiffres, les traits d'union, les signes de ponctuation….etc.

Ce processus retourne en résultat l'ensemble de termes de document qu'on peut adopter comme termes d'indexation.

2.1.3. Traitement sur les termes

Tous les termes (token) obtenus de l'étape précédente (Tokenisation) seront traités par un processus dit *Term-Pepline* qui lui-même se subdivise en deux sous-processus.

- **Term-Pepline :**

- ✓ *Elimination des mots vides (stopword-removal) :* Ce processus permet de supprimer les mots vides dans les documents tels que les pronoms, les connecteurs, les espaces, …etc. Le but est d'améliorer le résultat d'indexation.
- ✓ *La normalisation (stemming) :* Ce processus permet de retrouver les différentes forme d'un mot et les représenter sous une forme unique, dont le but est d'obtenir une forme suffisante à la représentation des différentes apparitions des autres formes.

  Terrier utilise plusieurs méthodes de normalisation comme différentes variantes de l'algorithme de porter…

  Après les trois étapes successives précédentes, une étape de construction des structures d'index vient pour compléter le processus d'indexation.
- ✓ *Les structures de l'indexe :*

Après l'indexation, les termes sont stockés en structures de données suivantes:

- ✓ Lexicon : qui stocke les informations de chaque terme de la collection ;
- ✓ Inverted Index : où le fichier inverse est stocké ;
- ✓ Document Index : qui contient des informations sur les différents documents de la collection ;
- ✓ …etc.

- **Index direct :**

L'index direct enregistre pour un document les termes qui apparaissent dans ce dernier. Il est souvent utilisé pour la reformulation de la requête, la classification et comparaison des documents.

- **Index inversé :**

L'index inversé contrairement à l'index direct enregistre pour un terme les documents dans lesquels il apparait, il contient aussi la position de chaque terme et sa fréquence dans ces documents.

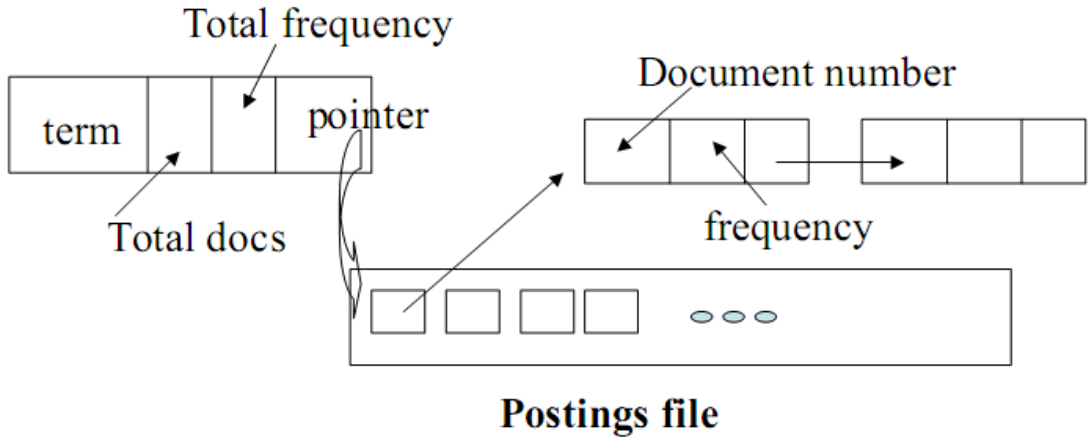


Figure 4 : composants de l'index inversé.

