Modèle flexible pour la Recherche d'Information dans des corpus de documents semi-structurés

Karen Sauvagnat

THÈSE

Présentée devant

l' Université Paul Sabatier de Toulouse

en vue de l'obtention du

Doctorat de l'Université Paul Sabatier

Spécialité : **INFORMATIQUE**

*P*ar

Karen SAUVAGNAT

Modèle flexible pour la Recherche d'Information dans des corpus de documents semi-structurés

---

Soutenue le 30 Juin 2005, devant le jury composé de :

M. M. BOUGHANEM	Professeur à l'Université Paul Sabatier, Toulouse III	Directeur de thèse
M. C. CHRISMENT	Professeur à l'Université Paul Sabatier, Toulouse III	Directeur de thèse
M. P. GALLINARI	Professeur à l'Université Pierre et Marie Curie, Paris VI	Rapporteur
M. J.-M. PINON	Professeur à l'INSA de Lyon	Examinateur
M. J. SAVOY	Professeur à l'Université de Neuchâtel, Suisse	Rapporteur
M. G. ZURFLUH	Professeur à l'Université de Toulouse I	Examinateur

INSTITUT DE RECHERCHE EN INFORMATIQUE DE TOULOUSE
Centre National de la Recherche Scientifique - Institut National Polytechnique - Université Paul Sabatier
Université Paul Sabatier, 118 Route de Narbonne, 31062 Toulouse Cedex 04. Tel : 05.61.55.66.11

Résumé

La nature de sources d'information évolue, et les documents numériques traditionnels " plats " ne contenant que du texte s'enrichissent d'information structurelle et multimédia. Cette évolution est accélérée par l'expansion du Web, et les documents semi-structurés de type XML (eXtensible Markup Language) tendent à former la majorité des documents numériques mis à disposition des utilisateurs. Le développement d'outils automatisés permettant un accès efficace à ce nouveau type d'information numérique apparaît comme une nécessité. Afin de valoriser au mieux l'ensemble des informations disponibles, les méthodes existantes de Recherche d'Information (RI) doivent être adaptées. L'information structurelle des documents peut en effet servir à affiner le concept de granule documentaire. Le but pour les Systèmes de Recherche d'Information (SRI) est alors de retrouver des unités d'information (et non plus de documents) pertinentes à des requêtes utilisateur. Afin de répondre à cette problématique fondamentale, de nouveaux modèles prenant en compte l'information structurelle des documents, tant au niveau de l'indexation, de l'interrogation que de la recherche doivent être construits.

L'objectif de nos travaux est de proposer un modèle permettant d'effectuer des recherches flexibles dans des corpus de document semi-structurés. Ceci nous a conduit à proposer le modèle XFIRM (*XML Flexible Information Retrieval Model*) reposant sur : (i) Un modèle de représentation des données générique, permettant de modéliser des documents possédant des structures différentes ; (ii) Un langage de requête flexible, permettant à l'utilisateur d'exprimer son besoin selon divers degrés de précision, en exprimant ou non des conditions sur la structure des documents ; (iii) Un modèle de recherche basée sur une méthode de propagation de la pertinence. Ce modèle a pour but de trouver les unités d'information les plus *exhaustives* et *spécifiques* répondant à une requête utilisateur, que celle-ci contienne ou non des conditions de structure. Les documents semi-structurés peuvent être représentés sous forme arborescente, et le but est alors de trouver les sous-arbres de taille minimale répondant à la requête. Les recherches sur le contenu seul des documents sont effectuées en

prenant en compte les importances diverses des feuilles des sous-arbres, et en plaçant ces derniers dans leur contexte, c'est à dire, en tenant compte de la pertinence du document. Les recherches portant à la fois sur le contenu et la structure des documents sont effectuées grâce à plusieurs propagations de pertinence dans l'arbre du document, et ce afin d'effectuer une correspondance vague entre l'arbre du document et l'arbre de la requête.
L'évaluation de notre modèle, grâce au prototype que nous avons développé, montre l'intérêt de nos propositions, que ce soit pour effectuer des recherches sur le contenu seul des documents que sur le contenu et la structure.

Mots-clés : Recherche d'Information, documents semi-structurés, XML, propagation de la pertinence

Remerciements

Octobre 2002, arrivée à Toulouse et à l'IRIT. Novembre 2002, premier TP de l'autre côté du miroir, porte refermée sur 30 paires d'yeux braqués sur moi. Premier papier. Première conférence. Première présentation en anglais. Premier amphi... Que d'ongles rongés avant toutes ces expériences, mais surtout que de chemin parcouru depuis mon arrivée à Toulouse ! Ces trois années de thèse ont été riches de rencontres, de joies et de travail.
C'est avec un immense plaisir que j'écris aujourd'hui ces lignes (qui ont longtemps représenté la lumière au but du chemin ténébreux de la rédaction) et que je remercie toutes les personnes qui ont fait que ce travail de thèse arrive à son terme dans les meilleures conditions.

Je tiens à remercier très sincèrement Messieurs les Professeurs Claude Chrisment et Gilles Zurfluh, responsables de l'équipe SIG, pour m'avoir accueillie au sein de leur équipe.
Cette thèse repose sur une collaboration CIFRE entre l'IRIT et la société Coelis. Que Monsieur Jérôme Thil, fondateur de Coelis, soit assuré de mes remerciements sincères pour avoir supporté mes travaux.

Je tiens à exprimer ma profonde gratitude à Monsieur Mohand Boughanem, Professeur à l'Université Paul Sabatier, pour avoir encadré et dirigé mes recherches. Je le remercie pour toute la confiance qu'il a su me porter, et pour la patience, la gentillesse et la disponibilité dont il a fait preuve à mon égard. Ses conseils et remarques constructives m'ont permis d'améliorer grandement la qualité de mes travaux et de ce mémoire.
Je remercie Monsieur Claude Chrisment, Professeur à l'Université Paul Sabatier pour avoir dirigé mes recherches. Je le remercie également pour toutes les discussions que nous avons eues, desquelles ont découlés de nombreux conseils et remarques constructives. Il peut être assuré de mon sincère respect et de ma profonde gratitude.
Un très grand merci à mes deux rapporteurs, dont la lecture approfondie de ce

mémoire a permis d'en améliorer la qualité : Monsieur Jacques Savoy, Professeur de l'Université de Neuchâtel, et Monsieur Patrick Gallinari, Professeur à l'Université Pierre et Marie Curie de Paris.
Je tiens également à remercier Monsieur Jean-Marie Pinon, Professeur à l'Institut National des Sciences Appliquées de Lyon, pour l'honneur qu'il me fait en participant à ce jury, trois ans après m'avoir remis mon diplôme d'ingénieur.
Je remercie Monsieur Gilles Zurfluh, Professeur à l'Université Toulouse I, pour l'intérêt qu'il a porté à mes travaux en examinant ce mémoire et pour l'honneur qu'il me fait en participant à ce jury.

Je tiens également à remercier tous les membres permanents de l'équipe SIG pour leur aide et leur gentillesse. Il faudrait dire un mot sur tous, mais ce serait trop long... Un merci plus particulier à Lynda, pour l'attention qu'elle a portée à la lecture de ce mémoire et pour les remarques pertinentes qui en ont découlées, et à Cécile pour son aide précieuse à mes débuts dans l'équipe.
Merci aussi au personnel du laboratoire (Agathe, Annie, Jean-Claude, Jean-Pierre, Pierre...) pour sa gentillesse et pour son aide lorsque la lumière a disparu ou que la malédiction frappait les ordinateurs que j'approchais.

Quant à tous les stagiaires et thésards qui ont partagés toutes ces heures de joies, de découragement et de salle machine, sachez que ce fut un plaisir pour moi de vous connaître et de passer ces heures avec vous.
Merci aussi à tous les thésards ou anciens thésards, compagnons de galères rencontrées au gré d'école d'été et de conférences, et avec qui j'attends avec impatience de repasser des soirées. Je pense notamment à Haïfa et Ludovic.
Merci aussi tout particulièrement à Saïd et Benoît, pour tous ces moments passés ensemble, pour les attaques à la dame et les brasses qui un jour peut-être se transformeront en papillon.
Asma et Nathalie. Je vous assure les filles, que ces lignes sont les plus dures à écrire, tellement il y a de choses que je voudrais vous dire. J'ai trouvé avec vous un trésor infiniment précieux, que je ferai tout pour conserver.

Je ne peux enfin clôturer ces remerciements sans remercier du fond du coeur mes parents, qui n'ont eu de cesse de me soutenir et de croire en moi pendant ces looooooooongues études.
Merci aussi à Chantal, Robert et Carine, qui ont fait que Toulouse soit aussi quelque part un petit bout de Clermont.
J'arrive au bout de ces remerciements, je vous assure.
Laurent ? MERCI.
Tranquille. Enfin je te promets que j'essaie.

Table des matières

Liste des tableaux

Table des figures

Introduction générale

L'Homme étend aujourd'hui ses activités à un nombre croissant de secteurs. Alors qu'au siècle des Lumières les savants pouvaient se vanter de regrouper toute la connaissance humaine connue dans leur fameuse Encyclopédie, cette démarche, si on voulait la reconduire aujourd'hui, ne serait plus qu'une utopie. L'augmentation quasi-exponentielle des connaissances de l'Homme ainsi que leur spécialisation, inévitable, dans des domaines d'intérêt très variés, conduit à la production d'un volume d'information sans précédent. Le nombre d'e-mails envoyés chaque année représente par exemple 400 000 teraoctets d'information, et la quantité totale d'information produite en 2002 avoisinerait les 5 exaoctets (un million de teraoctets) [210].
L'apparition et la popularisation des ordinateurs, des documents électroniques, de différents types de support pour stocker les documents et des réseaux de télécommunication ont profondément bouleversé les liens entre l'Homme et l'information. Les mémoires magnétiques et optiques ont permis un stockage des documents électroniques de très bonne qualité, pour un coût qui ne cesse de diminuer depuis leur création. Les réseaux de télécommunication permettent leur diffusion et un échange rapide et plus simple que jamais. Un exemple de cette révolution est la récente annonce de la firme Google, qui gère le moteur de recherche actuellement le plus utilisé sur Internet, de créer la plus grande bibliothèque en ligne ayant jamais existé, et ce en numérisant 15 millions d'ouvrages. Le développement d'outils automatisés permettant un accès efficace à cette quantité gigantesque d'information numérique apparait comme une nécessité.

Contexte de travail

Notre travail se situe dans le contexte de la *recherche d'information* (RI). L'objectif principal des Systèmes de Recherche d'Information (SRI) est de répondre au besoin en information des utilisateurs. Les utilisateurs interrogent, au moyen d'une requête, une base de documents numériques et les SRI leur renvoient une liste de documents susceptibles de répondre à leur besoin.
Aujourd'hui cependant, la nature des sources d'information évolue, et les docu-

ments traditionnels "plats" ne contenant que du texte s'enrichissent d'information structurelle et d'information multimédia. Cette évolution est accélérée par l'expansion du Web. De ce fait, les documents structurés ou semi-structurés de type HTML (*HyperText Markup Language*) ou XML (*eXtensible Markup Language*) tendent à former la majorité des documents numériques mis à disposition des utilisateurs.
Afin valoriser au mieux l'ensemble des informations disponibles, les méthodes existantes de recherche d'information doivent être adaptées ou de nouvelles méthodes doivent être proposées. C'est dans ce contexte de *recherche d'information structurée* que se situent plus particulièrement nos travaux. Nous nous plaçons plus précisément dans le cadre de *documents semi-structurés*, c'est à dire de documents ne disposant pas d'une structure fixe et homogène, mais au contraire d'une structure flexible ainsi que de contenus hétérogènes. Nous utiliserons le format XML tout au long de ce mémoire pour illustrer nos propos.

Problématique

Quel que soit le type de documents que l'utilisateur interroge, ce dernier s'intéresse rarement à une représentation ou à une structuration précise des documents, il veut du contenu. S'il est capable de fournir des informations supplémentaires dans sa requête, par exemple des informations structurelles, la réponse fournie par le système ne devrait être que plus précise.
Les documents XML, par leur structure même, doivent permettre aux SRI de se focaliser sur l'information pertinente des documents. Les documents peuvent en effet souvent posséder des contenus hétérogènes, délimités grâce à de l'information structurelle. Cette information structurelle peut alors servir aux SRI à traiter l'information textuelle avec une autre granularité que le document tout entier. Leur but est alors de retrouver des *unités d'information* (et non plus des documents) pertinentes à une requête utilisateur. Ces unités d'information doivent se suffire à elles-mêmes pour répondre à la requête, et pourront être présentées telles quelles à l'utilisateur (on ne cherche pas à lui fournir un point d'entrée dans le document, mais au contraire à lui donner une unité d'information ne dépendant pas d'une autre pour être comprise).
Ces unités dépendent fortement des requêtes de l'utilisateur. Ces dernières peuvent être exprimées de deux manières différentes :

- Elles peuvent être composées de simples mots-clés, c'est à dire similaires aux requêtes utilisées pour l'interrogation des SRI traditionnels. On parlera alors de *requêtes orientées contenu*.
- Elles peuvent également comporter des conditions de structure, ainsi que des conditions de contenu sur ces éléments de structure. L'utilisateur peut alors s'il le souhaite spécifier le type des unités d'information qu'il

désire voir retournées. Ces requêtes sont aussi nommées *requêtes orientées contenu et structure.*

Le traitement de ces requêtes soulève trois questions :

- *comment organiser (indexer) les informations contenues dans les documents, afin de pouvoir ensuite les utiliser pendant la recherche ?*
- *quel formalisme l'utilisateur va-til pouvoir utiliser pour exprimer son besoin ?*
- *comment sélectionner des unités d'information pertinentes au besoin de l'utilisateur ?*

Les réponses à ces questions ont été abordées selon deux angles principaux [75], les approches orientées données et celles orientées documents :

- ***Les approches orientées données*** voient les documents XML comme une suite de données, typées et relativement homogènes. Dans ce cadre, la recherche dans les documents XML consiste à représenter de façon complète la structure des documents et à évaluer de façon *exacte* des expressions du type *attribut=valeur*. Ces approches utilisent des techniques développées par la communauté des Bases de Données.
- ***Les approches orientées documents*** se focalisent sur des applications considérant les documents structurés d'une manière traditionnelle, c'est à dire que les balises servent uniquement à décrire la structure logique des documents. Pour ces approches, la recherche consiste à évaluer la *pertinence* du contenu (textuel ou structurel) des documents vis-à-vis de la requête. Elles sont prises en charge par la communauté de la Recherche d'Information.

Notre problématique se situe dans le cadre de la Recherche d'Information textuelle. Dans ce contexte, les trois questions précédentes soulèvent plusieurs problématiques spécifiques, dont la plus importante concerne les notions de *pertinence* et de *tri*. Cette problématique, spécifique à la RI, est absente des approches orientées BD.

Plus précisément, la problématique dans le cadre de l'indexation se situe essentiellement au niveau de l'information structurelle. Dans le cas des documents textes "plats", le contenu textuel des documents est traité afin de trouver et de pondérer les termes les plus représentatifs des documents. Dans le cas des documents semi-structurés, la dimension structurelle s'ajoute au contenu, et les questions suivantes se posent alors : que doit-on indexer de la structure des documents ? Comment relier cette structure au contenu même du document ? En fonction de quelle dimension (niveau éléments, documents, collection) doit-on pondérer les termes d'indexation ? De nombreux schémas ont été proposés dans la littérature pour l'indexation des documents, mais la plupart sont orientés données : ces méthodes traitent efficacement l'information structurelle contenue dans les documents, mais considèrent l'information textuelle comme un

tout (c'est à dire comme une entité unique), ce qui ne permet pas d'évaluer des degrés de similarité avec les requêtes. De nouvelles approches cherchent à combiner les approches orientées données et les approches orientées documents (c'est à dire les techniques d'indexation provenant de la RI traditionnelle). Ces techniques peuvent être classées en deux catégories : les solutions *non-extensibles* et les solutions *extensibles*. Les solutions non-extensibles *nécessitent une connaissance a priori de la structure des documents*. Leur inconvénient principal est que *les documents ayant une structure différente ne peuvent pas être ajoutés*. Les solutions extensibles peuvent au contraire indexer des structures génériques différentes, mais souvent, *des fonctionnalités manquent aux index pour répondre à des requêtes portant sur des chemins logiques d'accès précis, sur des hiérarchies ou encore sur des conditions de contenus relatives à des éléments de structure*.

Considérons à présent la problématique de l'expression des besoins de l'utilisateur, c'est à dire de l'interrogation des documents. Il s'agit ici de permettre à l'utilisateur d'exprimer des besoins diversifiés (concernant le contenu des documents et/ou la structure), et ce de manière simple. De très nombreux langages d'interrogation ont été proposés dans la littérature. Ces langages, souvent proposés par la communauté des bases de données, s'attachent surtout à proposer des syntaxes pour les requêtes orientées contenu et structure. La plupart sont très puissants, mais leur utilisation nécessite une formation poussée de l'utilisateur, car *leur syntaxe est difficilement accessible*. Une *connaissance parfaite de la structure des documents est aussi souvent nécessaire à l'utilisateur pour pouvoir formuler des requêtes. Il doit de plus spécifier l'élément qu'il désire voir retourné par le SRI, alors qu'il n'a pas forcément d'idée précise de ce qu'il recherche exactement*.

La dernière problématique concerne les modèles de recherche et de tri des unités d'information. La problématique traditionnelle liée à l'évaluation de la pertinence d'une information vis-à-vis d'une requête reste d'actualité, mais elle se complique et implique d'autres questions dans le cadre des documents XML, notemment en ce qui concerne la structure. Les requêtes orientées contenu, qui sont de loin les plus simples pour l'utilisateur, imposent au SRI de décider la granularité appropriée de l'information à renvoyer. Les unités d'informations devront être les plus *exhaustives* et *spécifiques* possibles par rapport à la requête. Contrairement à la RI traditionnelle, la pertinence dans le cadre de la RI structurée est en effet exprimée selon deux dimensions : l'exhaustivité et la spécificité. *L'exhaustivité permet de mesurer à quel point l'unité d'information répond à la demande de l'utilisateur, et la spécificité permet de mesurer à quel point le contenu de l'unité d'information se focalise sur le besoin de l'utilisateur. Les modèles de recherche et de tri des unités d'information devraient donc prendre en compte ces deux dimensions de manière explicite, ce qui n'est*

pas forcément le cas des approches proposées dans la littérature, et notamment des approches orientées BD. Dans le cadre des requêtes orientées contenu et structure, deux cas sont possibles. Dans le premier cas, l'utilisateur peut exprimer des conditions sur la structure des documents, mais ne pas préciser le type des unités d'information qu'il désire voir renvoyées par le système. *Cette problématique, dans laquelle l'information structurelle peut être utilisée seulement comme une indication pour aider à retrouver l'information pertinente et non comme une indication de ce que souhaite l'utilisateur, n'a pas (ou peu) été abordée dans la littérature.* Le deuxième cas concerne les requêtes pour lesquelles le type de l'élément à renvoyer est spécifié par l'utilisateur. D'autres notions de pertinence entrent alors en jeu. La dimension de spécificité n'a plus réellement de sens, puisque l'utilisateur précise la granularité de l'information qu'il désire. Cependant, *le contenu des éléments de structure ainsi que les expressions de chemin présentes dans la requête doivent pouvoir être traitées de manière vague. En d'autres termes, un degré de pertinence doit pouvoir être attribué aux éléments.*

Contribution

Notre contribution dans le cadre de la RI structurée se situe à plusieurs niveaux et tente de répondre aux limites des approches que nous avons brièvement présentées dans la section précédente :

1. Concernant l'*indexation et le stockage des documents*, nous proposons un modèle de représentation des données combinant une approche orientée BD et une approche orientée RI. Ce modèle étend l'approche XPath Accelerator proposée dans [93] avec des concepts orientés RI, ce qui permet de conserver la structure *arborescente* des documents, de naviguer de manière aisée au sein de cette dernière, ainsi que des recherches sur le contenu textuel des documents. Ce modèle de représentation des données se veut générique et permet l'implémentation de plusieurs modèles de recherche d'information. De plus, le modèle permet de traiter des documents semi-structurés possédant des structures hétérogènes, sans avoir de connaissance *a priori* sur cette structure.
   Nous nous sommes également intéressés au problème de la pondération des termes d'indexation, et nous proposons pour ce faire de tenir compte de l'importance locale (au niveau de l'élément) et globale (au niveau de la collection) des termes. D'autres paramètres, tels que l'importance "semi-globale" (au niveau du document) des termes ou la taille des éléments, sont pris en compte au niveau du modèle de recherche.
2. Concernant l'*interrogation des documents*, nous proposons un langage permettant à l'utilisateur d'énoncer son besoin selon divers degrés de précision et selon sa connaissance du corpus qu'il interroge. Le langage

possède une syntaxe simple, qui contrairement à la plupart des approches de la littérature, ne repose pas sur un schéma similaire à SQL. Il permet la formulation de requêtes à base de simples mots-clés, sans précision aucune sur la structure : ce type de requête pourra par exemple être utilisé lorsque l'utilisateur n'a pas la moindre idée de l'unité d'information qu'il désire voir retournée. Il permet en outre de formuler des contraintes sur la structure des documents, sans nécessairement donner le type de l'unité d'information à retourner ou de formuler des requêtes plus complexes, en introduisant la notion de hiérarchie entre les différentes contraintes de structure. Dans ce dernier cas, l'utilisateur n'est pas obligé de donner des chemins absolus : le langage permet en effet l'expression de chemins vagues. Il permet enfin d'étendre les requêtes grâce à un dictionnaire des noms de balises des différents noeuds rencontrés dans le corpus. Ceci sert particulièrement dans le cas de corpus composés de documents possédant des structures génériques différentes ou dans le cas de requêtes pour lesquelles l'utilisateur ne connaît pas exactement le nom des éléments qu'il recherche.