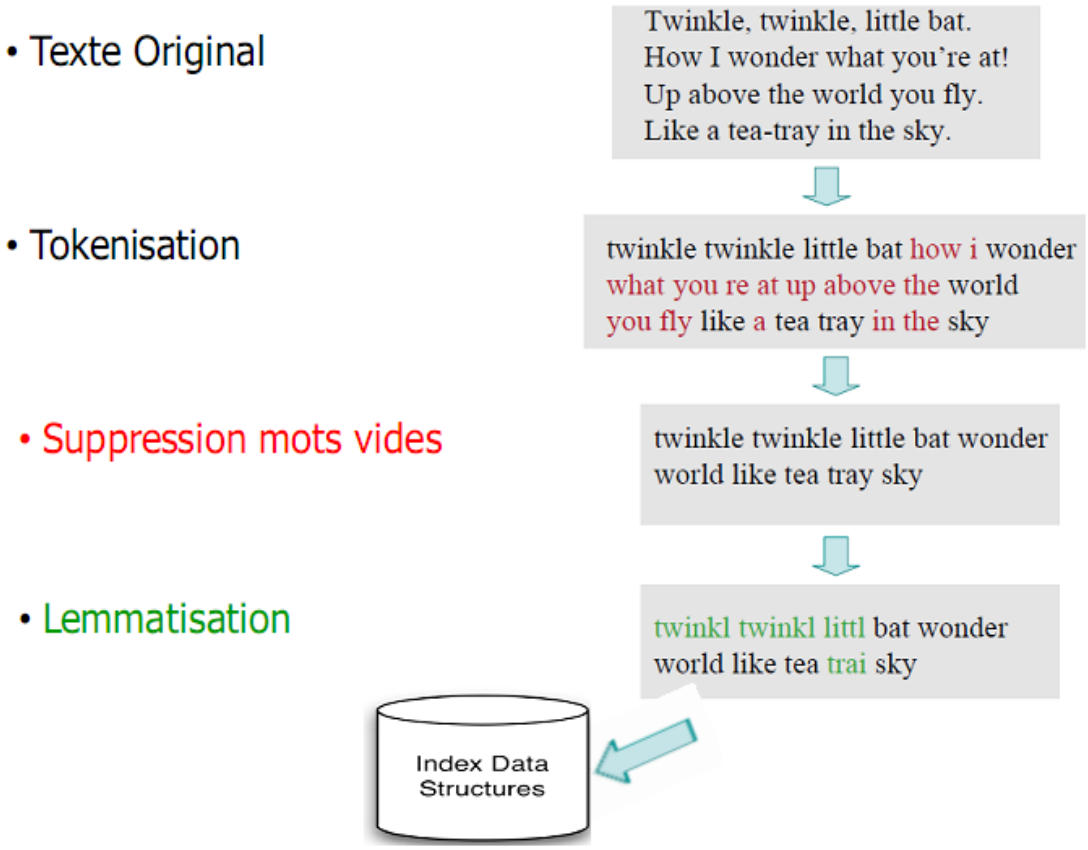


Figure 5 : Exemple d'indexation [28].

Pour construire ces fichiers, terrier dispose de deux méthodes :

A. Méthodes de deux phases

Elle s'effectue comme suit :

- Cette méthode commence par l'enregistrement des identificateurs pour chaque terme du document :

  -Ecrire l'index direct pour chaque document.

- Inversion :

  -Construire les posting-list pour chaque terme du document dans la mémoire.

  -Inverser ces listes une fois qu'on connait pour chaque terme les documents auxquels il appartient.

  Cette méthode est très couteuse en mémoire.

B. Méthode à une seule phase

Elle s'effectue comme suit :

- Analyser la collection, construire l'index inversé dans la mémoire

-En cas de saturation de la mémoire enregistrer dans le disque

- Fusionner les fichiers inversés crées.

2.2. La recherche avec terrier

Un des principaux objectifs de terrier est de faciliter la recherche d'information. Terrier implémente pour cela certain nombre de fonctionnalités de recherche qui offre un large choix pour le développement de nouvelles applications et pour les tests en RI. En effet, Terrier offre un grand choix de modèles pondération [], il propose aussi un langage de requêtes avancé []. Une autre fonction de recherche très importante intégrée dans Terrier est l'automatisation de l'expansion de requêtes.

Terrier utilise trois composants principaux pour la recherche : Query, Manager, Matching qui seront décrit ci-dessous :

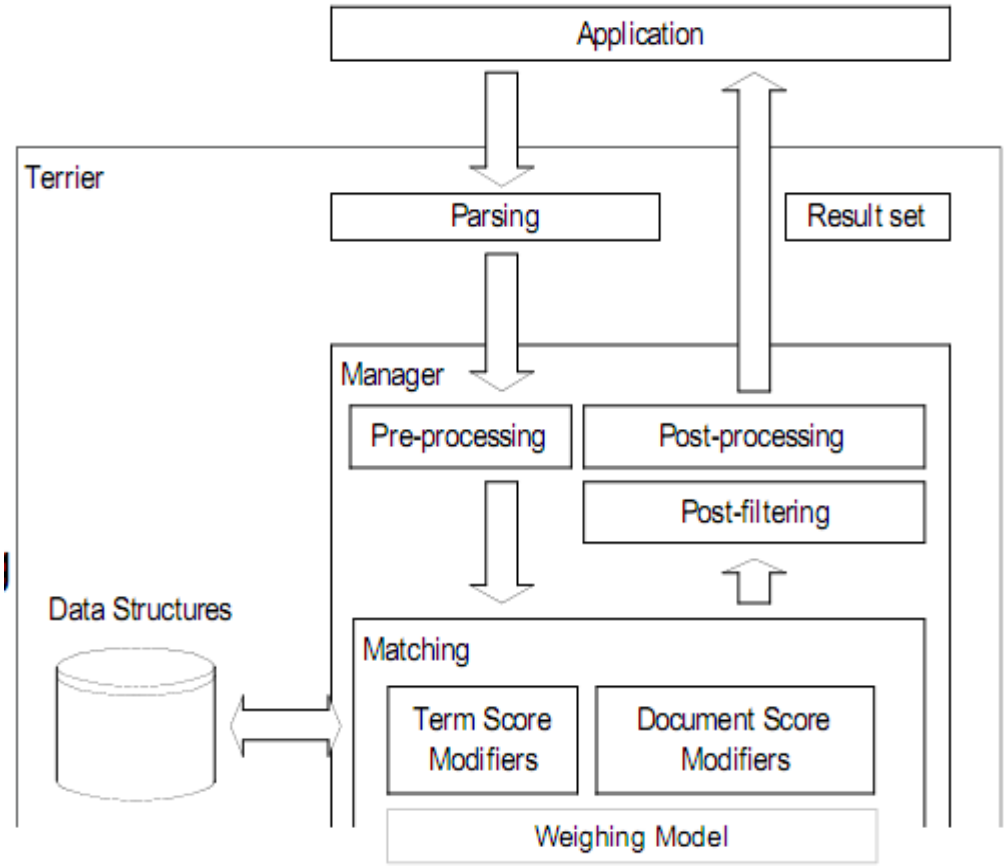


Figure 6 : Présentation du processus de recherche de Terrier.

- **Query :**

C'est l'entrée que l'application offre à Terrier. Le module Matching est responsable de déterminer quel document correspond à une requête spécifique et attribut un score au document qui respecte la requête. Query est une classe abstraite, qui se trouve dans le package uk.ac.gla.terrier.querying.parseret qui modélise les requêtes. Elle consiste en un sub-queries ou bien en un queryterms. Un objet Query est créé pour chaque requête.

- **Manager :**

C'est le modèle qui est chargé de la gestion de la recherche. Dans un premier niveau il lit chaque requête en utilisant le parseur approprié. Puis il crée un second niveau de gestion associé à chaque requête en récupérant l'ensemble des paramètres nécessaires et en spécifiant un modèle de pondération. Puis il crée un fichier résultat (result file) où il associe à chaque requête les documents qui lui sont pertinents. Le résultat de chaque requête est donné par le second niveau de gestion.

Le second niveau de gestion est responsable de l'ordonnancement et de la coordination des opérations principales de haut niveau sur une seule requête. Ces opérations sont : Pre-processing, Matching, Post-processing, Post-filtring.

- **Matching:**

Le composant Matching est responsable de déterminer les documents correspondant à la requête spécifiée et donne un score au document qui vérifie la requête. Il utilise l'interface WeightingModels pour assigner un score à chaque mot de la requête dans le document.

Terrier offre plusieurs classes qui implémentent cette interface parmi eux (TF-IDF, BM25).

✓ *Document Score Modifiers* : Modifie le score des documents en fonction des propriétés du document.

✓ *Term Score Modifiers :* Modifie le score des documents en fonction de la position des termes.

- **Post-processing :**

Après l'application de TermScoreModifers ou DocumentScoreModifers, l'ensemble de documents recherchés (retourné par l'opération de Matching) sera modifié en appliquant le Post-processing ou le Post-filtering.

Le Post-processing est approprié pour implémenter des fonctionnalités, qui apportent un changement à la requête originale. Un exemple de Post-processing est l'expansion de requête(EQ), puis exécute une autre fois le Matching avec cette nouvelle requête. Un autre exemple possible de Post-processing est l'application de clustering.

- **Post-filtring :**

Le Post-filtring est l'étape finale du processus de recherche de Terrier, où une série de filtres peut enlever les documents déjà recherchés, qui ne satisfont pas une condition donnée, Par exemple, dans le contexte d'un moteur de recherche Web, un Post filtre peut être utilisé pour réduire le nombre de documents recherchés depuis le même site Web, afin d'augmenter la diversité dans les résultats.

A la suite des étapes précédentes Terrier s'occupe de retourner un ensemble de documents triés selon l'ordre décroissant de leurs scores.

3. Utilisation de Terrier : « TREC Terrier »

Dans cette section, nous décrivons l'utilisation de TREC Terrier pour l'indexation, la recherche et l'évaluation sur le système d'exploitation Windows XP.

Indexation :

Pour indexer des documents avec Terrier, on peut utiliser deux méthodes :

- Avec l'invite de commande :
  - Spécifier le chemin vers Terrier : cd <le path vers Terrier>\bin
  - Spécifier la collection de documents à indexer : trec_setup<le path de la collection à indexer>. Si on ouvre le fichier collection.spec on s'aperçoit que Terrier a mis dans ce dernier tous les paths vers les fichiers contenus dans le répertoire donné automatiquement.

- L'indexation est lancée par la commande :
  - **« trec_terrier –i »**Pour une indexation classique two_pass ;
  - **« trec_terrier -i –j»** Pour une indexation classique Single_pass (Direct Index non construit : Gain d'espaces mémoire). Utilisé pour les collections complexes et volumineuses.

Remarque : pour indexer une collection autre qu'une Collection de TREC, il est nécessaire d'apporter quelques modifications au fichier terrier.properties.

S'il n'y a pas eu d'erreurs, on trouvera dans var\index les fichiers résultants de l'indexation

➢ Avec desktop_terrier (qui se trouve dans le répertoire bin). L'interface associée permet de sélectionner les documents à indexer puis à lancer l'indexation effective.