3. Au niveau de la *recherche des unités d'information pertinentes*, nous proposons un modèle de propagation de la pertinence permettant de retrouver les unités d'information les plus spécifiques et exhaustives à une requête. Ce modèle repose sur la représentation en arbre des documents XML, dans laquelle l'information textuelle des documents est conservée au niveau des noeuds feuilles. Un premier score de pertinence est calculé pour les noeuds feuilles des documents XML, prenant en compte les pondérations locales et globales des termes. Ce score est ensuite propagé dans l'arbre du document. Nous répondons aux critères de spécificité et exhaustivité en diminuant ce score dans la propagation : pour calculer le score des noeuds internes, nous utilisons la distance entre ce noeud et les noeuds feuilles pour diminuer les scores de pertinence. Nous introduisons ensuite la notion d'informativité d'un noeud. Pour le calcul de cette dimension d'informativité, nous proposons d'utiliser la taille des éléments comme indication sur leur importance durant la propagation (plus les éléments sont petits, plus le concepteur du document a cherché à faire ressortir leur contenu) ainsi que le contexte d'appartenance des éléments, en prenant en compte la pertinence du document dans son entier dans le calcul du poids de pertinence d'un noeud interne.
   Nous proposons également d'attribuer un score de pertinence aux éventuelles conditions de structure de la requête. La correspondance entre l'arbre de la requête et l'arbre du document est effectuée de manière vague, en effectuant diverses propagations dans l'arbre du document. Par conséquent, des documents possédant une structure différente de celle la requête peuvent être renvoyés à l'utilisateur, même si leur score de pertinence est plus faible que celui des documents pour lesquels toutes les conditions de structure sont respectées. Par exemple, un document possédant la structure /a/b/c sera pertinent pour une requête /a/d/c, mais aussi pour une

requête /a/c/b. Notre modèle permet également de déterminer la granularité de l'information à renvoyer dans le cas de requêtes possédant des conditions de structure, mais pour lesquelles aucune indication sur le type d'élément à renvoyer n'est donnée.

Notre modèle apporte ainsi de la flexibilité dans la recherche à plusieurs niveaux : la représentation des document (et par conséquent la structure d'index) est générique et permet de traiter des collections de documents hétérogènes, le langage permet à l'utilisateur d'exprimer son besoin selon plusieurs degrés de précision, et les conditions de contenu et les éventuelles conditions de structure des requêtes peuvent être traitées de manière vague.

Afin de vérifier la faisabilité de ces propositions, un prototype a été développé, et les propositions ont été évaluées sur la collection fournie par la campagne d'évaluation INEX (*INitiative for the Evaluation of XML retrieval*). Dans le but de situer nos travaux par rapport à des travaux similaires dans le domaine, notre démarche d'évaluation a été effectuée selon le canevas INEX. Le principal résultat que l'on peut tirer de cette comparaison est que les performances de notre système sont comparables aux meilleurs systèmes présentés à INEX, que ce soit au niveau de la recherche orientée contenu ou de la recherche orientée structure et contenu.

Organisation du mémoire

Ce mémoire est organisé en deux parties : la première présente le contexte dans lequel se situent nos travaux, c'est à dire la recherche d'information et plus précisément la recherche d'information dans des documents semi-structurés ; la seconde décrit notre contribution, à savoir le modèle flexible que nous proposons ainsi que les expérimentations que nous avons menées.

L'objectif de la première partie, ***Recherche d'Information et Structure***, est de présenter les approches proposées dans la littérature pour la recherche d'information traditionnelle dans des documents texte "plats" ainsi que pour la recherche d'information dans des documents semi-structurés.
Le chapitre 1, **Concepts de base de la Recherche d'Information**, présente les concepts-clés de la recherche d'information. Nous commençons tout d'abord par décrire le processus générique de recherche d'information (section 1.2). La section 1.3 présente la description des différents modèles utilisés pour l'appariement entre la requête et les documents. Ces modèles sont étroitement liés, mais on distingue trois principaux courants : (1) les modèles basés sur la théorie des ensembles (section 1.3.2), (2) les modèles algébriques (section 1.3.3) et (3)

les modèles probabilistes (sections 1.3.1.3 et 1.3.4). Nous présentons enfin le mesures et collections utilisées pour évaluer les différents systèmes de recherche dans la section 1.4.
Le chapitre 2 (**Recherche d'Information Structurée**) s'intéresse aux approches proposées dans la littérature pour la recherche d'information dans des documents semi-structurés de type XML. Nous donnons un bref historique et une description de ces documents dans la section 2.2. La RI structurée est un domaine de recherche relativement nouveau, et nous présentons les nouvelles problématiques soulevées dans la section 2.3. Afin d'utiliser au mieux les propriétés des documents semi-structurés, de nouvelles techniques d'indexation (section 2.4), ainsi que de nouveaux langages d'interrogation prenant en compte la structure (section 2.5) doivent être utilisés. Différents modèles de recherche ont été proposés dans la littérature (section 2.6). Ces modèles de recherche visent à répondre à des requêtes basées sur le contenu seul (section 2.6.2) ou à des requêtes basées sur le contenu et la structure (section 2.6.3). Enfin, de nouvelles mesures visant à évaluer les Systèmes de Recherche d'Information Structurée sont présentées dans la section 2.7.

La seconde partie de ce mémoire, intitulée ***Un modèle flexible pour la Recherche d'Information Structurée***, présente nos travaux, à savoir le modèle de recherche que nous proposons et les évaluations que nous avons effectuées pour valider notre approche.
Le chapitre 3 (**XFIRM : XML Flexible Information Retrieval Model**) présente le modèle que nous proposons pour répondre aux différentes problématiques de la recherche d'information structurée. La section 3.2 rappelle nos motivations et la section 3.3 donne une présentation générale de notre modèle. XFIRM repose sur un modèle logique de représentation des données (section 3.4) et sur un langage de requête permettant à l'utilisateur d'exprimer son besoin selon divers degrés de précision (section 3.5). Le processus de recherche que nous proposons est basé sur une méthode de propagation de la pertinence, et la recherche dans des requêtes portant sur des seules conditions de contenu est modélisée dans la section 3.6. La section 3.7 présente quant à elle la recherche dans le cadre de requêtes possédant des conditions de structure et de contenu. Enfin, la section 3.8 décrit l'architecture du prototype que nous avons développé pour valider notre approche.
Le dernier chapitre, nommé **Expérimentations et résultats**, a pour but de décrire le processus d'évaluation de notre modèle et les résultats que nous avons obtenus grâce à nos expérimentations. La section 4.2 présente la collection de test que nous avons utilisée, à savoir la collection INEX, et nos conditions expérimentales sont décrites dans la section 4.3. Les sections 4.4 et 4.5 décrivent nos expérimentations, respectivement pour les requêtes orientées contenu et les requêtes orientées contenu et structure. Nous étudions dans la section 4.6 l'impact de l'unité d'indexation minimale choisie. Nos résultats sont comparés avec les résultats des différents participants d'INEX dans la section

4.7. Enfin, nous présentons dans la section 4.8 les expérimentations que nous avons menées pour la tâche hétérogène de la campagne d'évaluation INEX 2004.

On trouvera pour conclure la description de quelques standards associés à XML en annexe A.

Première partie

Recherche d'Information et Structure

Chapitre 1

Concepts de base de la Recherche d'Information

1.1 Introduction

La **Recherche d'Information** (***RI***) [173, 177] est une branche de l'informatique qui s'intéresse à l'acquisition, l'organisation, le stockage, la recherche et la sélection d'information. D'un point de vue utilisateur, l'accès à l'information peut être effectué de manière délibérée à travers un **Système de Recherche d'Information** (***SRI***) (on parle alors de *recherche ad-hoc* ou de *collecte active* de l'information), ou bien de manière *passive* à travers un système de *filtrage* d'information. Un SRI est un ensemble de programmes informatiques qui a pour but de sélectionner des informations pertinentes répondant à des besoins utilisateurs, exprimés sous forme de requêtes. Un système de filtrage peut être défini comme un processus qui permet d'extraire à partir d'un flot d'informations (News, e-mail, actualités journalières, etc.), celles qui sont susceptibles d'intéresser un utilisateur ou un groupe d'utilisateurs ayant des besoins en information relativement stables [200].

Dans ce mémoire, nous nous intéressons à la collecte active de l'information. Ce chapitre a pour objectif de présenter les concepts de base de la RI. Il est organisé comme suit.
Nous commençons tout d'abord par donner quelques définitions, puis par décrire en détail le processus de Recherche d'Information (section 1.2). Ce processus cherche à mettre en correspondance une collection de documents (section 1.2.1) et une requête utilisateur (section 1.2.2) à travers un système de recherche d'information, composé d'un module d'indexation (section 1.2.3), d'un module d'appariement document-requête (section 1.2.4) et d'un module de reformula-

tion de la requête (section 1.2.5). Nous passerons ensuite en revue les différents modèles utilisés pour l'appariement entre la requête et les documents (section 1.3). Ces modèles sont étroitement liés, mais on distingue trois principaux courants : (1) les modèles basés sur la théorie des ensembles (section 1.3.2), issus du modèle booléen (section 1.3.1.1), (2) les modèles algébriques (section 1.3.3), dont le premier représentant a été le modèle vectoriel (section 1.3.1.2) et (3) les modèles probabilistes (sections 1.3.1.3 et 1.3.4). Enfin, nous présenterons les diverses mesures et collections utilisées pour évaluer ces modèles et systèmes (section 1.4).

1.2 Le processus de Recherche d'Information

Le processus de Recherche d'Information a pour but la mise en relation des informations disponibles d'une part, et les besoins de l'utilisateur d'autre part. Ces besoins sont traduits de façon structurée par l'utilisateur sous forme de requêtes. La mise en relation des besoins utilisateurs et des informations est effectuée grâce à un Système de Recherche d'Information (SRI), dont le but est de retourner à l'utilisateur le maximum de documents pertinents par rapport à son besoin (et le minimum de documents non-pertinents). La notion de pertinence est difficile à automatiser, car elle est fortement subjective, c'est à dire dépendante de l'utilisateur. Le but du SRI est alors de faire correspondre au mieux la pertinence système avec la pertinence utilisateur. Le processus de recherche, couramment appellé Processus en U de Recherche d'Information [177, 18], est schématiquement représenté sur la figure 1.1.

Ce processus est composé de trois fonctions principales :
- l'*indexation* des documents et des requêtes ;
- l'*appariement requête-document*, qui permet de comparer la requête et le document ;
- et la fonction de *modification*, qui intervient en réponse aux résultats obtenus. Les modifications éventuelles concernent les documents (ajout ou suppression éventuels de la base de données) ou la requête. Les modifications les plus courantes concernent la requête seulement : pour cette raison, on parlera dans la suite de *Reformulation de la Requête*.

Avant de décrire en détail ces différentes fonctions du SRI, nous allons brièvement définir les deux acteurs nécessaires à son fonctionnement, à savoir d'une part l'information disponible, c'est à dire le corpus documentaire, et d'autre part l'utilisateur et son besoin en information exprimé au travers d'une requête.

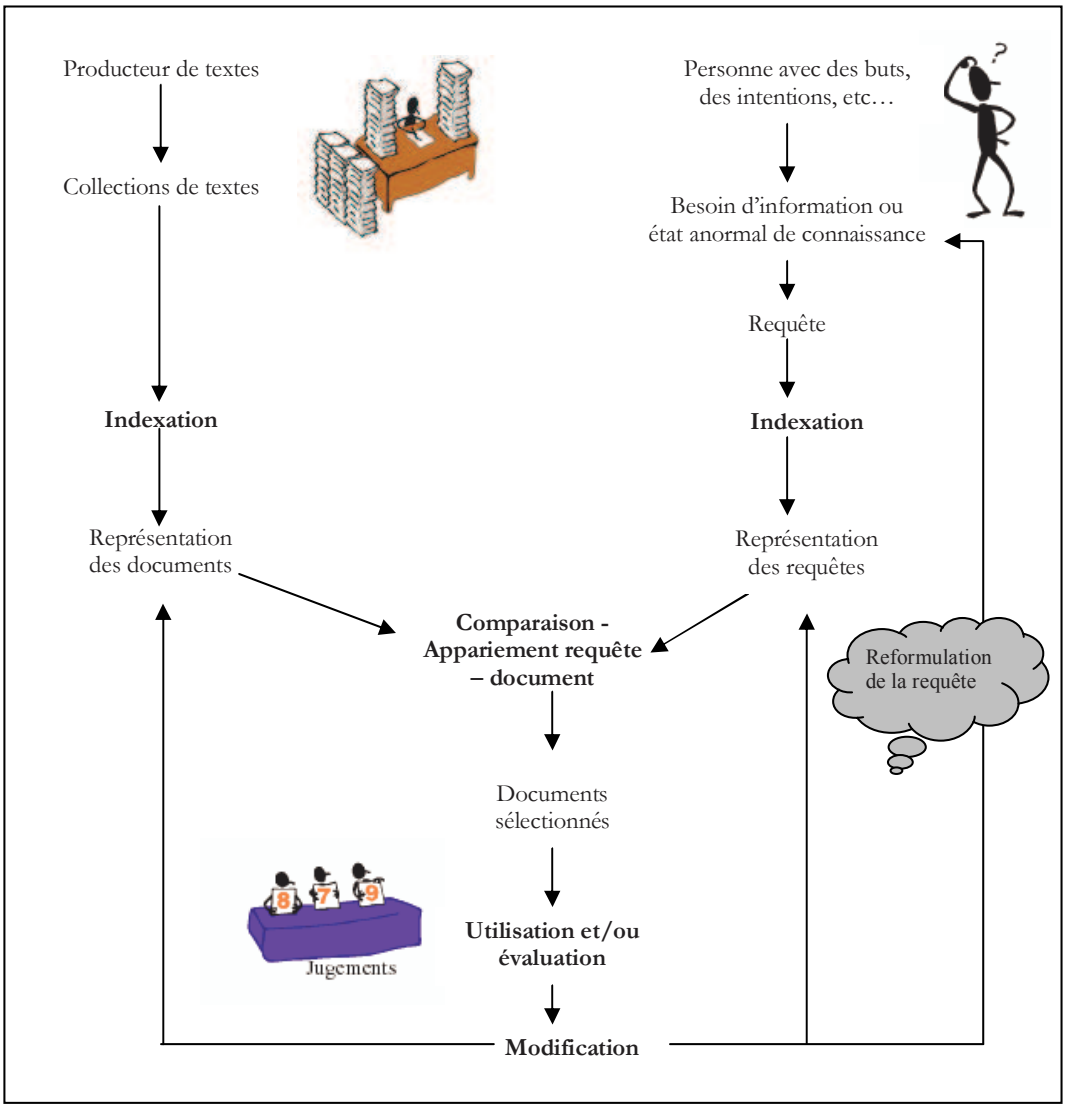


FIG. 1.1 – Processus en U de Recherche d'Information

1.2.1 Du document à la base documentaire

Le **document**, élément central du SRI, est un objet complexe sans cesse en évolution car lié aux développements des technologies de la communication. Il est ainsi important de signaler la difficulté de trouver une définition précise du terme document. Les dictionnaires donnent souvent une définition très générale. Citons par exemple le Petit Robert :

> *Document : écrit, servant de preuve ou de renseignement*

Une définition plus controversée est donnée dans [28]. Suzanne Briet affirme :

> *Un document est une preuve à l'appui d'un fait, [à savoir] tout indice concret ou symbolique, conservé ou enregistré, aux fins de représenter, de reconstituer ou de prouver un phénomène physique ou intellectuel.*

L'auteur donne plusieurs exemples (surprenants) de documents suivant cette définition. Par exemple, une antilope sauvage courant dans les plaines d'Afrique ne doit pas être considérée comme un document, alors que si elle est capturée, enfermée dans un zoo et devient l'objet d'une étude, elle devient un document. Elle est devenue une évidence physique pour ceux qui l'étudient. De plus, on peut considérer que les articles publiés sur l'antilope sont des documents secondaires, alors que l'antilope elle-même, tant qu'elle existe, est le document primaire.
La plupart des auteurs s'entendent cependant pour dire qu'un document est un objet porteur d'information. L'Institut International de Coopération Intellectuelle (*International Institute for Intellectual Cooperation*), une agence de la ligue des Nations, en collaboration avec l'Union français des Organismes de Documentation, donne, après de nombreuses concertations, la définition suivante d'un document :

> *Toute base de connaissance, fixée matériellement, susceptible d'être utilisée pour consultation, étude ou preuve.*

Un document peut ainsi être des hiéroglyphes taillés sur de la pierre, un texte sur du papier, unn texte dans un document électronique, un morceau de texte, une page Web, une image, une bande vidéo, un objet d'une collection,... On trouvera une discussion sur diverses autres définitions du terme document dans [31] et [32]. Nous nous intéressons ici *aux documents texte électroniques*. Dans cette dernière catégorie, on peut encore différencier les documents numérisés des documents non numérisés, c'est à dire des simples *fichiers texte*. Seuls ces derniers focalisent notre attention. Dans la suite du chapitre et pour plus se simplicité, le terme document sera utilisé pour représenter une Unité Documentaire. Le concept d'Unité Documentaire s'appuie sur la notion de granule, concept sémantique variant selon le contexte ou le(s) besoin(s) spécifique(s)

de l'application [50, 187]. Nous reviendrons sur la granularité de l'information dans le chapitre 2 de ce mémoire.

Un document texte peut être représenté selon plusieurs vues, comme le montre la figure 1.2 :

- la vue "*présentation*" décrit la représentation d'un document sur un medium à deux dimensions (alignement des paragraphes, entêtes et pieds de pages, ...) ;
- la vue *logique* présente la structure logique d'un document, qui contient des informations sur la structure et la partition du document ;
- la vue *de contenu*(aussi appelée vue *sémantique*) se concentre sur le contenu textuelle du document, c'est à dire sur l'information elle-même.

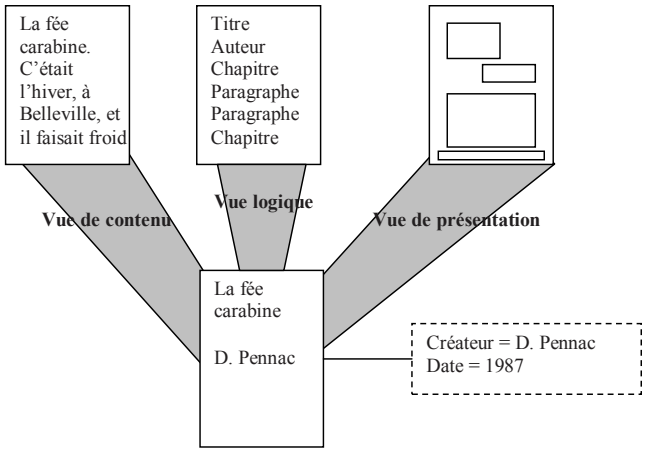


FIG. 1.2 – Vues d'un document texte, extrait de [73]

Pour les SRI traditionnels, la vue de contenu est essentielle puisque c'est sur cette vue que les utilisateurs formuleront leur requête. Les SRI traitant des documents semi-structurés s'intéressent quant à eux à la fois à la vue logique et à la vue sémantique : leur étude fait l'objet du chapitre 2.
Enfin, sous le terme **base documentaire (ou fond documentaire, collection de documents, corpus)**, on entend l'ensemble des informations exploitables et accessibles.

1.2.2 L'expression du besoin d'information : l'interrogation de la base documentaire

L'utilisateur est à la fois la source, le déclencheur d'une recherche d'information et le validateur du résultat de cette recherche. Belkin [19] constate pour l'utilisateur de Système de Recherche d'Information un état anormal de

connaissances (*Anomalous States of Knowlegde*) : l'utilisateur déclenche une recherche documentaire lorsqu'il est confronté à un manque dans sa connaissance sur un sujet. Mieux comprendre les mécanismes cognitifs de l'utilisateur, en particulier le mécanisme de satisfaction, permettrait d'améliorer les performances d'un Système de Recherche d'Information. Il ne faut pas perdre de vue que l'utilisateur d'un SRI est plus concerné par retrouver l'information sur un sujet plutôt que par retrouver des données qui satisfont une requête donnée.
De nombreuses études ont été entreprises pour modéliser l'utilisateur. Daniels [56] définit deux classes de modèles d'usager :

- Les modèles quantitatifs et empiriques : leur but est de modéliser le comportement externe de l'usager ;
- Les modèles analytiques et cognitifs : leur but est de modéliser le comportement interne de l'usager : connaissances, processus cognitifs etc...

D'une manière générale et l'analyse historique le montre, la majorité des études appartiennent à la première catégorie : l'étude des usages. Ces études apportent une modélisation du comportement de l'usager mais ne dégagent malheureusement pas une compréhension du système cognitif de l'usager, domaine des sciences psychologiques.