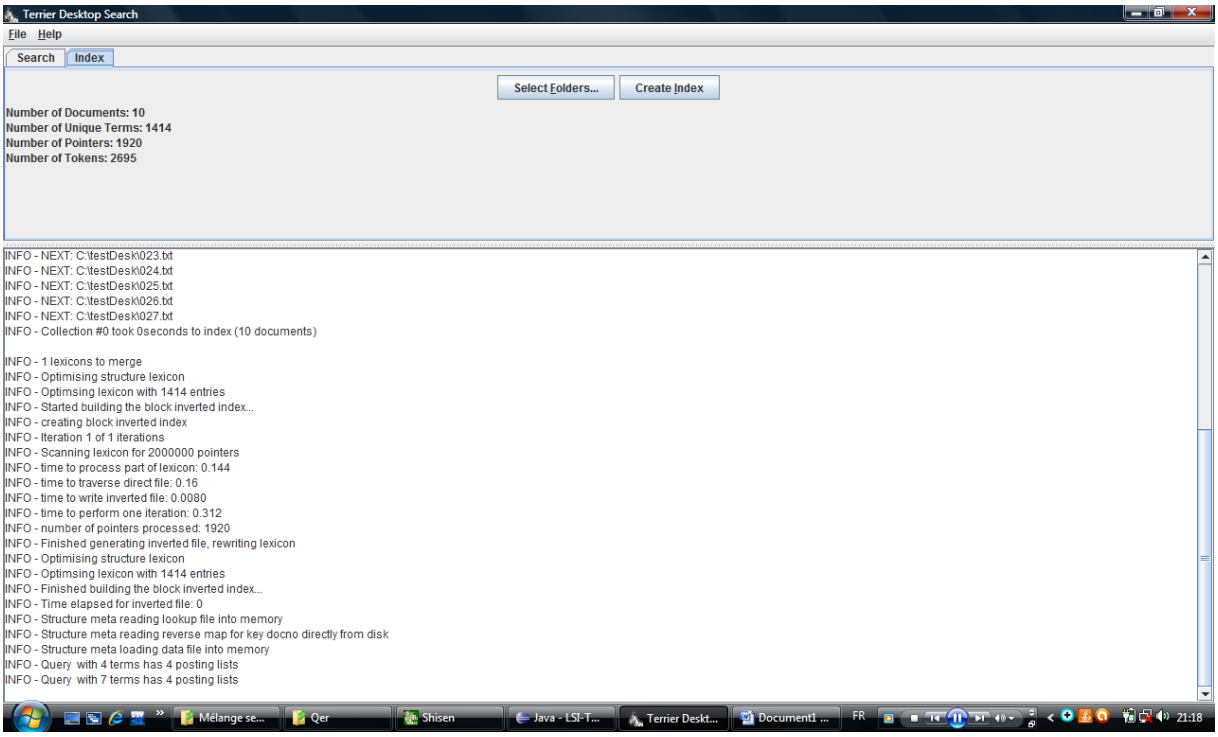


Figure 7: Interface de desktop_terrier pour l'indexation.

Recherche :

- Pour faire une recherche sur une collection de documents déjà indexés, on utilise la commande **« trec_terrier –r »**. mais avant, on doit :

  - Donner à Terrier les requêtes modifiant etc\terrier.properties comme suit : trec.topics=<le path vers les requêtes>.
  - Spécifier le modèle de pondération (ex : TF_IDF) à utiliser dans le fichier etc\trec.models.

- Les résultats de la recherche sont stockés dans :
  « var\result\nom_fonction_pondération.res »
- Notons que l'on peut aussi faire une recherche avec l'interface de Terrier (desktop_terrier).