Le besoin de l'utilisateur est l'expression mentale de ce qu'il recherche. Ce besoin est interprété (représenté) au travers d'un requête, qui sera ensuite traitée par le SRI. Il s'agit en général d'un ensemble de mots-clés, mais elle peut être exprimée en langage naturel, booléen ou graphique. Pour Kleinberg [113], d'un point de vue sémantique, il existe trois formes différentes de requêtes :

- *Les requêtes spécifiques*, du type "Quelle est la dernière version du JDK ?"
- *Les requêtes larges*, comme par exemple : "trouve des informations concernant le langage de programmation Java"
- *Les requêtes par similarité*, du type "trouve les pages similaires à *java.sun .com*"

Ces différentes sémantiques peuvent être formulées selon différentes syntaxes.
Les requêtes composées de **listes de mots clés** sont les plus courantes. Ces mots clés peuvent éventuellement être reliés entre eux par des opérateurs booléens (ET, OU, NON), ainsi que par des variables linguistiques (comme (plus) récent, (plus) important, ...). Les mots-clés peuvent aussi être organisés sous forme d'expressions, et de nombreux SRI étendent les requêtes à partir de mots-clés avec la possibilité de chercher des mots dans un contexte donné, c'est à dire dans le voisinage d'autres mots. Ainsi une requête consiste en plusieurs mots ou phrases, avec la distance permise (en nombre de mots) entre eux.
Les requêtes en **texte libre** (ou requêtes en langage naturel) permettent à l'utilisateur d'exprimer son besoin de façon plus naturelle qu'avec une suite de mots-clés. Ces requêtes offrent surtout la possibilité d'utiliser un document complet en tant que requête (ce qui reviendrait à dire : trouve-moi tous les documents semblables à celui-ci).

Enfin, les requêtes peuvent être formées par **navigation dans une hiérarchie de concepts** (comme dans le moteur de recherche sur Internet Yahoo[1] par exemple).

Les types de requêtes que nous venons de décrire ne prennent pas ou peu en compte les caractéristiques de structuration de certains documents, tels que les documents HTML ou XML. Des langages de requêtes satisfaisant cette contrainte existent cependant. Nous les décrirons dans le chapitre concernant la recherche d'information strucurée.

1.2.3 Le processus d'indexation

Pour que le coût de la recherche soit acceptable, il convient d'effectuer une étape primordiale sur la base documentaire. Cette étape consiste à analyser chaque document de la collection afin de créer un ensemble de mots-clés : on parle de l'étape d'indexation. Ces mots-clés seront plus facilement exploitables par le système lors du processus ultérieur de recherche. L'indexation permet ainsi de créer une représentation des documents dans le système. Son objectif est de trouver les concepts les plus importants du document (ou de la requête), qui formeront le *descripteur du document*. L'indexation peut être :

- *Manuelle* : chaque document est analysé par un spécialiste du domaine ou par un documentaliste ;
- *Automatique* : le processus d'indexation est entièrement informatisé ;
- *Semi-automatique* : le choix final revient au spécialiste ou au documentaliste, qui intervient souvent pour choisir d'autres termes significatifs

L'indexation manuelle permet d'assurer une meilleure pertinence dans les réponses apportées par le SRI. Elle présente toutefois plusieurs inconvénients : deux indexeurs différents peuvent présenter des termes différents pour caractériser un même document, et un indexeur à deux moments différents peut présenter deux termes distincts pour représenter le même concept. De plus, le temps nécessaire à sa réalisation est très important.
Dans le cas d'une indexation semi-automatique [132, 17], les indexeurs utilisent un thésaurus ou une base terminologique, qui est une liste organisée de descripteurs (mots clés) obéissant à des règles terminologiques propres et reliés entre eux par des relations sémantiques.
Enfin, l'indexation automatique [133], que nous décrivons en détail dans ce qui suit, regroupe un ensemble de traitements automatisés sur un document. On distingue : l'extraction automatique des mots des documents, l'élimination des mots vides, la lemmatisation (radicalisation ou normalisation), le repérage de groupes de mots, la pondération des mots et enfin la création de l'index.

[1] http ://www.yahoo.fr

Le choix et l'intérêt d'une méthode par rapport aux autres dépend d'un certain nombre de paramètres, dont le plus déterminant est le volume des collections. On trouvera une étude comparative de ces méthodes dans [10]. Le résultat de l'étude montre que les avantages et inconvénients de chacune des approches s'équilibrent : le choix d'une méthode doit être fait en fonction du domaine, de la collection et de l'application considérée.

1.2.3.1 L'analyse lexicale

L'analyse lexicale est le processus qui permet de convertir le texte d'un document en un ensemble de termes. Un terme est une unité lexicale ou un radical [69]. L'analyse lexicale permet de reconnaître les espaces de séparation des mots, des chiffres, les ponctuations, etc.

1.2.3.2 L'élimination des mots vides

Un des problèmes majeurs de l'indexation consiste à extraire les termes significatifs et à éviter les mots vides (pronoms personnels, prépositions,...). Les mots vides peuvent aussi être des mots athématiques (les mots qui peuvent se retrouver dans n'importe quel document parce qu'ils exposent le sujet mais ne le traitent pas, comme par exemple *contenir*, *appartenir*, etc). On distingue deux techniques pour éliminer les mots vides :

- L'utilisation d'une liste de mots vides (aussi appelée *anti-dictionnaire*),
- L'élimination des mots dépassant un certain nombre d'occurrences dans la collection.

Même si l'élimination des mots vides a l'avantage évident de réduire le nombre de termes d'indexation, elle peut cependant réduire le taux de rappel[2], c'est à dire la proportion de documents pertinents renvoyés par le système par rapport à l'ensemble des documents pertinents.

1.2.3.3 La lemmatisation

Un mot donné peut avoir différentes formes dans un texte, mais leur sens reste le même ou très similaire. On peut par exemple citer *économie*, *économiquement*, *économétrie*, *économétrique*, etc . Il n'est pas forcément nécessaire d'indexer tous ces mots alors qu'un seul suffirait à représenter le concept

[2]On trouvera une définition plus précise du rappel dans la section 1.4 de ce chapitre.

véhiculé. Pour résoudre le problème, une substitution des termes par leur racine, ou lemme, est utilisée.
Frakes et Baeza-yates [70] distinguent cinq types stratégiques de lemmatisation : la table de consultation (dictionnaire), l'élimination des affixes (on peut par exemple citer l'algorithme de Porter [160]), la troncature, les variétés de successeurs ou encore la méthode des n-gramme [5].
Cette phase de passage à la forme canonique n'est pas obligatoire. Elle présente le principal avantage d'indexer par exemple le mot "camions" et le mot "camion" de la même façon ("camion"), ce qui évite à l'utilisateur de devoir entrer les formes de pluriel des noms ou les formes conjuguées des verbes lors de sa recherche. Cependant, dans certains cas, le passage à la forme canonique supprime la sémantique originale du mot. Par exemple, la forme conjuguée "portera" du verbe "porter" sera indexée sous "porte", de la même façon que le mot "portes". Ainsi, lorsque l'utilisateur formulera une requête avec le verbe "porter", il aura très certainement, parmi la liste des documents résultats, des documents non pertinents relatifs au nom "porte"...Si la lemmatisation a pour but d'augmenter le rappel, la précision (c'est-à-dire la proportion de documents pertinents par rapport au nombre de documents renvoyés par le système) en fait souvent les frais ...
Pour solutionner ce problème, C.J. Crouch [54] propose une méthode en deux temps, dont les résultats s'avèrent encourageants :

- Une première recherche est effectuée, en utilisant une lemmatisation des mots ;
- Les documents sont ensuite réordonnés en fonction de la présence des termes non-lemmatisés de la requête dans leur contenu.

1.2.3.4 La pondération des termes

La pondération des termes permet de mesurer l'importance d'un terme dans un document. Cette importance est souvent calculée à partir de considérations et interprétations statistiques (ou parfois linguistiques). L'objectif est de trouver les termes qui représentent le mieux le contenu d'un document.
Si on dresse une liste de l'ensemble des mots différents d'un texte quelconque classés par ordre de fréquences décroissantes, on constate que la fréquence d'un mot est inversement proportionnelle à son rang de classement dans la liste. Cette constatation est énoncée formellement par la loi de Zipf [234] :

$$rang * fréquence = constante.$$

La loi de distributions des termes suit alors la courbe présentée sur la figure 1.3.
Zipf explique la courbe hyperbolique de la distribution des termes par ce qu'il appelle le *principe du moindre effort* : il considère qu'il est plus facile pour un

auteur d'un document de répéter certains termes que d'en utiliser de nouveaux. La relation entre la fréquence et le rang des termes permet de sélectionner les termes représentatifs d'un document : on élimine respectivement les termes de fréquences très élevées car ils ne sont pas représentatifs du document (on peut par exemple citer les mots outils), et les termes de fréquences très faibles (ce qui permet d'éliminer les fautes de frappes et les néologismes). Ce processus est illustré sur la figure 1.3. En utilisant cette approche, le nombre de termes faisant partie de l'index d'une collection peut être réduit considérablement.

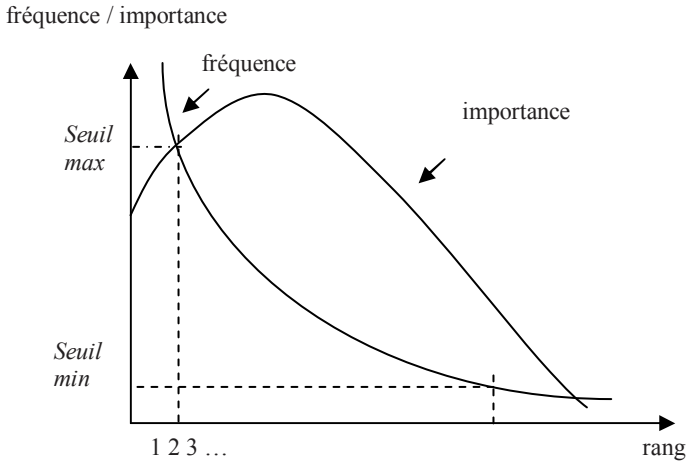


FIG. 1.3 – Importance d'un terme en fonction de sa fréquence d'apparition dans un document

A partir de ces constatations, des techniques de pondération ont vu le jour. La plupart de ces techniques sont basées sur les facteurs *tf* et *idf*, qui permettent de combiner les pondérations locale et globale d'un terme :

- **tf (Term Frequency)** : cette mesure est proportionnelle à la fréquence du terme dans le document (*pondération locale*). Elle peut être utilisée telle quelle ou selon plusieurs déclinaisons (*log(tf)*, présence/absence,...)
- **idf (Inverse of Document Frequency)** : ce facteur mesure l'importance d'un terme dans toute la collection (*pondération globale*). Un terme qui apparaît souvent dans la base documentaire ne doit pas avoir le même impact qu'un terme moins fréquent. Il est généralement exprimé comme suit : *log (N/df)*, où *df* est le nombre de documents contenant le terme et N est le nombre total de documents de la base documentaire

La mesure $tf * idf$ donne une bonne approximation de l'importance du terme dans le document, particulièrement dans les corpus de documents de taille homogène. Cependant, elle ne tient pas compte d'un aspect important du document : sa **longueur**. En général, les documents les plus longs ont tendance à utiliser les mêmes termes de façon répétée, ou à utiliser plus de termes pour décrire un sujet. Par conséquent, les fréquences des termes dans les documents seront plus élevées, et les similarités à la requête seront également plus grandes. Pour pallier cet inconvénient, Robertson [167] et Singhal et al. [195] proposent

d'intégrer la taille des documents à la formule de pondération : on parle de facteur de *normalisation*.

1.2.3.5 Les différentes techniques de création des index

Afin de répondre plus rapidement à une requête, des structures de stockage particulières sont nécessaires pour mémoriser les informations sélectionnées lors du processus d'indexation. Les moyens de stockage les plus répandus sont les suivants : les fichiers inverses ("*inverted files*"), les tableaux de suffixes ("*suffix arrays*") et les fichiers de signatures ("*signature files*").
Les fichiers inverses sont actuellement le meilleur choix possible pour la plupart des applications. Les fichiers inverses sont composés de deux éléments principaux :

- Le *vocabulaire*, qui est l'ensemble de tous les mots différents du texte ;
- Les *occurrences (posting)* : pour chaque mot, il s'agit de la liste de toutes les positions dans le texte pour lesquelles le mot apparaît.

La figure 1.4 montre un exemple de vocabulaire et d'occurrences.

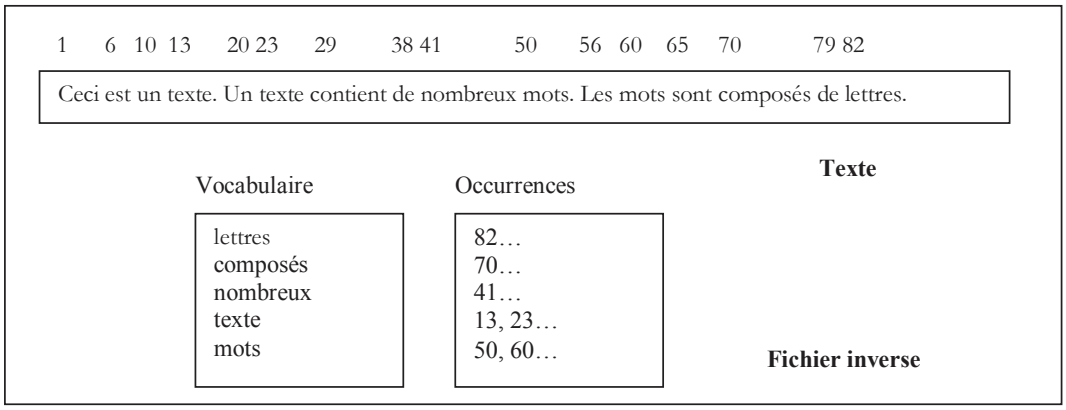


FIG. 1.4 – Un texte simple et le fichier inverse correspondant

L'espace nécessaire pour le vocabulaire est relativement petit, contrairement à celui nécessaire pour les occurrences. Pour solutionner ce problème, on utilise des blocs d'adressage [16].
Les tableaux de suffixes sont plus rapides pour des recherches de phrases et des requêtes un peu moins communes. Ils sont cependant plus difficiles à construire et à maintenir. Enfin, les fichiers de signatures sont basés sur le concept de *hashing*. Ils étaient très populaires dans les années 1980 et sont aujourd'hui beaucoup moins utilisés.
Rappelons enfin les différentes structures couramment utilisées pour l'indexation [16] :

- Les *arbres de recherche digitaux* (ou "tries") : il s'agit d'arbres multi-branches servant au stockage des chaînes de caractère. Ils sont capable de retrouver n'importe quelle chaîne de caractère en un temps proportionnel à leur longueur. Chaque arc de l'arbre est étiqueté avec une lettre. Pour chercher un mot dans un " trie ", il faut partir de la racine et descendre dans l'arbre en suivant les bons arcs
- Les *arbres de recherche binaires*
- Les *B-arbres*
- Les *tableaux triés ("sorted arrays")*
- Les *tables de hashage*

Avant de décrire le module d'appariement document-requête, rappelons que les documents ne sont pas les seuls à être indexés : les requêtes sont également perçues comme des listes de mots-clés.

1.2.4 L'appariement document-requête

La comparaison entre le document et la requête revient à calculer un score, supposé représenter la pertinence du document vis-à-vis de la requête. Cette valeur est calculée à partir d'une fonction ou d'une probabilité de similarité notée **RSV(Q,d)** *(Retrieval Status Value)*, où Q est une requête et d un document. Cette mesure tient compte du poids des termes dans les documents, déterminé en fonction d'analyses statistiques et probabilistes.
La fonction d'appariement est très étroitement liée aux opérations d'indexation et de pondération des termes de la requête et des documents du corpus. D'une façon générale, l'appariement document-requête et le modèle d'indexation permettent de caractériser et d'identifier un modèle de recherche d'information.
La fonction de similarité permet ensuite d'ordonner les documents renvoyés à l'utilisateur. La qualité de cet ordonnancement est primordiale. En effet, l'utilisateur se contente généralement d'examiner les premiers documents renvoyés (les 10 ou 20 premiers). Si les documents recherchés ne sont pas présents dans cette tranche, l'utilisateur considérera le SRI comme mauvais vis-à-vis de sa requête.
Le but de tout SRI est donc évidemment de rapprocher la pertinence système de la pertinence utilisateur (qui comme nous l'avons vu précédemment, est fortement subjective).

1.2.5 La reformulation de la requête

Il est souvent difficile, pour l'utilisateur, de formuler son besoin exact en information. Par conséquent, les résultats que lui fournit le SRI ne lui conviennent parfois pas. Retrouver des informations pertinentes en utilisant la seule requête initiale de l'utilisateur est aujourd'hui quasi-impossible, et ce à cause du volume croissant des bases documentaires. Afin de faire correspondre au mieux la pertinence utilisateur et la pertinence du système, une étape de reformulation de la requête est souvent utilisée. La requête initiale est traitée comme un essai pour retrouver de l'information. Les documents initialement présentés sont examinés et une formulation améliorée de la requête est construite, dans l'espoir de retrouver plus de documents pertinents. La reformulation de la requête se fait en deux étapes principales : trouver des termes d'extension à la requête initiale, et repondérer les termes dans la nouvelle requête.
La reformulation de la requête peut être automatique ou manuelle. Dans le premier cas, l'utilisateur n'intervient pas. L'extension de la requête est faite à partir d'un thésaurus qui définit les relations entre les différents termes de l'index et permet de sélectionner de nouveaux termes à ajouter à la requête initiale. Le thésaurus regroupe plusieurs informations de type linguistique (équivalence, association, hiérarchie) et statistique (pondération des termes). La construction du thésaurus peut être manuelle ou automatique. Parmi les thésaurus construits automatiquement, on peut citer un thésaurus basé sur les similarités [161], un thésaurus statistique [55], ou bien des mini-thésaurus construits seulement d'après la requête et à partir de techniques de clustering [14].
Considérons maintenant la reformulation manuelle de la requête. Il s'agit de la stratégie de reformulation de la requête la plus populaire [169, 23] . On la nomme communément *réinjection de la pertinence* ou *relevance feedback*. Dans un cycle de réinjection de pertinence, on présente à l'utilisateur une liste de documents jugés pertinents par le système comme réponse à la requête initiale. Après les avoir examinés, l'utilisateur indique ceux qu'il considère pertinents. L'idée principale de la réinjection de pertinence est de sélectionner les termes importants appartenant aux documents jugés pertinents par l'utilisateur, et de renforcer l'importance de ces termes dans la nouvelle formulation de la requête. Cette méthode a pour double avantage une simplicité d'exécution pour l'utilisateur qui ne s'occupe pas des détails de la reformulation, et un meilleur contrôle du processus de recherche en augmentant le poids des termes importants et en diminuant celui des termes non pertinents.

1.2.5.1 Conclusion : Points cruciaux d'un SRI

La description du processus en U de Recherche d'Information permet de dégager trois points cruciaux d'un SRI.

Tout d'abord, ce sont les index des documents qui sont jugés par rapport à une requête et pas les documents eux-mêmes. Pour que le processus de recherche soit efficace, il faut donc s'assurer que les index représentent bien le contenu sémantique des documents du corpus.
Deuxièmement, les index de documents sont comparés avec la requête interne (propre au système), et non pas avec l'expression mentale du besoin de l'utilisateur. Il faut donc que la traduction de la requête de l'utilisateur soit correctement effectuée par le SRI. Ceci en considérant bien sûr que la requête exprimée par l'utilisateur représente vraiment ses besoins, ce qui est loin d'être toujours le cas.
Enfin, si l'on considère que les représentations internes des documents et des requêtes sont correctes, il faut encore que la fonction de correspondance qui permet d'évaluer la pertinence des documents soit de bonne qualité.

1.3 Les modèles-piliers de la Recherche d'Information

Un modèle de RI a pour rôle de fournir une formalisation du processus de recherche d'information. Il doit accomplir plusieurs rôles dont le plus important est de fournir un cadre théorique pour la modélisation de la mesure de pertinence. On trouvera dans cette partie une revue des principaux modèles de recherche d'information. Ces modèles ont été décrits dans de nombreux ouvrages sur la Recherche d'Information [177, 16]. Nous détaillons ici les principaux, dont la compréhension est nécessaire pour mieux appréhender les modèles proposés pour la RI structurée, décrits dans le chapitre 2.
Ces modèles sont étroitement liés, mais on distingue trois principaux courants :

1. les modèles basés sur la *théorie des ensembles*, dont le représentant le plus connu est le modèle booléen. Dans ces modèles, des opérateurs logiques (OR, AND, NOT) séparent les termes de la requête et permettent d'effectuer des opérations d'union, d'intersection et de différence entre les ensembles de résultats associés à chaque terme.
2. les modèles *algébriques*, dont le premier représentant a été le modèle vectoriel : dans ces modèles, la pertinence d'un document vis-à-vis d'une requête est définie par des mesures de distance dans un espace vectoriel
3. les modèles *probabilistes*, reposant sur la théorie des probabilités : pour ces modèles, la pertience d'un document vis-à-vis d'une requête est vue comme une probabilité de pertinence document/requête

Dans la suite de cette section, nous nous proposons de décrire tout d'abord les trois modèles les plus représentatifs de chacun de ces courants (à savoir le modèle booléen, le modèle vectoriel et le modèle probabiliste), pour ensuite

lister les autres modèles qui les composent. Les notations que nous utilisons sont les suivantes. Soit $K = \{k_1, \ldots, k_t\}$ l'ensemble de tous les termes de l'index et k_i un terme de l'index. Soit d_j un document et w_{ij} un poids associé à la paire (k_i, d_j). Ce poids quantifie l'importance du terme de l'index pour décrire le contenu sémantique du document. A chaque document d_j est en général associé un vecteur des termes de l'index représenté par $\vec{d}_j = \{w_{1,j}, w_{2,j}, \ldots, w_{t,j}\}$. Enfin soit g_i la fonction qui retourne le poids associé au terme k_i : $g_i(\vec{d}_j) = w_{i,j}$.

1.3.1 Les modèles de RI classiques

1.3.1.1 Le modèle booléen

Le modèle booléen [174] est le plus simple des modèles de RI. C'est aussi le premier qui s'est imposé dans le monde de la recherche d'information.
Il est basé sur la théorie des ensembles et l'algèbre de Boole. Le modèle booléen considère que les termes de l'index sont présents ou absents d'un document. En conséquence, les poids des termes dans l'index sont binaires, c'est à dire $w_{i,j} \in \{0, 1\}$. Une requête q est composée de termes liés par les trois connecteurs logiques ET, OU, NON.
La similarité entre un document et une requête est définie par :

$$rsv(q, d) = \begin{cases} 1 & \text{si d appartient à l'ensemble décrit par la requête} \\ 0 & \text{sinon} \end{cases} \quad (1.1)$$

Ainsi, le modèle booléen affirme que chaque document est soit *pertinent* soit *non-pertinent*. Il n'y a pas de notion de réponse partielle aux conditions de la requête. Par exemple, considérons un document contenant les trois termes *recherche*, *information* et *traditionnelle*. Ce document ne sera pas pertinent pour la requête 'recherche ET information ET traditionnelle ET modèle'.
Le modèle booléen est le pionnier des systèmes de recherche d'information commerciaux. Son principal avantage est sa transparence. En effet, pour l'utilisateur, la raison pour laquelle un document a été sélectionné par le système est claire : il répond exactement à la requête qui a été formulée.
Cependant, il est parfois difficile pour l'utilisateur d'exprimer son besoin en information avec des expressions booléennes, et les expressions booléennes formulées sont généralement très simples, ce qui ne permet pas d'utiliser au mieux les caractéristiques du modèle. De plus, le fait que la pertinence soit basée sur un critère binaire sans notion d'échelle de gradualité empêche le modèle d'avoir de bonnes performances. Enfin, les résultats de la fonction de similarité (1 ou 0) ne permettent pas de fournir à l'utilisateur une liste ordonnée de résultats.
Aujourd'hui, il est connu qu'une pondération non binaire des termes de l'index peut amener à des améliorations notables des performances. La pondération de ces termes nous amène donc à introduire le modèle vectoriel.

1.3.1.2 Le modèle vectoriel

Le modèle vectoriel fait partie des modèles statistiques. L'utilisation des statistiques a pour but d'une part de caractériser d'un point de vue quantitatif les termes et les documents et d'autre part de mesurer le degré de pertinence d'un document vis à vis d'une requête. Le but final est d'arriver à retourner une liste ordonnée de documents selon ce degré. Un autre avantage réside dans l'expression des besoins de l'utilisateur : contrairement au modèle booléen où les termes de la requête doivent être reliés par des connecteurs logiques, l'utilisateur peut ici aussi exprimer son besoin en information en langage naturel ou sous forme d'une liste de mots clés.
Luhn [129] a été le premier à proposer une approche statistique de recherche d'information à la fin des années 1950. Il suggère que l'utilisateur fournisse un document qui ressemble à son besoin en information. La mesure de similarité entre le document fourni et la représentation des documents de la collection est utilisée pour ordonner ces documents. Le critère de similarité est ainsi défini :

> *Plus deux représentations contiennent les mêmes éléments, plus la probabilité qu'elles représentent la même information est élevée.*

Une telle définition revient en fait à compter le nombre d'éléments que partagent la requête et la représentation du document. Pour ce faire, considérons la représentation d'un document comme un vecteur $\vec{d_j} = \{w_{1,j}, w_{2,j}, \ldots, w_{t,j}\}$, où $w_{i,j}$ est le poids (0 ou 1) des termes dans le documents, t étant le nombre total de termes de l'index, et considérons la représentation de la requête comme un vecteur $\vec{q} = \{w_{1,q}, w_{2,q}, \ldots, w_{t,q}\}$, avec les mêmes notations. La mesure de similarité la plus simple est alors le produit scalaire :

$$RSV(\vec{d_j}, \vec{q}) = \sum_{i=1}^{t} w_{i,j} * w_{i,q} \tag{1.2}$$

Comme les poids des termes sont binaires, la mesure de similarité mesure le nombre de termes partagés entre le document et la requête.
Salton [173] a proposé un modèle basé sur cette mesure de similarité dans son projet SMART (*Salton's Magical Automatic Retriever of Text*). Le document (vecteur $\vec{d}$) et la requête (vecteur $\vec{q}$) sont représentés là encore dans un espace Euclidien de dimension élevée engendré par tous les termes de l'index. La similarité est alors le cosinus de l'angle formé par les deux vecteurs :

$$RSV(\vec{d_j}, \vec{q}) \quad = \quad \frac{\vec{d_j} \,.\, \vec{q}}{\left|\vec{d_j}\right| * \left|\vec{q}\right|}$$

$$= \frac{\sum_{i=1}^{t} w_{i,j} * w_{i,q}}{\sqrt{\sum_{i=1}^{t} w_{i,j}^2} * \sqrt{\sum_{i=1}^{t} w_{i,q}^2}} \quad (1.3)$$

D'autres fonctions de similarité ont été proposées dans la littérature, parmi lesquelles on peut citer les mesures de Jaccard et Dice.
Les documents sont ainsi classés en fonction de la mesure de l'angle qu'ils forment avec le vecteur requête. L'aspect le plus intéressant de cette mesure est l'influence d'un terme isolé sur le score de recherche. Si un terme est présent à la fois dans la requête et le document, il contribue au score. S'il est présent uniquement dans l'un des deux, il diminue le score parce que la requête et le document se correspondent moins.
Plusieurs algorithmes de recherche d'information ont prouvés leur performance lorsque les vecteurs requête et documents étaient normalisés. L'algorithme d'apprentissage de Rocchio en est un exemple [169].
Venons-en maintenant à la pondération des termes. Les travaux de Salton [174] ont montré qu'il ne s'agissait pas d'un problème trivial, mais les pondérations selon TF et IDF restent les plus courantes et les plus simples.
Les avantages d'un tel modèle sont nombreux : la pondération des termes augmente les performances du système, le modèle permet de renvoyer des documents qui répondent approximativement à la requête, et la fonction d'appariement permet de trier les documents selon leur degré de similarité avec la requête.
Théoriquement, le modèle vectoriel a l'inconvénient de considérer que les termes de l'index sont tous indépendants. Cependant en pratique, la prise en compte globale de la dépendance des termes peut faire baisser la qualité des réponses d'un système (puisque les dépendances sont généralement locales).
De nombreuses méthodes d'ordonnancement des résultats ont été comparées au modèle vectoriel, et celui-ci, malgré sa simplicité, est supérieur ou au moins aussi bon que les autres alternatives. C'est pour toutes ces raisons qu'aujourd'hui le modèle vectoriel est le plus populaire en recherche d'information.

1.3.1.3 Le modèle probabiliste

Le modèle probabiliste aborde le problème de la recherche d'information dans un cadre probabiliste. Le premier modèle probabiliste a été proposé par Maron et Kuhns [133] au début des années 1960. Le principe de base consiste à présenter les résultats de recherche d'un SRI dans un ordre basé sur la probabilité de pertinence d'un document vis-à-vis d'une requête. Robertson [164] résume ce critère d'ordre par le "principe de classement probabiliste", aussi désigné par PRP (*Probability Ranking Principle*).
Etant donnée une requête utilisateur, il y a un ensemble des documents qui contient exactement les documents pertinents et aucun autre. Nous appelle-

rons cet ensemble *l'ensemble de réponse idéal*. Si l'on connaît la description de cet ensemble idéal, on n'aura aucun problème à retrouver les documents qui le composent. Répondre à une requête revient donc à spécifier les propriétés de cet ensemble idéal.
Ce n'est bien sûr pas si simple que cela. Comme les propriétés de l'ensemble idéal ne sont pas connues au moment de la requête, il faut d'abord deviner ce qu'il pourrait être. Cette première tentative permet de générer une première description probabiliste de l'ensemble, qui est ensuite utilisée pour retrouver un premier ensemble de documents. Il faut ensuite une interaction avec l'utilisateur pour améliorer la description probabiliste de l'ensemble idéal (ou plutôt de l'échantillon représentant cet ensemble idéal) [164].

Le processus de recherche se traduit par calcul de proche en proche, du degré ou probabilité de pertinence d'un document relativement à une requête. Pour ce faire, le processus de décision complète le procédé d'indexation probabiliste en utilisant deux probabilités conditionnelles :

- $P(w_{ij}/Pert)$: probabilité que le terme t_i occurre dans le document d_j sachant que ce dernier est pertinent pour la requête.
- $P(w_{ij}/NonPert)$: que le terme t_i occurre dans le document d_j sachant que ce dernier n'est pas pertinent pour la requête.

Le calcul d'occurrences des termes d'indexation dans les documents est basé sur l'application d'une loi de distribution sur un échantillon représentatif de documents d'apprentissage. En posant les hypothèses suivantes :

- la distribution des termes dans les documents pertinents est la même que leur distribution par rapport à la totalité des documents.
- les variables "*document pertinent*", "*document non pertinent*" sont indépendantes.

La fonction de recherche est obtenue en calculant la probabilité de pertinence d'un document D, notée $P(Pert/D)$ [211] :

$$P(Pert/D) = \sum_{i=1}^{t} log \frac{P(w_{ij}/Pert)}{P(w_{ij}/NonPert)} \tag{1.4}$$

On trouvera dans [164] les formules utilisées pour calculer la similarité entre une requête et un document. Retenons seulement que Robertson propose aussi des formules permettant de se passer de l'intervention de l'utilisateur.
Parmi les applications du modèle probabiliste, citons le modèle 2-Poisson développé par Robertson et Walker [167] ou bien encore moteur de recherche Okapi [166, 217].

1.3.2 Autres modèles basés sur la théorie des ensembles

1.3.2.1 Le modèle flou

La représentation des documents et des requêtes par des ensembles reflète partiellement les contenus sémantiques des documents et des requêtes. Par conséquent, la correspondance d'un document avec les termes d'une requête est approximative (ou vague). Ceci peut être modélisé en considérant que chaque terme de la requête définit un ensemble flou et que chaque document possède un degré d'appartenance (généralement inférieur à 1) à cet ensemble. Le degré d'appartenance est utilisé pour représenter l'incertitude ou l'ambiguïté. Les bases de la logique floue sont présentées dans [232]. Les trois opérations les plus couramment effectuées sur des ensembles flous (le complément, l'union et l'intersection) sont ainsi définies :

$$\begin{array}{ll} \mu(a\ et\ b) & = min(\mu(a), \mu(b)) \\ \mu(a\ ou\ b) & = max(\mu(a), \mu(b)) \\ \mu(non\ b) & = 1 - \mu(b) \end{array} \tag{1.5}$$

où μ est la fonction d'appartenance floue. De nombreux opérateurs flous ont été développés. Nous proposons ici celui de Paice [149], inspiré du modèle booléen, pour qui le score d'un document pour une requête (a_1 **et** $a_2 \ldots a_n$) ou la requête (a_1 **ou** $a_2 \ldots a_n$) est calculé comme suit :

$$RSV(d,q) = \frac{\sum_{k=1}^{n} r^{k-1}\mu(a_k)}{\sum_{k=1}^{n} r^{k-1}} \tag{1.6}$$

où les $\mu(a_k)$ sont considérés dans un ordre décroissant pour les requêtes *ou* et croissant pour les requêtes *et*. Pour les requêtes booléennes contenant plus d'un opérateur, l'évaluation est effectuée de manière récursive. La valeur de r est déterminée expérimentalement pour les deux opérateurs, elle détermine la "douceur" de l'opérateur. Pour une valeur proche de 1, les deux opérateurs possèdent le même comportement. Pour de grandes valeurs, les opérateurs se comportent de plus en plus comme dans le modèle booléen.
Une autre application du modèle flou est la construction et l'utilisation d'un thesaurus [145] ou d'une ontologie [128] pour étendre la requête. Le modèle peut enfin être utilisé pour former une nouvelle requête dans un cycle de reformulation de la requête [30].

1.3.2.2 Le modèle booléen étendu

Le modèle booléen étendu a été introduit en 1983 par Salton, Fox et Wu [176]. Ce modèle peut être vu comme une combinaison du modèle vectoriel et du modèle booléen.
Prenons un exemple simple avec seulement deux termes dans la requête, k_x et k_y. On peut représenter les requêtes et les documents dans un espace à deux dimensions :

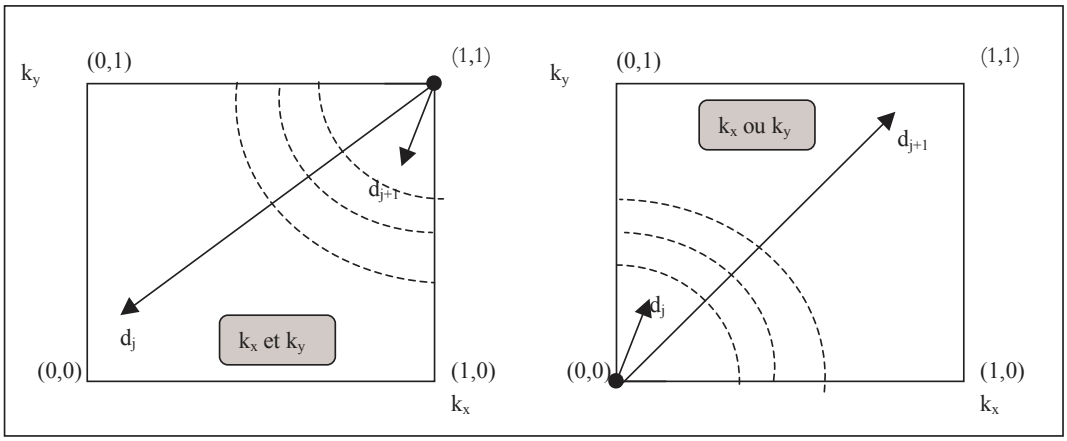


FIG. 1.5 – Logique booléenne étendue en considérant un espace composé de deux termes k_x et k_y

Pour plus de simplicité dans la suite des notations, le poids $w_{x,j}$ sera appelé x et le poids $w_{y,j}$ sera appelé y. Pour la requête k_x **ou** k_y, le point (0,0) correspond à la situation où aucun des termes k_x et k_y n'est présent dans le document, cette situation est donc la moins désirée. Pour la requête k_x **et** k_y, le point (1,1) représente la situation où les termes k_x et k_y sont présents dans le document, il s'agit du cas le plus désiré. On peut donc conclure que les requêtes avec l'opérateur **ou** doivent ranger les documents dans l'ordre décroissant de leur distance au point (0,0) et que les requêtes avec l'opérateur **et** doivent ranger les documents dans l'ordre croissant de leur distance au point (1,1).
Sur cette base, les scores de similarité document-requête sont ainsi calculés :

$$\begin{aligned} RSV(\vec{d}, q_{ou}) &= \sqrt{\frac{x^2+y^2}{2}} \\ RSV(\vec{d}, q_{et}) &= 1 - \sqrt{\frac{(1-x)^2+(1-y)^2}{2}} \end{aligned} \tag{1.7}$$

Le modèle *p-norm* généralise cette notion de distance en incluant non seulement les distances euclidiennes mais aussi les p-distances, avec $1 \leq p \leq \infty$. La

valeur de p est indiquée au moment de la requête. Si m est le nombre de termes dans la requête, les fonctions de similarité deviennent alors :

$$\begin{aligned} RSV(\vec{d}, q_{ou}) &= (\frac{x_1^p + x_2^p + \ldots + x_m^p}{m})^{\frac{1}{p}} \\ RSV(\vec{d}, q_{et}) &= 1 - (\frac{(1-x_1)^p + (1-x_2)^p + \ldots + (1-x_m)^p}{m})^{\frac{1}{p}} \end{aligned} \quad (1.8)$$

Si $p = 1$, on se ramène au modèle booléen et si $p = 2$, on retrouve les formules de l'équation 1.7. Enfin si $p = \infty$, on peut vérifier que l'on se ramène aux opérateurs flous :

$$\begin{aligned} RSV(\vec{d}, q_{ou}) &= max(x_i) \\ RSV(\vec{d}, q_{et}) &= min(x_i) \end{aligned} \quad (1.9)$$

Le modèle booléen étendu étend l'algèbre de Boole avec des distances algébriques. Il s'agit ainsi d'un modèle hybride qui inclut les propriétés des modèles ensembliste et algébrique. Le modèle booléen étendu n'a pas été beaucoup utilisé par la suite, mais il donne un cadre nouveau à la recherche d'information, cadre qui pourrait s'avérer utile dans le futur.

1.3.3 Autres modèles algébriques

1.3.3.1 Le modèle vectoriel généralisé

Les trois modèles classiques décrits au paragraphe précédent considèrent que les termes de l'index sont indépendants. Pour le modèle vectoriel, ceci se traduit par le fait que les vecteurs représentant les termes de l'index sont orthogonaux deux à deux.
En 1985, Wong, Ziarko et Wong proposent une interprétation dans laquelle les vecteurs des termes de l'index sont linéairement indépendants mais non orthogonaux deux à deux. Cette interprétation est appelée le modèle vectoriel généralisé. On trouvera les détails de cette théorie dans [224]. D'une manière générale, la contribution principale du modèle est l'établissement d'un cadre formel dans lequel les dépendances entre les termes de l'index peuvent être facilement représentées.
Cependant, il est loin d'être prouvé que l'introduction de dépendances entre termes dans un modèle permette d'augmenter son efficacité. De plus, le modèle vectoriel généralisé est plus compliqué et plus lent que le modèle vectoriel classique.
Il n'en reste pas moins remarquable qu'un tel modèle introduit de nouvelles possibilités dans le monde de la recherche d'information.

1.3.3.2 Latent Semantic Indexing Model (LSI)

L'idée principale du modèle LSI (*Latent Semantic Model*) [80] est que les idées dans un texte sont plus reliées aux concepts décrits par elles que les termes de l'index utilisés pour leur description. Ainsi, la correspondance entre un document et une requête donnée devrait être basée sur la correspondance des concepts plutôt que sur la correspondance des termes de l'index. L'objectif fondamental est d'aboutir à une représentation conceptuelle des documents. Ainsi, les documents qui partagent des termes co-occurents ont des représentations proches, ce qui permet de sélectionner un document même s'il ne contient aucun mot de la requête. Pour ce faire, on se place dans un espace de moindre dimension associé aux concepts. Les vecteurs des termes de l'index sont convertis dans cet espace, et le modèle affirme que la recherche dans l'espace réduit donne de meilleurs résultats que la recherche dans l'espace des termes de l'index.
Formellement, soit N la matrice termes-documents (par exemple en utilisant le critère TF- IDF). LSI permet de trouver une approximation $\widetilde{N}$ de N telle que :

$$\widetilde{N} = U\widetilde{\sum}V^t \tag{1.10}$$

où U et V représentent des matrices telles que $U^tU = V^tV = I$, et $\widetilde{\sum}$ donnée par :

$$\widetilde{\sum} = (\sigma_1, \ldots \sigma_r, 0, \ldots, 0) \tag{1.11}$$

est une matrice diagonale, les σ_i sont les composantes principales avec $\forall i \in \{1, \ldots r-1\}$, $\sigma_i \geq \sigma_{i+1}$, et $\widetilde{\sum} = (\sigma_1, \ldots \sigma_s, 0, \ldots, 0)$ est une approximation de $\sum$, avec $s < r$.
La similarité entre deux documents d_i et d_j est calculée comme suit :

$$\begin{aligned} RSV(d_i, d_j) &= S_{i,j} \\ S &= \widetilde{N}\widetilde{N}^t \\ &= U\widetilde{\sum}^2 U^t \end{aligned} \tag{1.12}$$

Le calcul de similarité entre le document et la requête est calculé de la même façon. D'après [202], le principal inconvénient de cette méthode est qu'elle n'est pas souple pour certains types d'applications dont le filtrage. En effet, la performance et la stabilité du système dépendent largement de la quantité et de la qualité des données traitées. Si le nombre de documents est faible, le calcul de $\widetilde{N}$ ne donne pas une vraie approximation de N et le processus devient erroné.