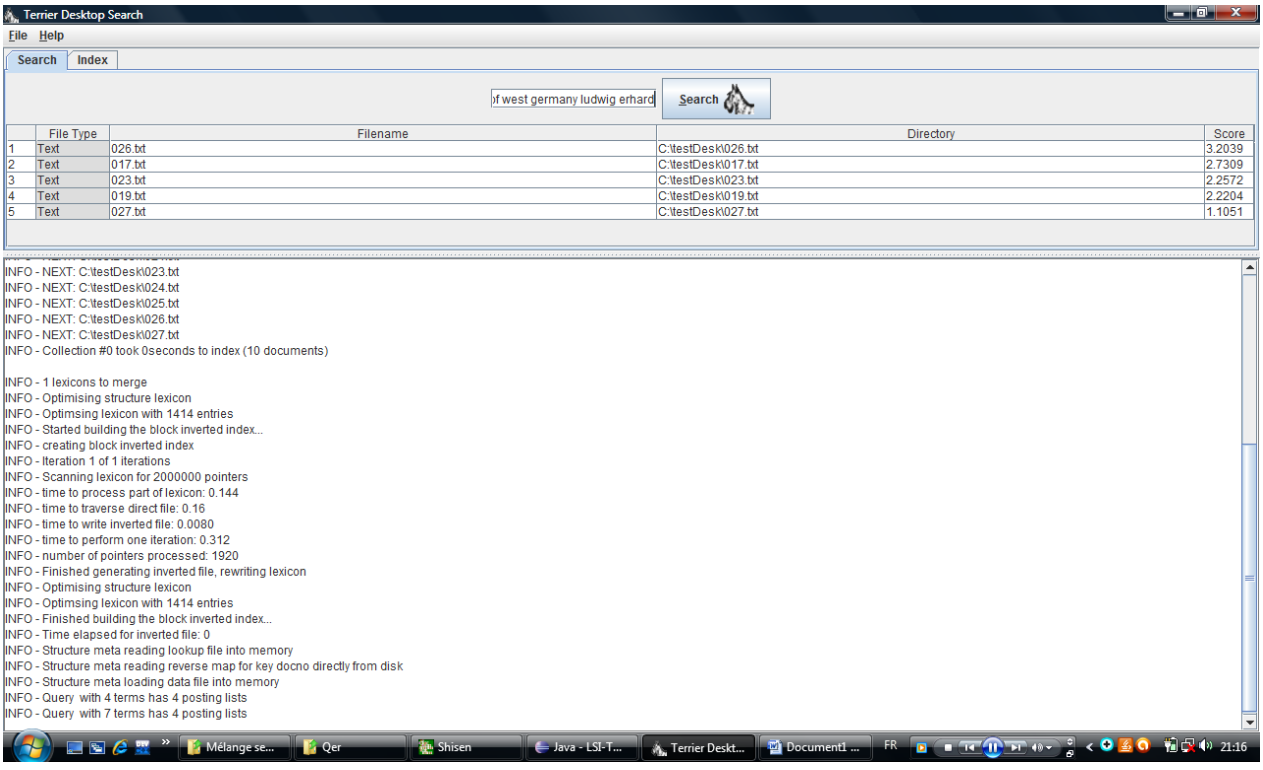


Figure 8: Interface de desktop_terrier pour la recherche.

Reformulation :

- Pour faire une reformulation on utilise la commande **« trec_terrier -q –r »**

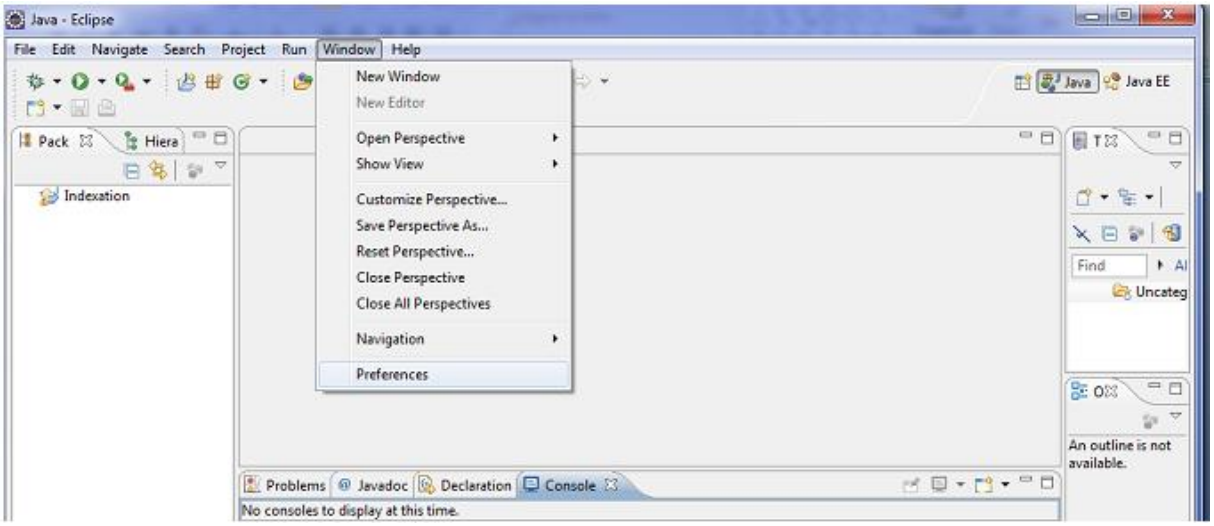
Evaluation :

- Pour effectuer une évaluation, il faut spécifier à Terrier le chemin vers les jugements de pertinence (ou Qrels) de la collection, en spécifiant trec.qrels=<le path vers le fichier qrels de la collection> dans le fichier terrier.propreties.
- La commande d'évaluation effective est : trec_terrier –e.
- Les résultats de l'évaluation se trouvent dans :
  « var\result\nom_fonction_pondération.eval ».

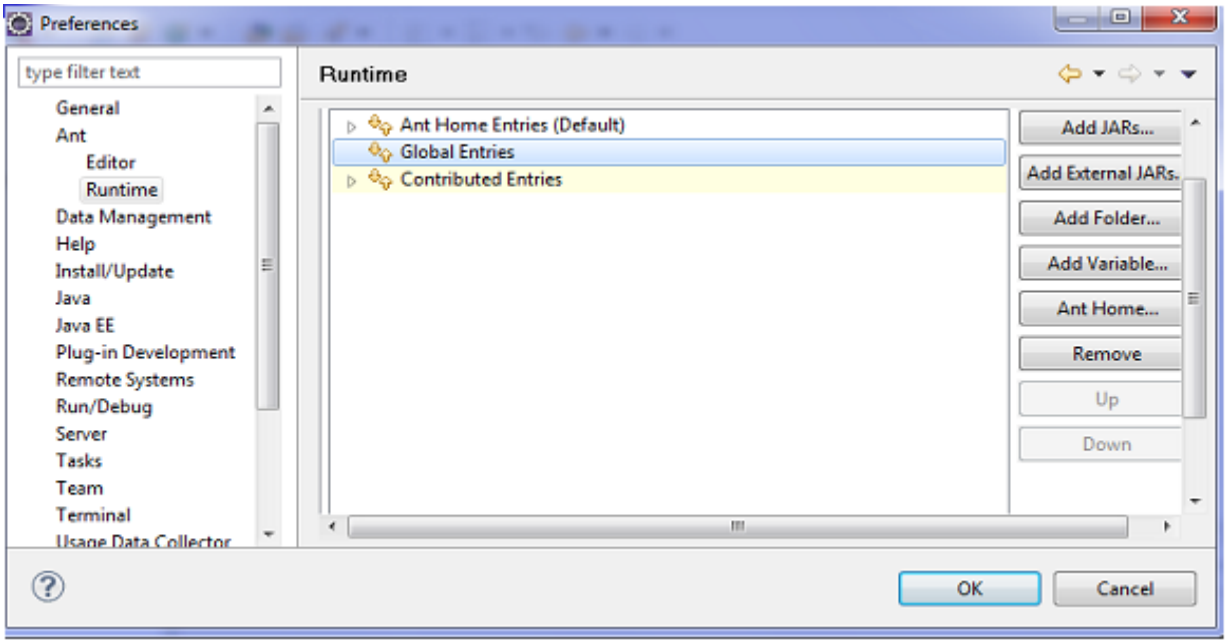
4. Compilation de terrier :