1.3.3.3 Le modèle connexionniste

Sous le terme réseaux de neurones, on regroupe un certaine nombre de modèles dont l'objectif est d'imiter quelques fonctions du cerveau humain en reproduisant certaines de ses structures de base.
Le fonctionnement du réseau se fait par propagation de signaux de la couche d'entrée vers la couche de sortie. Chaque neurone de la couche d'entrée reçoit une valeur d'activation, calcule une valeur de sortie et la transmet vers les neurones qui lui sont reliés dans la couche suivante. Ce processus se reproduit jusqu'à arriver à la couche de sortie, les valeurs de sorties dans la couche de sortie servant de critère de décision.
La notion de réseau en général est très intéressante pour représenter les différentes relations et associations qui existent entre les termes et les documents. Ceci est d'autant plus vrai quand ces relations sont valuées. Différentes relations peuvent exister entre les termes et les documents :

- Relations entre les termes : synonymie, voisinage,...
- Relations entre les documents : similitude, référence,...
- Relations entre les termes et les documents : fréquence, poids,...

La figure 1.6 représente un modèle de réseaux de neurones pour la recherche d'information.
Une représentation sous forme de réseau permet de mettre en évidence l'im-

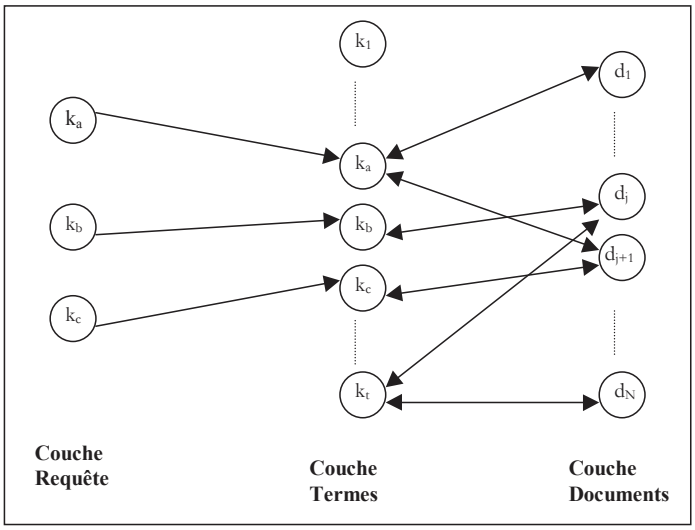


FIG. 1.6 – Un modèle de réseau de neurones pour la recherche d'information

portance des relations et des interactions qui peuvent exister entre les différents éléments d'un système documentaire. Il n'existe pas de représentation unique d'un réseau de neurones pour la recherche d'information, c'est au constructeur du système de la définir (nombre de couches, nombre de neurones par couche, fonction de sortie de chaque neurone, liens entre les neurones et poids des neurones, couche d'entrée et couche de sortie).

La propriété la plus importante dans un réseau de neurones est l'apprentissage. Il s'agit en fait d'un entraînement du réseau : on présente au réseau des entrées et on lui demande de modifier sa pondération de telle sorte que l'on retrouve la sortie correspondante. Pour effectuer cet apprentissage, l'algorithme de retro-propagation du gradient [171] est de loin le plus utilisé. On notera cependant que l'apprentissage dans les réseaux de neurones n'est pas exempt de contraintes comme un coût élevé en temps d'exécution, et une efficacité significative à partir d'un certain nombre de couches cachées, nombre qu'il n'est pas aisé de définir.
Citons maintenant quelques unes des applications des réseaux de neurones en recherche d'information. Ogawa [146] et Robertson [165] s'en servent pour l'expansion des requêtes. Dans [114, 115, 143], les cartes auto-organisatrices de Kohonen (en anglais *Self Organization Map*, *SOM*) sont utilisées pour répondre à des problèmes de classification. Dans [202], le modèle connexionniste est utilisé pour le filtrage d'information. Wilkinson [222] fait partie des précurseurs de l'utilisation des réseaux de neurones pour la recherche de documents pertinents. PIRCS [119] et Mercure (*ModèlE de Reseau Connexionniste poUr la REcherche d'Information*) [22] sont deux systèmes de recherche d'information entièrement basés sur l'approche connexionniste.
Les réseaux de neurones proposent une approche originale de la recherche d'information, et ce grâce aux possibilités de leur apprentissage. On peut cependant regretter leur aspect "boîte noire" : il est très difficile, voire impossible pour l'utilisateur, de comprendre pourquoi tel ou tel document a été sélectionné, contrairement aux modèles booléens et vectoriels.

1.3.4 Autres modèles probabilistes

1.3.4.1 Les réseaux bayésiens

Les réseaux bayésiens [150] sont des graphes directs acycliques dans lesquels les noeuds représentent des variables aléatoires, et les liens des relations de dépendance entre ces variables. En associant des probabilités initiales pour les racines du graphe, on calcule de proche en proche le degré de croyance associé à chacun des noeuds restants. Deux écoles traditionnelles en probabilité s'affrontent : l'une est basée sur l'aspect fréquentiel et l'autre sur l'aspect épistémologique. L'approche fréquentielle prend les probabilités comme une notion statistique reliée aux lois du hasard. L'approche épistémologique interprète les probabilités comme un degré de croyance dont les spécifications viennent de statistiques expérimentales.
Les réseaux inférentiels bayésien [209] considèrent le problème de la recherche d'information d'un point de vue épistémologique. Ils associent des variables

aléatoires avec les termes de l'index, les documents et les requêtes de l'utilisateur. Les termes de l'index et les documents sont représentés comme des noeuds. Une variable aléatoire associée avec un document d_j représente l'événement d'observer ce document. Les arcs sont dirigés du noeud document vers ses noeuds termes : ainsi, l'observation d'un document est la cause d'une augmentation de la valeur des variables associées avec ses termes d'index. La variable aléatoire associée à la requête de l'utilisateur modélise l'événement que la requête d'information spécifiée dans la requête a été vérifiée. La valeur de ce noeud requête est une fonction des valeurs des noeuds associés aux termes de la requête. Ainsi, les arcs sont orientés des noeuds des termes de l'index vers le noeud de la requête.
La figure 1.7, issue de [209], illustre un réseau inférentiel bayésien simple de pertinence d'un document vis à vis d'une requête composée de trois termes. L'événement "la requête est accomplie" (Q=1) est réalisé si le sujet lié à un

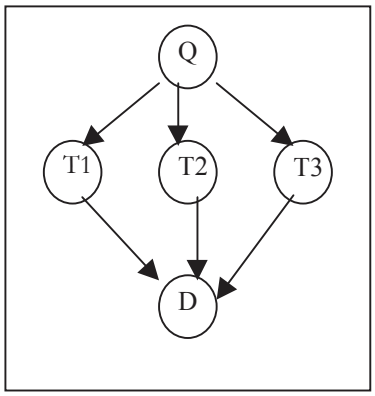


FIG. 1.7 – Modèle de réseau inférentiel bayésien simple

terme est vrai (T1=1, T2=1 ou T3=1), ou une combinaison de ces événements. Les trois sujets sont inférés par l'événement "le document est pertinent" (D=1). Par l'enchaînement de règles de probabilités, la probabilité jointe des autres noeuds du graphe est :

$$P(D,T1,T2,T3,Q) = P(D)\ P(T1|D)\ P(T2|D,T1)\ P(T3|D,T1,T2)\ P(Q|D,T1,T2,T3)$$

La direction des arcs indiquant les relations de dépendance entre les variables aléatoires, l'équation devient :
$P(D,T1,T2,T3,Q) = P(D)P(T1|D)P(T2|D)(T3|D)P(Q|T1,T2,T3)$
La probabilité de réalisation de la requête $P(Q=1|D=1)$ peut être utilisée comme score d'ordonnancement des documents :

$$P(Q=1|D=1) \quad = \quad \frac{P(Q=1,D=1)}{P(D=1)}$$

$$= \frac{\sum P(D=1, T1=t_1, T2=t_2, T3=t_3, Q=1)}{P(D=1)} \quad (1.13)$$

Le modèle nécessite la connaissance de $P(D=[0|1])$, $P(Ti=[0|1]|D=[0|1])$, $P(Q=[0|1]|\ (T1,T2,\ldots,Tn) \in \{0,1\}^n\ )$, cette dernière étant la plus difficile à trouver car le nombre de probabilités à spécifier augmente exponentiellement avec le nombre de termes de la requête. Pour résoudre ce problème, Turtle [208] a identifié quatre formes canoniques de $P(Q|T1,T2,\ldots Tn)$: *and, or , sum* et *wsum*.
Le modèle inférentiel bayésien a été mis en oeuvre dans le système Inquery [7]. Le cadre probabiliste dans lequel se situe Inquery peut être utilisé pour formuler des requêtes simples basées sur des mots clés, des requêtes booléennes, des requêtes basées sur des phrases ou bien une combinaison des trois types [52]. Pour ce faire, Inquery propose des opérateurs de moyenne et de moyenne pondérée, des opérateurs booléens probabilistes ou stricts (on conserve alors les probabilités), des opérateurs de proximité et de synonymie. Une procédure d'analyse de la requête permet de générer une forme inférentielle prête à être évaluée. Inquery propose également une expansion de requête.
Basés sur les réseaux inférentiels bayésiens, les "belief networks" ont été introduits en 1996 par Ribeiro-Neto et Muntz [163]. Ils sont aussi basés sur une interprétation épistémologique des probabilités, mais travaillent dans un espace différent. En conséquence, on obtient une topologie de réseau différente, qui permet la séparation entre l'espace des documents et l'espace des requêtes. On peut ainsi combiner des sources distinctes d'évidence (requêtes passées, cycles de feedback précédents, formulations distinctes de requêtes), ce qui permet d'augmenter les performances du système (c'est à dire augmenter la qualité de la liste ordonnée de documents renvoyée par le système).
Dans [213], le document est représenté dans le réseau de deux façons différentes (les termes du titre et les termes du résumé du document), et la requête peut aussi être représentée par des requêtes différentes.
L'inconvénient principal des réseaux bayésiens reste le calcul des probabilités, qui demande un temps exponentiel au nombre de termes dans la requête même si l'introduction des quatre formes canoniques dans [208] résout partiellement le problème.

1.3.4.2 Les modèles de langage

Dans les modèles de recherche probabilistes "classiques", on cherche à estimer la probabilité que le document réponde à la requête. L'hypothèse de base dans ces modèles est qu'un document n'est pertinent que s'il ressemble à la requête. Les modèles de langage sont basés sur une hypothèse différente : u*n utilisateur en interaction avec un système de recherche fournit une requête en pensant à un ou plusieurs documents qu'il souhaite retrouver*. La requête est

alors inférée par l'utilisateur à partir de ces documents. Un document n'est pertinent que si la requête utilisateur ressemble à celle inférée par le document. On cherche alors à estimer la probabilité que la requête soit inférée par le document [159, 25]. Les modèles de langages calculent cette probabilité et l'utilisent pour ordonner les documents. Etant donné une requête $T_1, T_2, \ldots, T_n$, les documents sont ordonnés selon la mesure suivante :

$$P(T_1, T_2, \ldots, T_n|D) = \prod_{i=1}^{n}((1-\lambda_i)P(T_i) + \lambda_i P(T_i|D)) \tag{1.14}$$

Cette mesure est une combinaison linéaire du modèle de document et du modèle de contexte du document (la collection), où : λ_i est la probabilité que le terme à la position i soit important, $1-\lambda_i$ est la probabilité que le terme ne soit pas important, $P(T_i|D)$ est la probabilité d'un terme important et $P(T_i)$ est la probabilité d'un terme sans importance. Les probabilités sont définies de la manière suivante :

$$P(T_i|D) = \frac{tf(T_i|D)}{\sum_T tf(T,D)}, \text{terme important} \tag{1.15}$$

$$P(T_i) = \frac{df(T_i)}{\sum_T df(T)}, \text{terme sans importance} \tag{1.16}$$

où $tf(T_i|D)$ est la fréquence du terme T_i dans le document D et $df(T)$ est le nombre de documents dans lesquels T apparait. Ces deux probabilités sont estimées en utilisant une estimation de vraisemblance (*maximum likelihood estimation*), et λ est appelé paramètre de lissage (*smoothing parameter*). Le calcul des probabilités peut être réduit à la formule de calcul de scores suivante :

$$\begin{aligned} s(D, T_1, T_2, \ldots, T_n) &= \beta.log(\sum_T tf(T,D)) \\ &+ \sum_{i=1}^{n} log(1 + \frac{\lambda.tf(T_i,D).(\sum_T df(T))}{(1-\lambda).df(T_i).(\sum_T tf(T,D))}) \end{aligned} \tag{1.17}$$

Le paramètre β sert à estimer des probabilités *a priori* (*prior probability*) et est utilisé pour introduire la longueur des documents dans la formule de calcul des scores, c'est à dire pour normaliser ces scores. Une question se pose cependant : comment estimer la valeur de λ_i ? Pour une première recherche, on a : $\lambda_i = constante$, c'est à dire que tous les termes sont considérés comme ayant la même importance. λ_i est ensuite réévalué pour chaque terme dans un cycle de réinjection de la pertinence.

Les modèles de langages, reposant sur la théorie des probabilités et sur les chaînes de Markov, ont aussi été appliqués avec succès à la reconnaissance vocale [159, 96] et leur application à la recherche dans des documents structurés [104] ou à la traduction automatique de documents est aujourd'hui en cours d'essai [226].

1.4 Evaluation des Systèmes de Recherche d'Information

L'évaluation des systèmes peut être abordée selon deux angles : l'*efficience* et l'*efficacité*. L'efficience regroupe le temps et l'espace : plus le temps de réponse est court et plus l'espace occupé par le système est faible, meilleur est considéré le système. Ces critères ne concernent cependant que les systèmes qui assurent parfaitement une fonction précise, ce qui n'est pas le cas dans le domaine de la recherche d'information.
D'autres mesures de performances des SRI ont donc été introduites, dans le but d'évaluer l'efficacité des systèmes. Parmi elles ont peut citer la facilité d'utilisation du système, ou encore la présentation des résultats [47]. Nous nous intéressons ici à celle qui nous semble la plus importante : *la capacité d'un système à sélectionner des documents pertinents*. Les mesures que nous présentons dans la suite de cette section rendent possible la comparaison des SRI entre-eux. Cependant, pour que la comparaison soit valable, il faut que ces mesures soient effectuées sur les mêmes bases documentaires. C'est de cette nécessité que sont nées de nombreuses campagnes d'évaluation, dont nous donnons un exemple dans la deuxième partie de cette section.
La performance des SRI est évaluée à partir de la pertinence des documents renvoyés. Cette notion de pertinence est ambiguë. En effet, on peut parler de pertinence objective, c'est à dire une pertinence calculée à partir des résultats du SRI, mais aussi de pertinence subjective : un document peut être jugé pertinent à une requête par un utilisateur et pas par un autre. De même, la pertinence d'un document dépend des connaissances de l'utilisateur sur le sujet, et peut affecter la pertinence des documents examinés par la suite.
C'est pour ces raisons que des mesures d'évaluation orientées utilisateurs ont été introduites. Nous présentons ici les mesures d'évaluation de SRI les plus courantes, ainsi qu'un exemple de campagne d'évaluation utilisée par les centres de recherche pour comparer leurs différents systèmes.

1.4.1 Evaluation de la performance d'un Système de Recherche d'Information

1.4.1.1 Rappel et précision

D'une façon générale, tout SRI a deux objectifs principaux : retrouver tous les documents pertinents, et rejeter tous les documents non pertinents. Ces objectifs sont évalués par les mesures de rappel et précision. Soient $|A|$ le nombre de documents renvoyés par un système pour une requête donnée, $|R|$ le nombre de documents pertinents dans la collection pour cette requête et $|Ra|$ le nombre

de documents pertinents renvoyés par le système (figure 1.8).

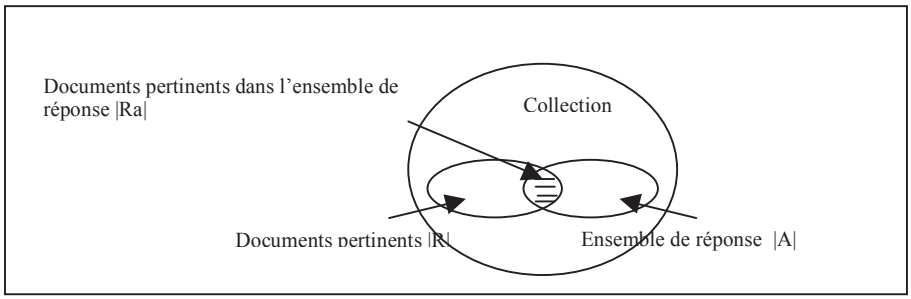


FIG. 1.8 – Précision et Rappel

La **précision** mesure la proportion de documents pertinents relativement à l'ensemble des documents restitués par le système. Elle est exprimée par :

$$precision = \frac{|Ra|}{|A|} \tag{1.18}$$

Le **rappel** mesure la proportion de documents pertinents restitués par le système relativement à l'ensemble des documents pertinents contenus dans la base documentaire. Il est exprimé par :

$$rappel = \frac{|Ra|}{|R|} \tag{1.19}$$

Courbes de Rappel-Precision La précision mesurée indépendamment du rappel et inversement est peu significative. Pour pouvoir examiner les résultats efficacement, on calcule la paire des mesures (taux de rappel, taux de précision) à chaque document restitué. Le tableau 1.1 illustre des calculs de précision et de rappel pour les 10 premiers documents renvoyés par un système pour deux requêtes différentes, pour lesquelles la collection contient respectivement 5 et 4 documents pertinents. Les courbes de rappel-précision associées sont tracées sur la figure 1.9.

Comme on peut le constater, plusieurs valeurs de précision peuvent correspondre au même point de rappel. Afin d'obtenir des courbes plus aisées à lire, on ne représente généralement que la précision calculée à chaque point de rappel (c'est à dire à chaque document pertinent restitué) (voir figure 1.10).

Pour avoir une évaluation de la performance du système sur toutes les requêtes et non pas sur une seule, on calcule une **précision moyenne** à chaque

	R1			R2		
Rang du doc. restitué	Pertinent	Rappel	Précision	Pertinent	Rappel	Précision
1	x	0.20	1.0	x	0.25	1
2	x	0.40	1.0		0.25	0.50
3		0.40	0.67	x	0.50	0.67
4	x	0.60	0.75		0.50	0.50
5		0.60	0.60		0.50	0.40
6	x	0.80	0.67		0.50	0.33
7		0.80	0.57		0.50	0.29
8		0.80	0.50		0.50	0.25
9		0.80	0.44		0.50	0.22
10	x	1	0.50	x	0.75	0.30

TAB. 1.1 – Exemple de calcul de rappel et précision pour les requêtes R1 et R2

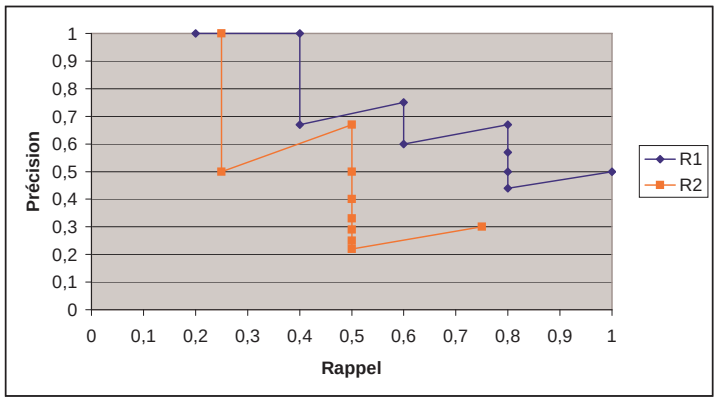


FIG. 1.9 – Courbes de rappel-précision des requêtes R1 et R2

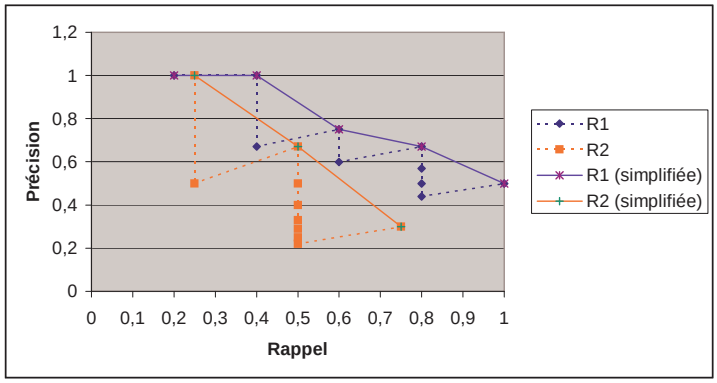


FIG. 1.10 – Courbes de rappel-précision simplifiées des requêtes R1 et R2

niveau de rappel. Pour ce faire, il faut unifier les niveaux de rappel pour l'ensemble des requêtes. On retient généralement 11 points de rappel standards, de

	R3		R4	
Rang	Pertinent	Non Pertinent	Pertinent	Non Pertinent
1	2	1	1	0
2	4	5	1	1
3	2	0	5	8
4	1	4	2	0
5	1	79	2	78
	10	90	10	90

TAB. 1.2 – Exemple liste de documents non ordonnée strictement

0 à 1 à pas de 0.1. Les valeurs de précision non obtenues à partir des valeurs de rappel sont calculées comme suit, par interpolation linéaire.
Pour deux points de rappel, i et j, $i < j$, si la précision au point i est inférieure à celle au point j, on dit que la précision interpolée à i égale la précision à j. Formellement :

$$p'_i = max(p_i, p_j), \forall i < j \tag{1.20}$$

où p'_i est la précision interpolée au point de rappel i, et p_i est la vraie précision au point de rappel i. Cette interpolation est encore discutable, mais présente un intérêt dans l'évaluation de SRI [176].
Le système parfait trouverait seulement les documents pertinents, avec une précision et un rappel de 100%. En pratique, les mesures de rappel et précision évoluent inversement, ce qui signifie que le courbe interpolée de précision en fonction du rappel est décroissante. Plus la courbe est élevée, plus le système est performant.

Le problème de l'ordonnancement Les valeurs de rappel et de précision donnent une bonne approximation de la performance d'un système lorsque celui-ci renvoie des listes strictement triées de résultats. Cependant, il arrive souvent que plusieurs documents obtiennent le même score de pertinence, et soient donc renvoyés au même rang par le système. Considérons les exemples du tableau 1.2, pour lesquels le nombre de documents pertinents dans la collection est de 10. Pour la requête R3, le système a renvoyé au premier rang 2 documents pertinents et 1 document non-pertinent, au second rang 4 documents pertinents et 5 non-pertinents, etc. Si l'on applique les mesures de rappel/précision comme vu ci-dessus et que dans un même rang, les documents pertinents se trouvent "classés" aléatoirement à la fin, les performances du systèmes seront plus faibles que s'ils sont "classés" au début. Pour pallier ce problème, la mesure de **Precall** a été proposée [231, 162].

Soit $|R|$ le nombre de documents pertinents à une requête donnée dans la collection et $|Ra|$ le nombre de documents pertinents qu'un système doit re-

trouver (correspondant au point de rappel $|Ra|/|R|$). Commençons la recherche de ces $|Ra|$ documents au premier rang des documents renvoyés par le système (rang auquel le score des documents est le plus élevé) et descendons jusqu'au rang l auquel ils sont tous retrouvés. Supposons qu'il y ait r documents pertinents et i documents non-pertinents à ce rang l. Imaginons que les r documents pertinents forment r intervalles et que les i documents non-pertinents soient distribués uniformément sur ces r intervalles. Pour chaque document pertinent retrouvé, on estime donc que i/r documents non pertinents sont aussi retrouvés. Le nombre $|NR|$ de documents non pertinents renvoyés par le système au rang l et au point de rappel $|Ra|/|R|$ est ainsi exprimé de la façon suivante :

$$|NR| = j + \frac{s.i}{r} \tag{1.21}$$

où j est le nombre de documents non pertinents rencontrés dans les rangs précédant l et s est le nombre de documents pertinents rencontrés au rang l avant d'atteindre le $|Ra|$ième document pertinent.
D'après la méthode de Precall, la précision moyenne au niveau de rappel $|Ra|/|R|$ est alors exprimée de la façon suivante :

$$Precision = \frac{|Ra|}{|Ra| + j + (s.i)/r} \tag{1.22}$$

Par exemple, pour la requête R3, la précision au point de rappel (1/10) est égale à 1/(1+0+1*1/2)=2/3 et au point de rappel (5/10) à 5/(5+1+3*5/4)=20/39. On trouvera une représentation des courbes de rappel-précision des requêtes R3 et R4 en suivant la méthode de Precall sur la figure 1.11.

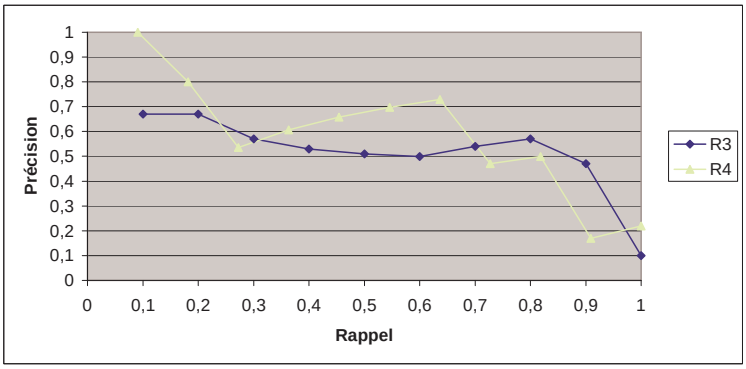


FIG. 1.11 – Courbes de rappel-précision des requêtes R3 et R4 en suivant la méthode de Precall

La précision peut aussi être interprétée comme la probabilité $P(pert|retr)$ qu'un document retrouvé dans la liste des résultats soit pertinent [71]. Cette précison au point de rappel ($|Ra|/|R|$) s'exprime de la façon suivante :

$$P(pert|retr)(|Ra|) = \frac{|Ra|}{|Ra| + esl_{|Ra|}} = \frac{|Ra|}{|Ra| + j + s.i/(r+1)} \tag{1.23}$$

où $esl_{|Ra|}$ est la longueur supposée de recherche (*expected search length*), mesure introduite par Cooper [51] en 1968. La pertinence $P(pert|retr)$ est connue dans la littérature sous le nom de *Probability of Relevance* (**PRR**).
Dans [162], Raghavan et al. donnent une justification théorique que des nombres réels intermédiaires peuvent aussi être utilisés pour calculer cette probabilité de pertinence, et proposent ainsi une méthode intuitive pour l'interpolation :

$$P(pert|retr)(x) = \frac{x.|R|}{x.|R| + esl_{x.|R|}} = \frac{x.|R|}{x.|R| + j + s.i/(r+1)} \tag{1.24}$$

où $|R|$ est le nombre de documents pertinents dans la collection.

Autres mesures dérivées du rappel et de la précision Une mesure communément utilisée est la précision exacte ou **R-précision**. Si la requête admet n documents pertinents, la précision exacte est la précision calculée à partir des n premiers documents de la liste ordonnée des documents restitués.
Une autre mesure, dérivée de la R-précision, est souvent utilisée : on fixe le nombre de documents retrouvés à plusieurs niveaux : top5, top10, top20, top 50, etc. Pour chaque niveau, on mesure la précision, et on calcule une moyenne de ces précisions sur toutes les requêtes. Cette manière de faire permet de repérer facilement les hautes précisions.
Enfin, des histogrammes de précision ou des tables résumé de statistiques peuvent permettre de parfaire la comparaison entre plusieurs algorithmes de recherche.

1.4.1.2 Mesures alternatives

L'inconvénient principal des mesures de rappel et de précision est que ces deux mesures représentent des aspects différents de l'ensemble des documents retrouvés. L'utilisation d'une mesure unique combinant les propriétés de ces deux mesures serait peut-être plus appropriée. Dans [142], S. Mizzaro fait une étude complète des différentes mesures d'évaluation utilisées en RI. Ceci permet de dégager d'autres mesures de performance relativement importantes.

Moyenne harmonique
La moyenne harmonique F combine le rappel et la précision en un nombre compris entre 0 et 1 [189].

$$F(j) = \frac{2}{\frac{1}{R(j)} + \frac{1}{P(j)}} \tag{1.25}$$

où $P(j)$ et $R(j)$ sont respectivement la précision et le rappel du j^{ieme} document renvoyé par le SRI. Si $F(j) = 0$, aucun document pertinent n'a été renvoyé,

et si $F(j) = 1$, tous les documents renvoyés sont pertinents. Ainsi, la moyenne harmonique a des valeurs élevées uniquement lorsque les taux de rappel **et** de précision sont élevés.

Mesure E

La mesure E permet à l'utilisateur de spécifier s'il est plus intéressé par le rappel ou la précision. [211].

$$F(j) = 1 - \frac{1 + b^2}{\frac{b^2}{r(j)} + \frac{1}{P(j)}} \tag{1.26}$$

où $P(j)$ et $r(j)$ sont respectivement la précision et le rappel du j^{ieme} document renvoyé par le SRI. La variable b mesure l'importance relative de la précision ou du rappel. Si $b > 1$, on privilégie la précision et si $b < 1$ on privilégie le rappel.

Mesures orientées utilisateur

Des utilisateurs peuvent avoir une interprétation différente sur la pertinence de tel ou tel document. Pour remédier à ce problème, Korhage a proposé des mesures de pertinences orientées utilisateur [116](figure 1.12).

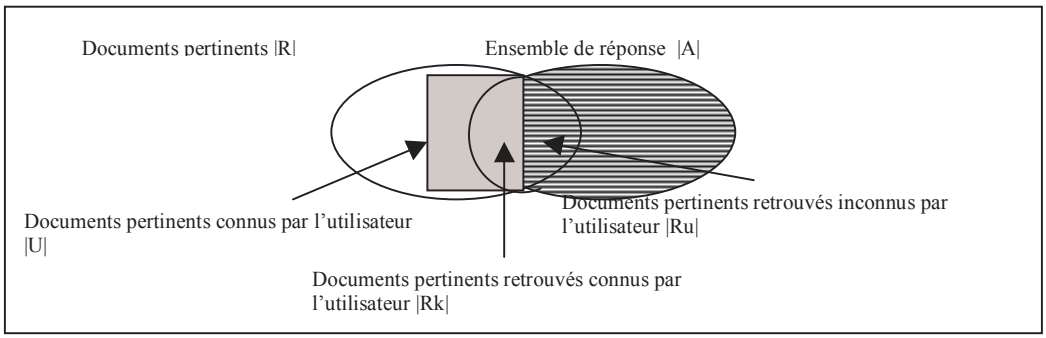


FIG. 1.12 – Mesures de performances orientées utilisateur

– *Ratio de couverture (Coverage ratio)* : fraction des documents connus (par l'utilisateur) comme pertinents et qui ont été retrouvés :

$$coverage = \frac{|Rk|}{|U|} \tag{1.27}$$

– *Ratio de nouveauté (Novelty ratio)* : fraction des documents retrouvés qui étaient inconnus à l'utilisateur

$$novelty = \frac{|Ru|}{|Ru| + |Rk|} \tag{1.28}$$

- *Rappel relatif* : ratio entre le nombre de documents pertinents trouvés par le système et le nombre de documents pertinents que l'utilisateur espérait trouver
- *Effort de rappel* : ratio entre le nombre de documents pertinents que l'utilisateur espérait trouver et le nombre de documents examinés dans l'espoir de trouver des documents pertinents.

D'autres mesures moins conventionnelles peuvent aussi servir à évaluer la performance d'un SRI. On peut par exemple citer la *satisfaction* (qui ne prend en compte que les documents pertinents) et la *frustration* (qui ne prend en compte que les documents non pertinents) [116].

1.4.2 Collections de référence - Un exemple : TREC

Les mesures d'évaluation des SRI permettent certes de les comparer, mais encore faut-il que les évaluations soient faites sur les collections de documents. De nombreux projets basés sur des corpus d'évaluation se multiplient depuis les années 70. On peut par exemple citer la Collection CACM, la Collection ISI, ou encore la campagne CLEF (Cross Language Evaluation Forum)[3].
Le projet le plus ambitieux est sans aucun doute le projet d'évaluation TREC (Text Retrieval Conference)[4] de la DARPA[5]. La campagne d'évaluation TREC, co-organisée par le NIST[6] et la DARPA, a commencé en 1992. Elle a pour but d'encourager le recherche documentaire basée sur de grandes collections de test, tout en fournissant l'infrastructure nécessaire pour l'évaluation des méthodologies de recherche et de filtrage d'information. Pour chaque session de TREC, un ensemble de documents et de requêtes (les "topics") est fourni. Les participants exploitent leurs propres systèmes de recherche sur les données et renvoient à NIST une liste ordonnée de documents.
NIST évalue ensuite les résultats comme suit. L'ensemble des documents pertinents pour chaque requête est obtenu en prenant les K documents les mieux classés des différents SRI participant à la campagne d'évaluation. Ces documents sont ensuite montrés à des juges qui décident finalement de la pertinence de chaque document. Les participants à TREC disposent de la liste des documents pertinents pour chaque requête, et peuvent ainsi évaluer les performances de leurs SRI respectifs.
De TREC-1 à TREC-6, les recherches étaient centrées sur deux tâches principales : la tâche de routage et la tâche adhoc. La tâche ad hoc est constituée d'un ensemble de nouvelles requêtes qui sont lancées sur une collection de documents fixés, et la tâche de routage est composée d'un ensemble de requêtes fixes lancées sur une collection de documents en évolution perpétuelle. Autour

[3]http ://www.clef-campaign.org
[4]http ://trec.nist.gov
[5]Defense Advanced Research Project Agency
[6]National Institue of Standards and Technology (www.nist.gov)

de ces tâches, bon nombre de pistes ont été explorées, et de nouvelles tâches sont apparues. On peut par exemple citer des évaluations de recherche de documents écrits dans une autre langue que l'anglais (espagnol, français, ou encore chinois, arabe)(à partir de 1994), des évaluations de recherche à travers des langages multiples, des évaluations sur de très grands corpus (tâche Terabyte en 2004), ou des évaluations portant sur des aspects plus diversifiés (la tâche interactive depuis 1994, la tâche QA (question-réponse) en 1999...), ou encore des évaluations de recherche sur des vidéos (depuis 2001) ou des documents Web (depuis 1999).

Les différents moyens d'évaluation que nous venons de décrire pour évaluer les SRI concernent essentiellement l'aspect algorithmique des différents modèles. Il ne faut cependant pas perdre de vue le fait que ces modèles sont destinés à des utilisateurs et que la plus importante des évaluations est celle faite par ces derniers !

1.5 Conclusion : Vers la Recherche d'Information Structurée

Dans ce chapitre, nous avons passé en revue les méthodes, modèles et algorithmes fondamentaux utilisés en recherche d'information "traditionnelle". Chacun de ces modèles ou stratégies participe à la résolution des problèmes inhérents à la recherche d'information, à savoir la création d'une base documentaire, son indexation, et le choix du modèle utilisé pour la recherche d'une part, et un besoin d'information utilisateur, la formulation de requête et le retour éventuel de pertinence d'autre part. L'ensemble de ces points contribue à la performance finale des systèmes.
Les SRI que nous avons décrits ici fonctionnent généralement sur des documents multi-formats (structurés, non structurés, balisés ou non balisés). Ils n'exploitent cependant que le contenu sémantique des documents, c'est à dire leur texte.
Devant le nombre croissant de documents semi-structurés ou structurés mis à disposition, de nouveaux modèles cherchent à tirer parti de l'information structurelle très dense contenue dans ce type de documents, en combinant cette dernière avec l'information de contenu. Dans le chapitre suivant, nous nous intéressons particulièrement à ces nouveaux modèles pour la recherche d'information structurée. Les principales problématiques qu'ils cherchent à résoudre (interrogation, indexation, identification des éléments pertinents) sont comparables à celles soulevées dans la RI traditionnelle, mais doivent être abordées en ajoutant la dimension structurelle à la dimension de contenu.

Chapitre 2

Recherche d'Information Structurée

2.1 Introduction

Le type des documents mis à disposition des utilisateurs évolue : du simple document texte "plat", on assiste aujourd'hui à la généralisation des documents structurés ou semi-structurés. Des formats tels que SGML (*Standard Generalized Markup Language*)[88] ou encore XML (*eXtensible Markup Language*) [215, 26], conçus à l'origine pour faciliter l'échange et la standardisation des données, voient leur importance augmenter grâce à l'expansion d'Internet [140].
Du point de vue des systèmes de Recherche d'Information, l'accès à ce type de documents soulève de nouvelles problématiques liées à la co-existence de l'information structurelle et de l'information de contenu. La prise en compte de la dimension structurelle devrait permettre de mieux répondre aux différents besoins des utilisateurs. Elle réactualise cependant le problème de la granularité de l'information à retourner.
Ce chapitre a pour objectif de présenter les différentes problématiques soulevées par la RI structurée, ainsi que les différentes solutions proposées dans la littérature.

Nous commençons tout d'abord par définir la notion de structure et présenter les documents semi-structurés (section 2.2), en nous attardant plus particulièrement sur XML. Nous présentons ensuite dans la section 2.3 les différents défis soulevés par la recherche d'information dans les documents semi-structurés. La section 2.3 nons permet de distinguer les grands types d'approches proposées dans la littérature, à savoir les approches orientées BD et les approches orientées

RI. Afin d'utiliser au mieux les propriétés des documents semi-structurés, de nouvelles techniques d'indexation, présentées en section 2.5, ainsi que de nouveaux langages d'interrogation prenant en compte la structure (section 2.6) sont utilisés. Nous décrivons ensuite les différents modèles de recherche proposés dans la littérature (section 2.7). Ces modèles de recherche visent à répondre à des requêtes basées sur le contenu seul ou à des requêtes basées sur le contenu et la structure . La section 2.8 présente enfin les approches utilisées pour l'évaluation des systèmes.

2.2 Documents semi-structurés

Comme nous l'avons vu dans le chapitre 1, un document peut se définir par le *fond* et par la *forme*. La forme peut être construite grâce à des éléments de structure et de présentation. Le fond quant à lui, repose sur des élements de sémantique, éventuellement complétés par des éléments de structure [83]. De multiples approches intégrant ces notions ont été définies pour gérer des documents électroniques. On peut par exemple citer SGML (*Standard Generalized Markup Language*) pour la structuration ou encore HTML (*HypertText Markup Language*) pour la présentation. D'autres formats mêlent quant à eux le fond et la forme. Nous présentons dans la section suivante un rapide historique de ces formats, pour nous attarder ensuite sur la notion de structure, et le langage de balisage connaissant la plus grande expansion, à savoir XML (*eXtensible Markup Language*).

2.2.1 Historique des langages de balisage

L'ajout d'annotations ou de notes dans un texte est pour bon nombre de lecteurs une attitude naturelle. Le marquage électronique relève du même esprit : il s'agit d'insérer, non plus à la surface d'une page, mais dans un fichier électronique (que l'on peut considérer comme linéaire) des informations liées au texte lui-même, mais n'en faisant pas directement partie [11].
Les premiers marquages électroniques concernent des commandes typographiques (passer en gras, en italique, changement de police de caractères,...) et aboutissent à des formats tels que RTF (*Rich Text Format*) ou MIF (*Maker Interchange Format*). Ces marquages sont faits à partir de *balises*[1], qui permettent d'indiquer des changements d'états. Dans le cas de RTF et MIF, on parle de *balisage spécifique*, car le nom et le nombre des balises sont fixés.

[1]On trouvera une définition détaillée de la notion de balise dans la section suivante.

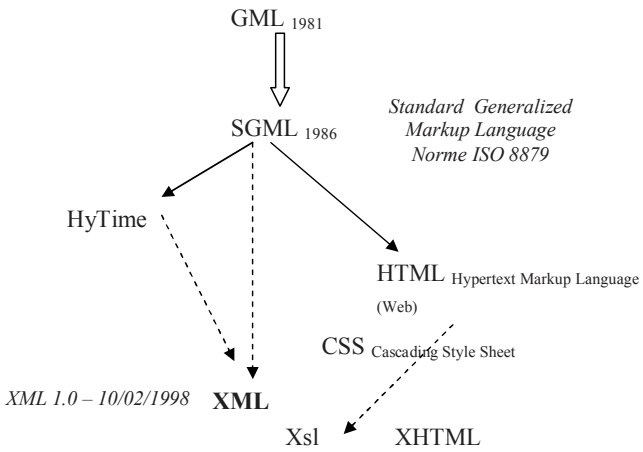


FIG. 2.1 – Historique des langages de balisage, extrait de [44]

En 1981, Charles Goldfarb, Edward Mosher et Raymond Lorie inventent le premier langage de *balisage générique*, GML (*Generalized Markup Language*). GML doit permettre aux sous-systèmes d'édition, de formatage et de repérage de partager les mêmes documents et non pas de créer un fichier destiné à l'édition, un autre au repérage, etc. *Pour la première fois, un langage rend indépendant les contenus par rapport aux outils manipulant ces contenus.* Le créateur de documents GML peut définir ses propres balises, selon ses besoins et les besoins de ses applications. De plus, GML introduit le concept de types de documents formellement définis comprenant une structure d'éléments imbriqués.
En 1986, au travers de la norme ISO 8879, GML evolue en SGML (*Standard Generalized Markup Language*)[88]. On voit alors apparaître des concepts importants comme la possibilité de prévoir l'ordre d'apparition d'éléments dans la structure d'un document. Les documents sont balisés conformément à une grammaire, la DTD (*Document Type Definition*) : ceci implique la notion de validité d'un document. De plus, la conception ou le choix d'une DTD permettent d'ajouter un balisage sémantique du fond du document. SGML, malgré ses nombreux avantages, souffre cependant de quelques inconvénients : la mise en oeuvre de documents respectant ce format est lourde et complexe, et la création de liens hypertextes est possible, mais reste elle aussi complexe.
En 1992, le W3C (World Wide Web Consortium) propose HTML (*HyperText Markup Language*), un langage de balisage pour le Web. HTML est en fait une DTD de SGML, qui ne cesse d'évoluer depuis sa création. HTML est un langage simple, possédant des balises standardisées et permettant la mise en forme d'un texte. C'est un standard reconnu par tous les navigateurs, et par conséquent très populaire sur le Web. HTML mélange cependant le fond et la forme des documents, ce qui rend les mises à jours difficiles et qui surtout, mêle les données utiles (possédant une sémantique) et les données de mise en forme. De plus, il n'est pas un outil idéal pour l'échange de données.

En 1998, le W3C publie une recommandation officielle concernant le langage XML (*eXtensible Markup Language*) [215]. XML se veut être un langage séparant la structure et la présentation. Il s'agit en fait d'un sous-ensemble de SGML, idéal pour l'échange de données semi-structurées. XML est en passe de devenir le format de documents semi-structurés le plus répandu : sa présence et son utilisation sur le Web se font de plus en plus importantes, tant dans les domaines génériques que dans les zones géographiques dans lesquelles il apparaît [140]. On trouvera les caractéristiques d'XML ainsi que les différents standards associés dans les sections 2.2.2 et 2.2.3.