Pour compiler Terrier, utiliser l'Ant de l'environnement IDE Eclipse et le fichier build.xml de terrier 3.5.

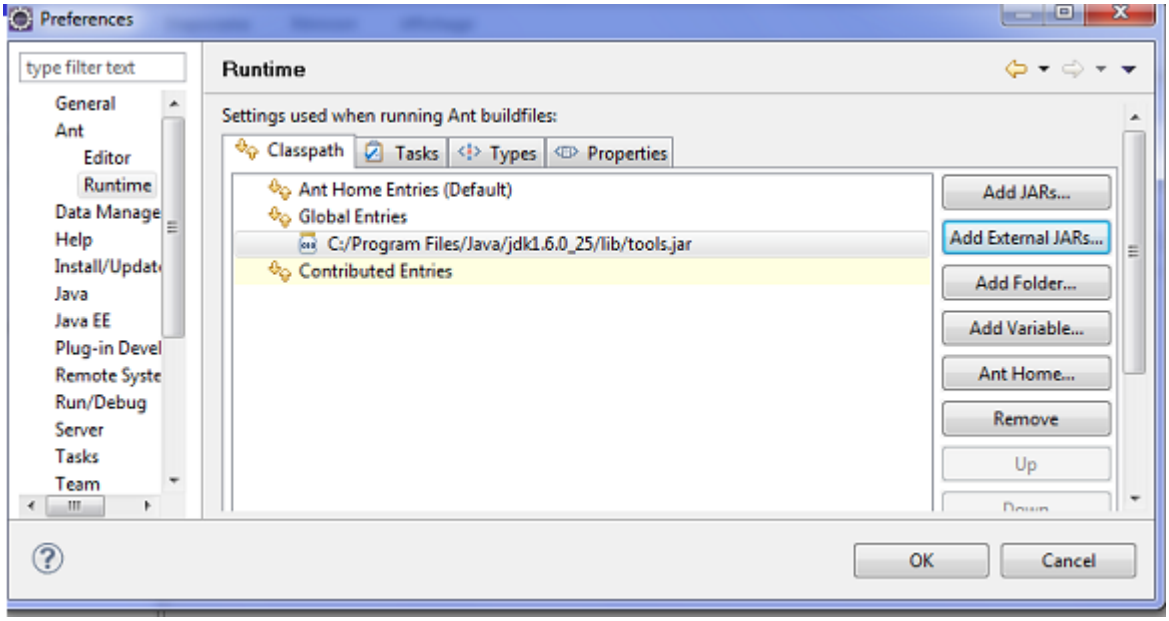
Configuration Ant de l'IDE Eclipse :



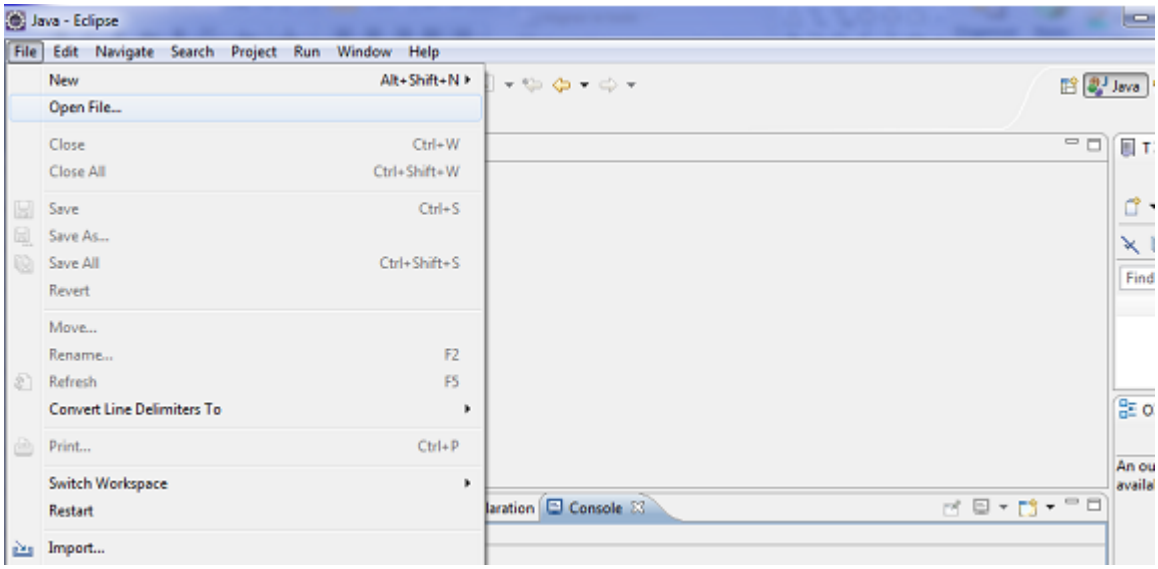
Configuration Ant : 1- Window->Preferences