2.2.2 La notion de structure

Comme nous l'avons vu dans le paragraphe précédent, la structure des documents est définie par des balises encadrant les portions d'informations. Nous présentons ici les notions de bases liées à la structure, s'appliquant aussi bien à des documents SGML que des documents XML.
Une *balise* (ou *tag* ou *label*) est une suite de caratères encadrés par "<" et ">", comme par exemple <*nombalise*>. Un *élément* est une unité sémantique identifiée, délimitée par des balises de début $< b >$ et de fin $< /b >$ [44], comme par exemple <*mabalise*> mon texte </*mabalise*>. Les éléments peuvent être imbriqués :

<*producteur*>
    <*nom*> Laurent Pinel </*nom*>
    <*adresse*>
        <*rue*>Bd Jean Brunhes</*rue*>
        <*ville*>Toulouse</*ville*>
    </*adresse*>
</*producteur*>

Les *attributs* des balises sont spécifiés au début de l'élément et après le nom de la balise, en utilisant la syntaxe *nomattribut=valeur*. Par exemple,
<*mabalise monattribut=mavaleur*>*texte* </*mabalise*>.

La DTD (*Document Type Definition*) associée au document contient l'ensemble des balises qu'il est possible d'inclure, ainsi que des relations de composition entre ces balises.
Contrairement à SGML, il n'est pas obligatoire d'associer une DTD à un document XML. Les documents XML doivent cependant être *bien formés*, c'est à dire qu'ils doivent respecter un certain nombre de règles lexicales et syntaxiques concernant l'encodage, le pairage des balises de début et de fin, la

déclaration des attributs, les commentaires, etc. On trouvera un exemple de document XML bien formé dans le tableau 2.1. Lorsque la grammaire d'un document est définie dans une DTD et que le document respecte cette DTD, on parle de document *valide*. Le tableau 2.2 présente une DTD correspondant au document de type *article* présenté dans le tableau 2.1. Notons enfin que l'on assiste aujourd'hui au développement d'une nouvelle forme de grammaire, qui permet de définir des éléments plus complexes et possède un typage des données plus riche, les *XML-schémas* [64].

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

TAB. 2.1 – Exemple de fichier XML *article.xml*

Une classe de document possède donc une structure *générique* définie par la DTD (ou le schéma XML) alors qu'un document instance de cette classe possède une structure *spécifique*. Des documents peuvent aussi posséder des structures *logiques* similaires (représentations hiérarchiques et sémantiques similaires) tout en ne suivant pas la même DTD. Par exemple, les deux docu-

```
<?xml version="1.0" ?>
<!--DTD pour l'exemple d'article -->
<!ELEMENT article (en-tete, corps)>
<!ATTLIST article
        annee CDATA #REQUIRED
>
<!ELEMENT en-tete (auteur+, titre)>
<!ELEMENT auteur (#PCDATA)>
<!ELEMENT titre (#PCDATA)>
<!ELEMENT corps (section+)>
<!ELEMENT section (sous-titre, par)>
<!ELEMENT sous-titre (#PCDATA)>
<!ELEMENT par (#PCDATA)>
```

TAB. 2.2 – Exemple de DTD correspondant à *article.xml*

ments du tableau 2.3 ont des structures logiques similaires. Ces documents nous permettent aussi de constater que les balises d'un document ne sont malheureusement pas forcément porteuses de sémantique.

```
<RESTAURANT ID="1">
<NOM> Le RU 1 </>
<GUIDES>
        <PREMIER> Michelin
        <NOTE> ****</NOTE>
        <REF>2005, page 52</REF>
        </PREMIER>
        <SECOND> Le Guide du Routard
        <NOTE>A</NOTE>
        <REF>2002, page 126</REF>
        </SECOND>
</RESTAURANT>
```

```
<RESTAURANT ID="2">
<NOM> Le RU 2 </>
<GUIDES>
        <G id="1"> Michelin
        <N> **</N>
        <M>2005, page 53</M>
        </G>
        <G id="2"> Le Guide du Routard
        <N>B</N>
        <M>2002, page 127</M>
        </G>
</RESTAURANT>
```

TAB. 2.3 – Exemple de documents XML possédant des structures logiques similaires

Les formats SGML et XML permettent de produire des documents *structurés* ou *semi-structurés*.
Les documents *structurés* possèdent une structure régulière, ne contiennent pas d'éléments mixtes (c'est à dire d'éléments contenant du texte ET d'autres éléments) et l'ordre des différents éléments qu'ils contiennent est généralement non significatif.
Les documents *semi-structurés* quant à eux sont des documents qui possèdent une structure flexible et des contenus hétérogènes. La modification, l'ajout ou

la suppression d'une donnée entraîne une modification de la structure de l'ensemble. Abiteboul, dans [1], donne la définition suivante :

> *Par semi-structuré, nous signifions que même si les données possèdent une structure, celle-ci n'est pas aussi rigide, aussi régulière ou complète que la structure requise par les systèmes de gestion de bases de données traditionnels.*

On trouve une autre définition dans [139] :

> *Nous appelons [...] donnée semi-structurée la donnée qui n'est (d'un certain point de vue) ni une donnée brute ni une donnée strictement typée .*

Dans notre contexte, nous nous intéressons plus particulièrement à la recherche d'information dans des documents semi-structurés, les documents structurés servant plutôt à conserver des données au sens bases de données. Par abus de langage, on parlera cependant de *RI structurée*. Le format XML nous permettra d'illustrer nos propos.

2.2.3 La galaxie XML : extraits

XML, utilisé dans la recherche d'information, permettrait d'effectuer des requêtes très fines sur le contenu ou sur des fragments de documents. Une galaxie de standards ou de recommendations a émergé conjointement à XML afin de définir des outils et des applications autour du langage. Parmi les plus connus et les plus susceptibles d'aider à la recherche d'information, citons les mécanismes de base pour adresser des éléments dans des documents XML (XPath), pour traiter les documents XML (DOM et SAX), pour présenter et transformer les contenus XML (XSL), les espaces de nom, XLink et XPpointer pour la gestion des liens, RDF,...Nous nous contentons ici de détailler les deux standards devenus aujourd'hui indissociables d'XML, DOM et XPath. La compréhension de ces standards est très utile pour la suite du mémoire, tant pour la suite de ce chapitre que pour la partie décrivant notre contribution. On trouvera une présentation détaillée des autres outils liés à XML en Annexe A.

2.2.3.1 DOM (Document Object Model)

XML n'est pas un langage de programmation, mais un métalangage permettant de représenter des données. Pour traiter ces données, il faut disposer d'un *analyseur*, encore appelé *parser* en anglais. Il existe deux types d'analyseurs : le parser SAX (*Simple API for XML*) produisant un flux d'évènements et le parser DOM (*Document Object Model*) produisant un graphe d'objets en mémoire. Le premier est standardisé par le groupe XML-DEV, le second par

le W3C [214].
L'API DOM est basée sur une structure d'objets pour représenter un document balisé. L'analyseur génère un arbre d'objets reliés entre eux, chaque objet représentant un atome du document XML. On trouvera un exemple d'arbre DOM sur la figure 2.2. Un tel arbre se compose d'une *racine* Document, de *noeuds internes* représentant les éléments ou les attributs, et de *noeuds feuilles* contenant les valeurs d'éléments ou d'attributs. Dans la suite de ce mémoire, nous représenterons les documents XML sous cette forme, et utiliserons indifféremment les termes *éléments* ou *noeuds* pour désigner des sous-arbres de documents XML.

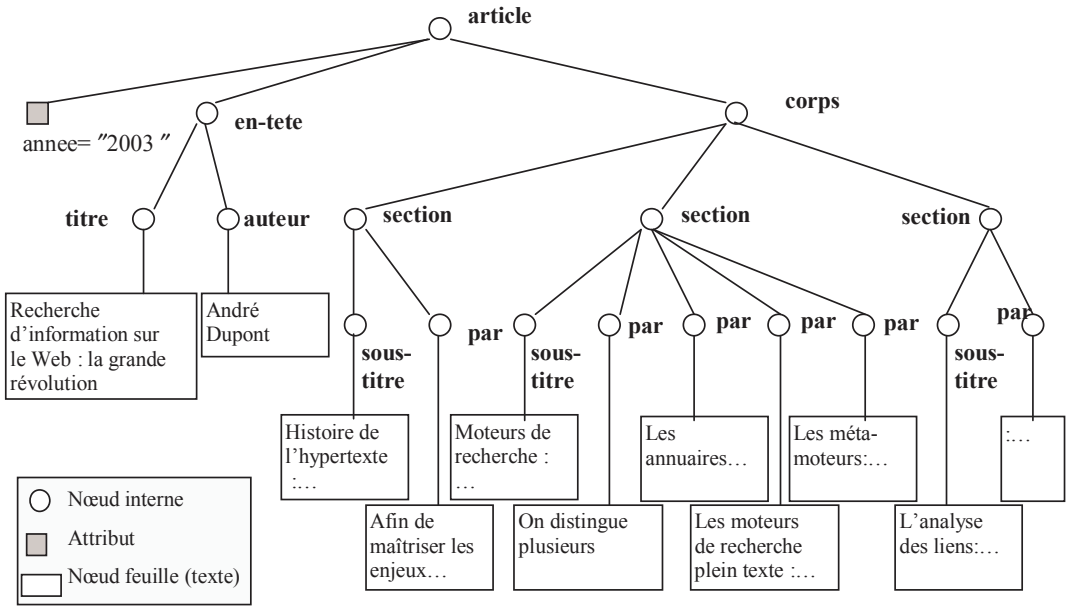


FIG. 2.2 – Exemple d'arbre DOM correspondant au document du tableau 2.1

DOM permet aussi de naviguer facilement dans des arbres existants, mais les performances restent limitées, car la place mémoire nécessaire est importante (due à l'éclatement du documents en atomes).

2.2.3.2 XPath

Xpath est un langage de spécification d'expressions régulières décrivant un chemin ou une famille de chemins dans une arborescence XML. Xpath 1.0 est une recommandation du W3C [45].
Il s'agit d'un langage permettant de sélectionner des sous-arbres d'un document XML. Il possède une syntaxe simple et non ambiguë et implémente les types usuels (chaines, nombres, booléens, variables, fonctions). Il permet aussi

de manipuler des noeuds et des ensembles de noeuds. Xpath est utilisé par Xpointer et XSLT (voir Annexe A).
XPath permet d'effectuer des expressions de chemins :

- recherche d'élements : *sous-titre*
- parent-enfant : *section/sous-titre*
- ancêtre-descendant : *article//section*
- racine du document : */article/**
- filtre sur la structure : *//article[section]*
- filtre sur le contenu : */article[@annee="2002" and auteur="Tim Bray"]*

XPath permet aussi d'effectuer la navigation dans un document XML selon différents axes, comme décrit sur la figure 2.3.

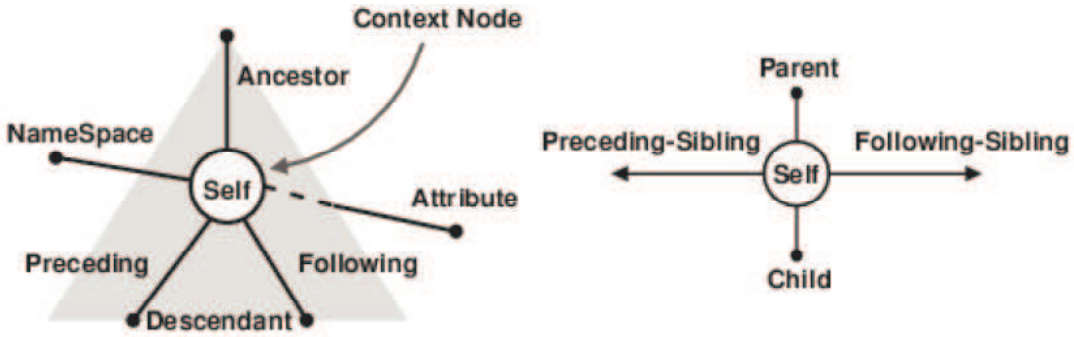


FIG. 2.3 – Axes de navigation XPath

Ces axes sont :

- **child : :** enfants du noeud contextuel
- **descendant : :** descendants du noeud contextuel
- **parent : :** parent du noeud contextuel
- **ancestor : :** ancêtre du noeud contextuel
- **following-sibling : :** tous les noeuds suivant le noeud contextuel et ayant le même noeud parent
- **preceding-sibling : :** tous les noeuds précédant le noeud contextuel et ayant le même noeud parent
- **following : :** tous les noeuds dans le même document que le noeud contextuel et étant après lui dans l'ordre du document (lecture séquentielle)
- **preceding : :** tous les noeuds dans le même document que le noeud contextuel et étant avant lui dans l'ordre du document
- **attribute : :** attributs du noeud contextuel
- **namespace : :** noeuds espaces de nom du noeud contextuel
- **self : :** le noeud contextuel lui-même
- **descendant-or-self : :** le noeud contextuel ou ses descendants
- **ancestor-or-self :** le noeud contextuel ou ses ancêtres

2.3 Recherche d'Information Structurée : problèmes et enjeux

2.3.1 L'unité d'information recherchée : la redéfinition de la notion de document

Le but des systèmes de recherche d'information est d'apporter une réponse non nécessairement exacte (au sens base de données) aux besoins en informations de leurs utilisateurs. Ces derniers s'intéressent rarement à une représentation ou à une structuration précise des collections consultées, "ils veulent du contenu". S'ils sont capables de préciser leur requête parce qu'ils connaissent la ou les collections interrogées, les réponses fournies par le système ne devront être que plus précises.

En recherche d'information traditionnelle, les SRI, tant dans leur modèle de représentation des données que dans les résultats qu'ils renvoient, traitent les granules des collections (c'est à dire les documents) dans leur globalité. Les notions de document *logique* et de document *physique* sont alors confondues. Cependant, un document possède souvent des contenus hétérogènes, et l'utilisateur doit alors aller chercher l'unité d'information pertinente à sa requête au milieu des autres thèmes abordés par le document. Une solution à ce problème serait de dissocier l'unité d'information *logique* renvoyée à l'utilisateur de l'unité d'information physique de la collection.
Les documents semi-structurés, en permettant le balisage des contenus des documents, réactualisent cette problématique, et permettent ainsi de traiter l'information avec une granularité plus fine. Le but des SRI traitant des documents semi-structurés est alors d'identifier des parties de documents les plus pertinentes à une requête donnée. Ceci nous amène à affiner le concept de granule, "*unité d'information*" renvoyée à l'utilisateur. Une unité d'information est un volume d'information *auto-explicatif*, c'est à dire que l'information contenue ne dépend pas d'une autre pour être comprise. Le but des SRI dans notre contexte est alors de renvoyer des unités d'information auto-explicatives à l'utilisateur, et non des points d'entrée dans les documents : les résultats renvoyés doivent se suffire à eux-même.
Dans le cadre des documents XML, l'unité d'information correspond à un noeud de l'arbre du document, c'est à dire à un *sous-arbre*. La pertinence d'un noeud vis-à-vis d'une requête est évaluée selon les deux notions suivantes : l'*exhaustivité* et la *spécificité* [41, 120].

> *On dit qu'une unité d'information est exhaustive à une requête si elle contient toutes les informations requises par la requête et qu'elle est spécifique si tout son contenu concerne la requête.*

Dans [41], on trouve "le principe de recherche dans les documents structurés" : *un système devrait toujours retrouver la partie la plus spécifique d'un document répondant à une requête.* Cette définition suppose que le système sélectionne d'abord des documents entiers répondant de manière exhaustive à une requête, puis extrait de ces documents les unités d'information les plus spécifiques. La plupart des SRI traitant les documents structurés permettent une recherche directe des unités d'information, sans passer au niveau de granularité document entier. Le principe de recherche dans les documents structurés pourrait donc être étendu ainsi : *un système devrait toujours retrouver l'unité d'information la plus exhaustive et spécifique répondant à une requête.* Dans des corpus de documents XML, chercher les noeuds les plus exhaustifs et spécifiques pour une requête revient donc à trouver les sous-arbres de taille minimale pertinents à la requête.

De part cette structure, l'utilisateur interrogeant des corpus de documents XML peut formuler deux types de requêtes, selon sa connaissance du corpus :

- des requêtes portant sur le contenu seul des unités d'information : ces requêtes sont composées de simples mots-clés, et l'utilisateur laisse le SRI décider de la granularité de l'information à renvoyer,
- des requêtes portant sur la structure et le contenu des unités d'information, dans lesquelles l'utilisateur spécifie des besoins précis sur certains éléments de structure. Dans ce type de requêtes, l'utilisateur peut utiliser les conditions de structure pour indiquer le type des éléments qu'il désire voir renvoyer, mais aussi plus simplement pour préciser son besoin.

Afin de permettre ces différentes recherches, les techniques de la recherche d'information traditionnelle doivent être adaptées ou de nouvelles méthodes doivent être proposées pour l'*indexation*, l'*interrogation* ou encore la *recherche* et le *tri* des unités d'information. Nous nous proposons de détailler ces différentes problématiques dans la section suivante.

2.3.2 Les problématiques spécifiques à la RI structurée

La problématique dans le cadre de l'indexation se situe essentiellement au niveau de l'information structurelle. Dans le cas des documents textes "plats", le contenu textuel des documents est traité afin de trouver et de pondérer les termes les plus représentatifs des documents. Dans le cas des documents semi-structurés, la dimension structurelle s'ajoute au contenu, et les questions suivantes se posent alors : que doit-on indexer de la structure des documents ? Comment relier cette structure au contenu même du document ? En fonction de quelle dimension (niveau éléments, documents, collection) doit-on pondérer

les termes d'indexation ?

Considérons à présent la problématique de l'interrogation des documents. Il s'agit ici de permettre à l'utilisateur d'exprimer des besoins diversifiés (concernant le contenu des documents et/ou la structure), et ce de manière simple.

La dernière problématique concerne les modèles de recherche et de tri des unités d'information. La problématique traditionnelle liée à l'évaluation de la pertinence d'une information vis-à-vis d'une requête reste d'actualité, mais elle se complique et implique d'autres questions dans le cadre des documents XML, notemment en ce qui concerne la structure. Les requêtes orientées contenu, qui sont de loin les plus simples pour l'utilisateur, imposent au SRI de décider la granularité appropriée de l'information à renvoyer.
Dans le cadre des requêtes orientées contenu et structure, deux cas sont possibles. Tout d'abord, l'utilisateur peut spécifier le type des éléments à renvoyer par le système. D'autres notions de pertinence entrent alors en jeu. La dimension de spécificité n'a plus réellement de sens, puisque l'utilisateur précise la granularité de l'information qu'il désire. Cependant, le contenu des éléments de structure ainsi que les expressions de chemin présentes dans la requête doivent pouvoir être traitées de manière vague. En d'autres termes, la pertinence des informations structurelles doit pouvoir être évaluée, et l'arbre de la requête et l'arbre du document doivent pouvoir être comparés de façon non stricte. Le second cas concerne les requêtes pour lesquelles l'utilisateur exprime des conditions sur la structure des documents, mais sans préciser ce qu'il recherche exactement. Si le problème de l'évaluation de la pertinence des informations structurelles se pose de nouveau, vient s'y ajouter, comme dans les requêtes orientées contenu, celui de la granularité de l'information à renvoyer.

Nous nous proposons de présenter dans la suite de cette section les premières approches proposées dans la littérature pour répondre à quelques unes de ces problématiques. Nous nous attarderons ensuite sur les deux grands types d'approches spécifiques proposées pour la recherche dans des documents XML, à savoir les approches orientées Bases de Données et les approches orientées Recherche d'Information

2.3.3 Les précurseurs

Bien avant l'apparition d'XML, des travaux concernant la granularité de l'information à renvoyer à l'utilisateur ont été présentés. Ces travaux ont cherché à découper un document textuel en entités plus petites : il s'agit des travaux

basés sur *la recherche de passage*. Plus tard, des travaux cherchant à exploiter la structure fixe des documents HTML ou bien les liens qu'ils contiennent ont été proposés dans le cadre de la *RI sur le Web*. Même si ces travaux ne sont pas directement applicables à la RI dans des documents XML (puisque l'information structurelle qu'ils contiennent sert à découper les unités d'information mais que cette information structurelle n'est pas fixe), ils sont les précurseurs de toutes les approches proposées.

2.3.3.1 La recherche de passages

Ces travaux cherchent à proposer une démarche permettant de caractériser des granules d'informations plus fins que les granules de la collection explorée. L'intérêt de considérer une granularité plus fine est de traiter des documents que l'on va supposer homogènes [9, 95].
Il existe un grand nombre de définitions de la notion de passage [175]. Dans [9, 221, 235, 106, 34], les auteurs proposent de renvoyer une partie de document en se basant sur un *découpage physique* du document, ou bien encore en utilisant l'*information structurelle*. Dans [9], les auteurs utilisent par exemple une segmentation en pages physiques (limitées en nombre de caractères).
Les passages peuvent aussi être vus comme des séquences de mots ou de phrases, limités par des changements de sujet [34, 138, 141, 95, 175, 178]. Les méthodes utilisées pour la détection de ces *segments thématiques* relèvent alors des méthodes statistiques ou probabilistes. Une des approches les plus connues de segmentation est celle proposée par Hearst [95]. L'élément d'information de base est la phrase, et pour chaque phrase donnée, sa similarité est calculée avec les k phrases précédentes et les k phrases suivantes. Si l'on trace une courbe des numéros de phrase en fonction des similarités, les brusques changements dans l'allure de la courbe délimitent les changements thématiques.
Enfin, dans [112], les passages sont vus comme des *fenêtres* d'un nombre fixé de termes, les fenêtres pouvant se recouvrir si nécessaire.
Les différentes approches de recherche par passage sont relativement simples à mettre en place et efficaces, mais leur application reste limitée aux seuls documents texte et les méthodes ne s'appliquent qu'à des documents ayant des tailles homogènes.