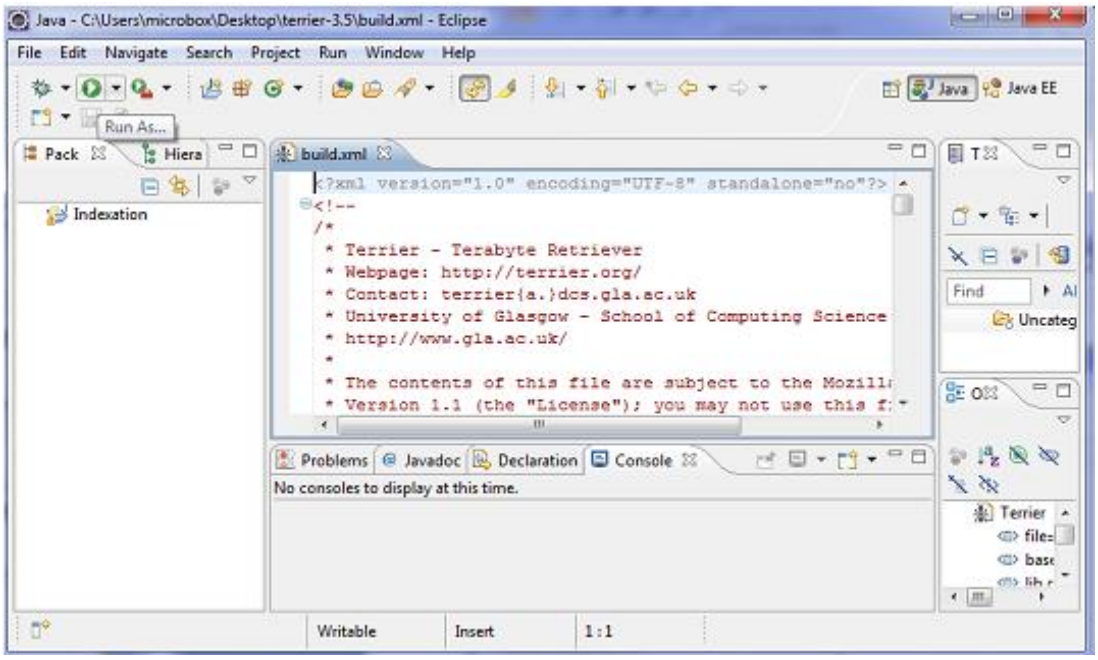
Configuration Ant : 2- Ant ->Runtime->Global Entries



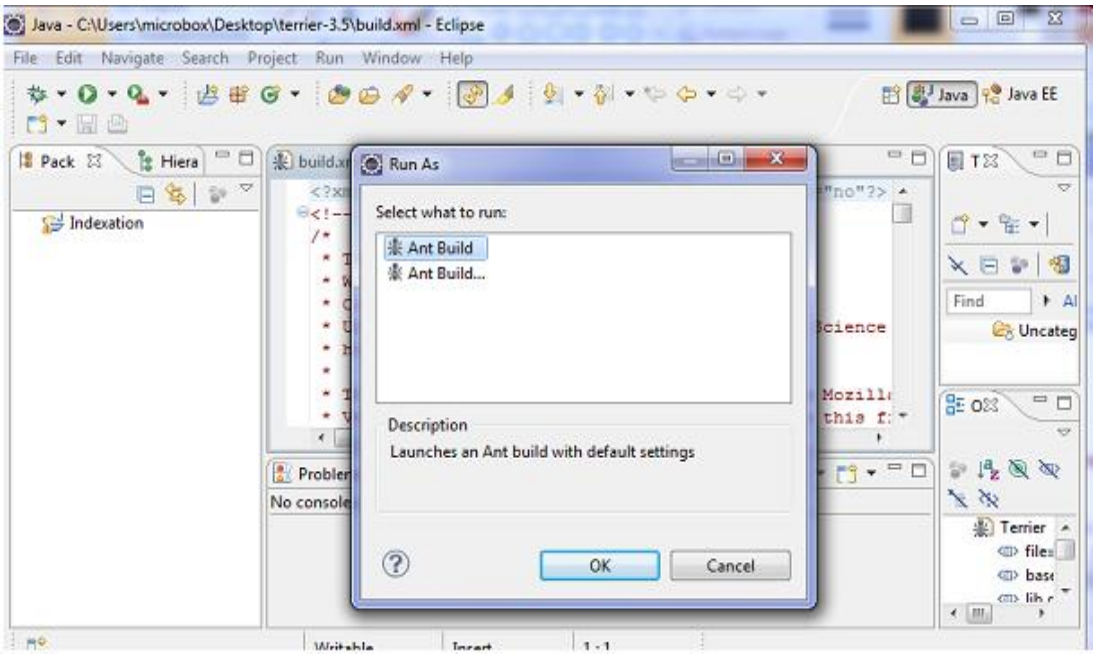
Configuration Ant : 3- Ajouter avec add Externals JARS le fichier ../Java/ jdk1.6.0_25/ lib/tools.jar



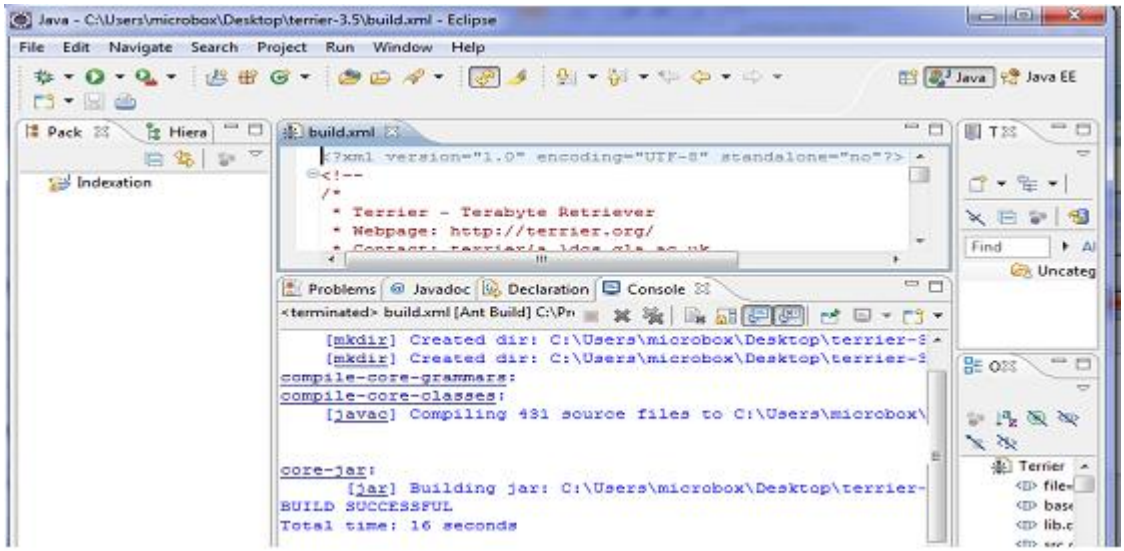
Ouvrir le fichier build.xml de terrier



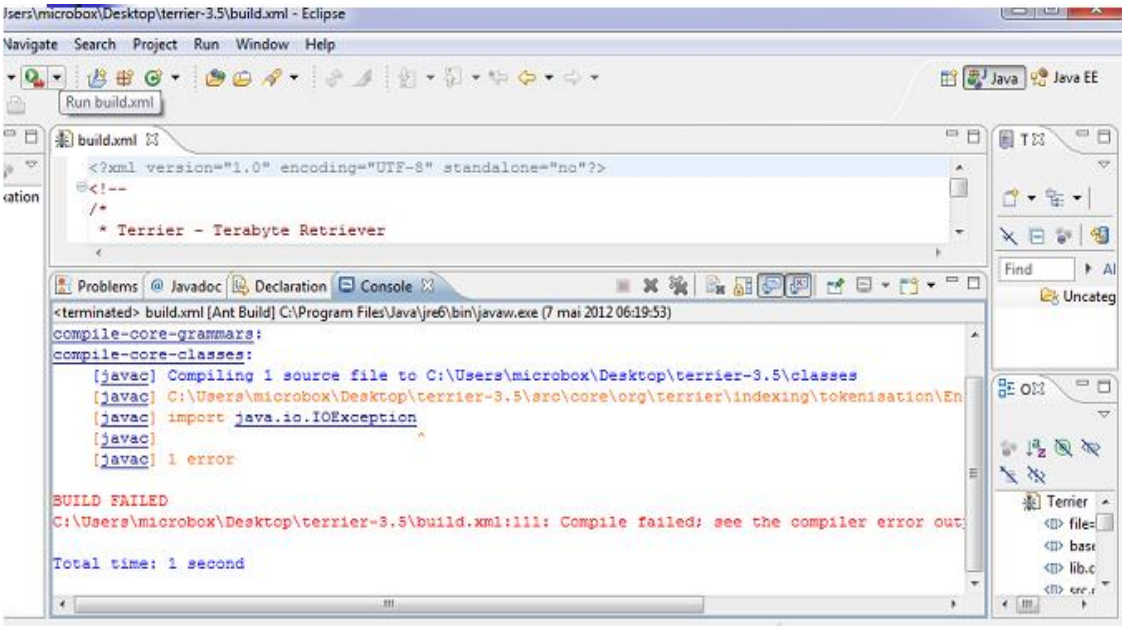
Exécuter le script: Bouton Run



Exécuter Ant Build



BUILD SUCCESSFUL: Compilation réussie



BUILD FAILED: Compilation échouée

5. Les possibilités d'extension de terrier :

Ce logiciel offre différentes possibilités d'extension que se soit sur ses processus d'indexation, dans ses modèles de pondération ou dans ses étapes de reformulation de la requête. Ces extensions peuvent aussi survenir dans le cas d'intégration de nouvelles collections de documents spéciales ou personnelles qui peuvent servir pour l'évaluation.