2.3.3.2 RI sur le Web

Utilisation des liens La spécification des liens du Web peut contenir de nombreuses informations implicites qui peuvent aider pour ordonner ou filtrer des pages Web. En particulier, un lien d'une page A à une page B peut être considéré, dans la plupart des cas, comme une recommandation de la page B par l'auteur de la page A. Ainsi, les liens, dont le but premier est de faciliter

la navigation à l'intérieur d'un site, peuvent aussi être vus comme des liens de proximité sémantique entre pages Web [39].
L'algorithme utilisé dans [36] fait partie des tous premiers à exploiter la topologie des liens pour aider aux classement des pages. Les liens sont utilisés pour essayer de contrer les problèmes liés au "vocabulary problem" [81], c'est à dire à la difficulté pour les utilisateurs à formuler leur besoin en information.
Brin et Page [29] utilisent la notion de *propagation de popularité* pour construire leur algorithme *Page Rank*, utilisé dans le célèbre moteur de recherche Google. La propagation de popularité (ou "macroscopic distillation" [38]) provient initialement de l'analyse de citations ou de co-citations dans la littérature scientifique [220]. Au lieu de modifier directement l'index des documents, la méthode consiste à mettre en avant les documents qui jouent un rôle particulier dans le réseau de liens. Cette approche s'avère très efficace en marketing, mais a montré ses limites, notamment lors des campagnes d'évaluation TREC 2001-2003 [203].
L'algorithme HITS (*Hyperlinked Induced Topic Search*) [113] améliore la propagation de popularité en prenant en compte la pertinence des pages : "Une page référencée par un grand nombre de pages *pertinentes* est une bonne page", ou "une page qui référence un grand nombre de pages *pertinentes* est une bonne page". Contrairement à la technique du PageRank, qui assigne un score global à chaque page, l'algorithme HITS est une technique d'ordonnancement dépendante de la requête. De plus, au lieu de donner un simple score, l'algorithme en donne deux : les scores d'*autorité* et de *rayonnement*.
Enfin, dans [153], les auteurs montrent que l'utilisation des liens dans un modèle d'argumentation probabiliste (*PAS : Probabilistic Argumentation System*) permet d'améliorer significativement le classement des documents.

Utilisation des méta-balises Gloria Bordogna et Gabriella Pasi [21] proposent un modèle flexible d'interrogation de documents Web. Ce modèle permet aux utilisateurs de personnaliser la représentation des documents structurés. L'idée principale est d'exploiter la structure logique du document dans le calcul des poids des termes de l'index. Dans une première phase, les termes sont indexés en fonction des différentes sections du document. Chaque section possède une fonction d'appartenance floue, et les poids des termes sont calculés dans les sections principales grâce à des fonctions d'agrégation. Des quantifieurs linguistiques définis à l'aide d'OWA (*Ordered Weighted Averaging Operators*[228]) sont ensuite associés aux fonctions d'agrégation.

2.3.4 Les approches spécifiques

Dans la suite du chapitre, nous nous proposons de décrire en détail les méthodes proposées dans la littérature pour l'indexation, l'interrogation, la recherche et le tri des documents XML. Ces méthodes peuvent être divisées en deux courants principaux [75] :

- ***L'approche orientée données*** voit les documents XML comme des collections de données, typées et relativement homogènes. Elle utilise des techniques développées par la communauté des *bases de données*.
- ***L'approche orientée documents*** se focalise sur des applications considérant les documents structurés d'une manière traditionnelle, c'est à dire que les balises servent uniquement à décrire la structure logique des documents. Cette approche a quant à elle été prise en charge par la communauté de la *recherche d'information*.

Alors que les deux communautés sont historiquement à l'origine de méthodes bien dissociées, la frontière entre les différentes approches pour la recherche dans des documents XML tend aujourd'hui à s'estomper. Citons à titre d'exemple le dernier Workshop RI+XML organisé dans le cadre de la conférence internationale SIGIR 2004, qui a finalement été couplé avec le workshop BD+RI.
Nous nous proposons ici de lister les solutions proposées par les deux communautés pour l'indexation, l'interrogation et l'appariement requête-unités d'information.

En ce qui concerne l'*indexation*, la problématique réside essentiellement en l'extraction des clés de recherche, à savoir les termes les plus représentatifs des documents ainsi que l'information structurelle qu'ils contiennent.
Les approches orientées BD confondent les notions d'indexation et de stockage. *Toute* l'information textuelle et structurelle des documents est ainsi stockée au sein de tables dans des bases de données. Ceci pose particulièrement problème pour les recherches sur le contenu textuel des documents, puisque ce dernier est indexé en tant que chaîne de caractères, et non sous forme de termes indépendants. Ces approches proposent néanmoins des schémas de stockage optimaux pour la structure des documents. Cette dernière peut être reflétée dans le schéma de la base de données, ou bien être stockée de manière générique dans des tables particulières.
Les approches orientées RI utilisent des techniques traditionnelles pour l'extraction des termes d'indexation, mais de nouvelles problématiques sont soulevées concernant la structure. Que doit-on indexer de la structure des documents ? Comment relier cette structure au contenu même du document ?

Considérons maintenant les *langages d'interrogation*, dont la grande majorité a été proposée par la communauté des bases de données. Ces langages d'interrogation doivent permettre à l'utilisateur d'exprimer des conditions sur

le contenu et/ou la structure des documents.
La communauté BD a été historiquement la première a proposer des langages pour l'interrogation des documents XML. Ces langages, presque exclusivement basés sur des syntaxes proches de SQL, permettent à l'utilisateur d'exprimer des conditions très précises sur la structure des documents. L'expression de conditions sur les chemins est par exemple permise. Des prédicats de type "*contains*" sont aussi proposés pour effectuer des recherches sur le contenu textuel. Cependant, ces dernières conditions doivent toujours porter sur des conditions de structure bien définies, et l'utilisateur doit de plus spécifier le type d'élément qu'il désire voir retourné par le système, alors qu'il n'a pas forcément d'idée précise sur la question.
Les approches orientées RI cherchent quant à elles à simplifier ces langages en ce qui concerne les conditions de structure, tout en proposant de nouvelles fonctionnalités concernant la recherche sur le contenu (utilisation d'un prédicat "*about*" pour remplacer le prédicat "*contains*", ou bien encore d'opérateurs booléens dans les conditions de contenu).

La dernière problématique concerne enfin *le traitement de la requête*.
Les approches orientées BD évaluent de façon *exacte* des expressions du type *attribut* = *valeur*. Le traitement des requêtes est donc fait de manière booléenne et il n'est pas possible de renvoyer à l'utilisateur une liste de résultats triés en fonction de leur pertinence.
Les approches orientées RI cherchent quant à elle à évaluer le degré de similarité entre la requête et les unités d'informations et attribuent à ces dernières un score de *pertinence*. L'intérêt est double : tout d'abord sélectionner les unités d'informations qui répondent *au mieux* au besoin de l'utilisateur, et lui proposer ensuite une liste triée de résultats.

D'une manière générale et comme nous allons le voir dans la suite du document, les solutions proposées par la communauté RI peuvent être utilisées comme "sur-couche" aux solutions orientées BD. Cette sur-couche sert essentiellement à intégrer la notion de pertinence dans la recherche, en complétant les approches proposées par la communauté BD pour le stockage et l'interrogation des documents.

En résumé et comme le montre la figure 2.4, les approches orientées BD peuvent servir de socle pour l'indexation et l'interrogation des documents. Les approches orientées RI complètent ces méthodes afin d'intégrer la notion de pertinence dans la recherche.
Nos travaux se positionnent clairement dans le domaine de la RI. Par conséquent, les problématiques détaillées dans la suite du document (à savoir l'indexation, l'interrogation et le traitement des requêtes) sont, dans le domaine du possible,

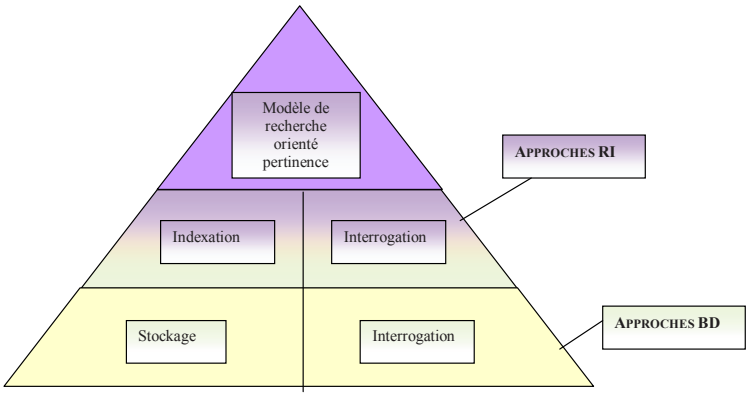


FIG. 2.4 – Domaines de compétence de la BD et de la RI

abordées sous cet angle.

2.4 Techniques d'indexation des documents semi-structurés

Comme nous l'avons vu au chapitre 1, le processus d'indexation consiste à extraire les clés de recherche des documents. Dans le cas des documents textes "plats", le contenu textuel des documents est traité afin de trouver et de pondérer les termes les plus représentatifs des documents. Dans le cas des documents semi-structurés, la dimension structurelle s'ajoute au contenu, et les questions suivantes se posent alors : que doit-on indexer de la structure des documents ? Comment relier cette structure au contenu même du document ? Comment pondérer les termes d'indexation, c'est à dire comment évaluer l'importance d'un terme au sein de l'élément, du document et de la collection ? Dans cette section, nous présentons les différentes approches proposées dans la littérature pour répondre à la problématique de l'indexation.

2.4.1 Que faut-il indexer ?

La façon la plus simple d'indexer des documents XML est bien sûr de les considérer comme des fichiers plats, et le processus d'indexation dans ce cas-là est similaire à celui utilisé en RI traditionnelle, c'est à dire qu'il consiste à sélectionner les termes importants du contenu textuel des documents. Cependant, aucune recherche sur la structure n'est plus possible, et les documents existent uniquement dans leur intégralité.

Un schéma d'indexation de documents XML devrait couvrir les aspects suivants :

1. permettre la reconstruction du document XML décomposé dans les structures de stockage ;
2. permettre le traitement des expressions de chemin sur la structure XML ;
3. accélérer la navigation dans des documents XML ;
4. autoriser le traitement de prédicats vagues et précis sur le contenu de documents XML ;
5. permettre la recherche par mots-clés

Par conséquent, la plupart des schémas d'indexation proposés dans la littérature redéfinissent la granularité du stockage et utilisent la structure des documents XML.

Les différentes approches utilisées pour indexer des documents semi-structurés peuvent être caractérisées selon deux dimensions [89] : le schéma de stockage des documents, et les types de tranformations possibles entre les documents XML et les structures de stockage.
Considérons d'abord le schéma de stockage. Deux approches sont possibles :

- les approches orientées Systèmes de Gestion de Bases de Données (SGBD) (ou *middleware de transformation*).
- les modèles de stockage XML natifs. Les SGBD natifs XML sont développés spécifiquement pour XML. A la différence des SGBD relationnels, ils stockent des documents complets ou des parties de documents dans des fichiers et ne réalisent pas de transformations (c'est à dire *mapping*) en tables. Un document étant un arbre, ils sont donc conçus pour gérer efficacement des arbres.

La seconde dimension représente les différents types de transformation (*mapping*) possibles entre les documents XML et les structures de stockage [230]. On distingue :

- **les approches de transformation basées sur un modèle** : ces approches créent un schéma générique de base de données qui reflète le modèle de données du format XML [124, 68, 93, 126]. Le schéma de l'index est fixe et connu à l'avance. Des variantes simples de ces approches prennent la représentation en graphe des documents XML et stockent les noeuds et les arcs du graphe dans une base de données. D'autres variantes utilisent la représentation du graphe la plus détaillée du modèle DOM, qui distingue des types de noeuds comme les éléments, les attributs ou les commentaires. Ces solutions, considérées comme extensibles, n'ont pas besoin de la DTD des documents pour les indexer, mais souvent, des fonctionnalités manquent aux index pour répondre à des requêtes portant sur des XPath précis, sur des hiérarchies ou encore sur des conditions de contenus relatives à des éléments de structure.

– **les approches de transformations basées sur la structure** : ces approches utilisent la structure logique des documents XML ou leur schéma. L'idée est de construire automatiquement un schéma d'index (qui correspondant le plus souvent à un schéma de base de données) prenant en compte la sémantique de l'application [63, 20, 59]. Contrairement aux approches de mapping basées sur des modèles, dans le cas de mapping basé sur la structure, des applications XML ayant des structures de documents XML différentes donneront lieu à des schémas d'index différents. Ces solutions sont non-extensibles, car les documents possédant des structures différentes ne peuvent pas être ajoutés.

Dans cette section, nous préférerons cependant adopter une autre classification. Même si l'indexation des informations de contenu et des informations structurelles sont étroitement liées, nous nous proposons de les décrire séparément, afin de mieux comprendre les différents enjeux soulevés par l'une et l'autre.

2.4.2 Indexation de l'information textuelle

Le processus d'indexation de la recherche d'information traditionnelle consiste à extraire les termes importants des documents. Cette problématique reste bien entendue d'actualité dans le cadre des documents structurés.
Pour les approches orientées BD, l'unité textuelle d'indexation est le texte complet des noeuds feuilles. Pour les approches orientées RI, il s'agit au contraire du terme, qui sera de plus pondéré afin de refléter son importance.
Quelle que soit l'unité textuelle d'information choisie, le problème de la portée des termes d'indexation se pose, et nous nous proposons de le détailler dans la section suivante.

2.4.2.1 Portée des termes d'indexation

Le problème de la portée des termes d'indexation est le suivant : Comment rattacher les termes à l'information structurelle ? Doit-on chercher à agréger le contenu des noeuds ou au contraire à indexer tous les contenus des noeuds séparément ? Ces deux solutions correspondent aux approches d'indexation dites *des sous-arbres imbriqués* et des *unités disjointes*.

Sous-arbres imbriqués Les approches de ce premier groupe considèrent que le texte complet de chaque noeud de l'index est un document atomique

[4, 192, 104] et propagent donc les termes des noeuds feuilles dans l'arbre des documents. En d'autres termes, ces approches *indexent tous les sous-arbres* (jugés potentiellement pertinents) des documents. Comme les documents XML possèdent une structure hiérarchique, les noeuds de l'index sont imbriqués les uns dans les autres et l'index contient de nombreuses informations redondantes. On trouvera une illustration de l'indexation de sous-arbres imbriqués sur la figure 2.5.

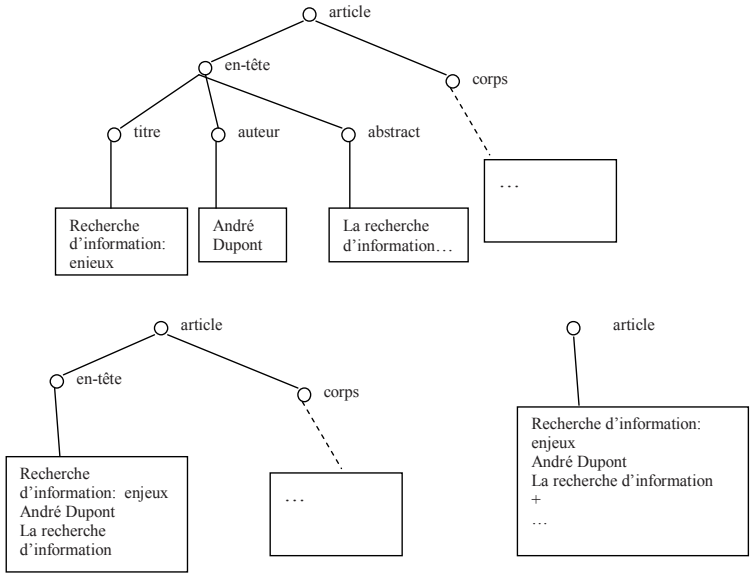


FIG. 2.5 – Indexation de sous-arbres imbriqués

Les termes "*andré dupont*" sont par exemple reliés aux noeuds */article/en-tête/auteur*, */article/en-tête*, et */article*.

Unités disjointes Dans ces approches, le document XML est décomposé en unités disjointes, de telle façon que le texte de chaque noeud de l'index est l'union d'une ou plus de ces parties disjointes [147, 75, 84, 111, 170, 12]. Les termes des noeuds feuilles sont uniquement reliés au noeud parent qui les contient.
Si on reprend en exemple l'arbre de la figure 2.5, les termes "*recherche d'information enjeux*" seront uniquement reliés au noeud */article/en-tête/titre*, les termes "*alain dupond*" au noeud */article/en-tête/auteur* et les termes "*la recherche d'information*" au noeud */article/en-tête/abstract*. Le noeud */article/en-tête* n'est quant à lui relié à aucun terme.

L'approche utilisée pour indexer le contenu des documents semi-structurés implique l'utilisation de méthodes différentes pour la recherche dans les documents. Nous reviendrons sur ces différentes méthodes dans la section 2.7.

2.4.2.2 Pondération des termes d'indexation

Les approches orientées BD se contentent de stocker le texte des documents comme un tout, c'est à dire sous forme de chaînes de caractères. Ce type d'approche pour l'information textuelle montre peu d'intérêt dans le cadre de la RI, puisque la pondération des termes n'est pas permise, et que par conséquent, seules des mesures de pertinence très simples pourront être calculées par le système (comme le nombre de termes communs entre la requête et l'élément).
Les approches orientées RI extraient les termes d'indexation selon des processus similaires à ceux utilisés en RI traditionnelle. La pondération de ces termes doit cependant être vue sous un nouvel angle. Alors qu'en RI traditionnelle, le poids d'un terme cherche à rendre compte de son importance de manière locale au sein du document et de manière globale au sein de la collection, s'ajoute en RI structurée l'importance du terme au niveau de l'élément qui le contient.
Les occurrences des termes ne suivent plus forcément une loi de Zipf [234, 89]. Le nombre de répétitions des termes peut être (très) réduit dans les documents XML et l'utilisation d'*idf* (Inverse Document Frequency) n'est pas forcément appropriée.
L'utilisation d'ief (Inverse Element Frequency) a été proposée par de nombreux auteurs [223, 90]. On trouvera des exemples d'adaptation des formules de pondération traditionnellement utilisées en RI à la RI structurée dans [205].
Dans [233], le calcul du poids des termes est influencé par le contexte (l'unité d'indexation) dans lequel ils apparaissent. Ce calcul de poids s'inspire de la méthode *tf-idf* qu'on applique aux balises. Ainsi, les auteurs définissent le *tf-itdf*(*Term Frequency - Inverse Tag and Document Frequency*), qui permet de calculer la force discriminatoire d'un terme t pour une balise b relative à un document d.
Dans [111], l' importance d'un terme dans un élément est l'agrégation (effectuée à l'aide d'opérateurs OWA [228]) de l'importance du terme dans le contenu du noeud même, dans le contenu de ses descendants, dans le contenu de ses voisins directs et dans le contenu des noeuds auquel il est relié. Le calcul du poids des termes est effectué au moment de l'indexation.
D'autres paramètres permettant d'évaluer l'importance des termes peuvent être pris en compte : la fréquence du terme au sein de l'élément bien sûr, mais aussi la fréquence du terme au sein du document, ou encore la longueur de l'élément et la longueur moyenne des éléments de la collection.

2.4.3 Indexation de l'information structurelle

L'information structurelle peut être indexée selon des granularités variées [130], c'est à dire que toute l'information structurelle n'est pas forcément utilisée dans le processus d'indexation. Parmi les approches proposées dans la

littérature, on distingue trois types d'approches pour l'indexation de l'information structurelle : l'indexation basée sur des champs, l'indexation basée sur des chemins, et enfin l'indexation basée sur des arbres. Nous nous proposons de les détailler ici, par ordre croissant de quantité d'information stockée.
Les schémas d'indexation de la structure que nous présentons dans la suite sont indépendants de l'*unité textuelle* d'indexation (*terme* ou bien *texte entier des feuilles*) choisie. En d'autres termes, les exemples que nous utilisons pour étayer nos propos peuvent être utilisés indifféremment pour traiter l'information textuelle selon des approches orientées RI ou bien orientées BD.

2.4.3.1 Indexation basée sur des champs

Il s'agit certainement de la méthode d'indexation semi-structurée prenant en compte la structure la plus simple. Un document est représenté comme un ensemble de champs (par exemple *titre*, *auteur*, *abstract*, etc) et de contenu associé à ces champs. Pour permettre une recherche restreinte à certains champs, les termes de l'index sont construits en combinant le nom du champ avec les termes du contenu, comme l'illustre la figure 2.6.

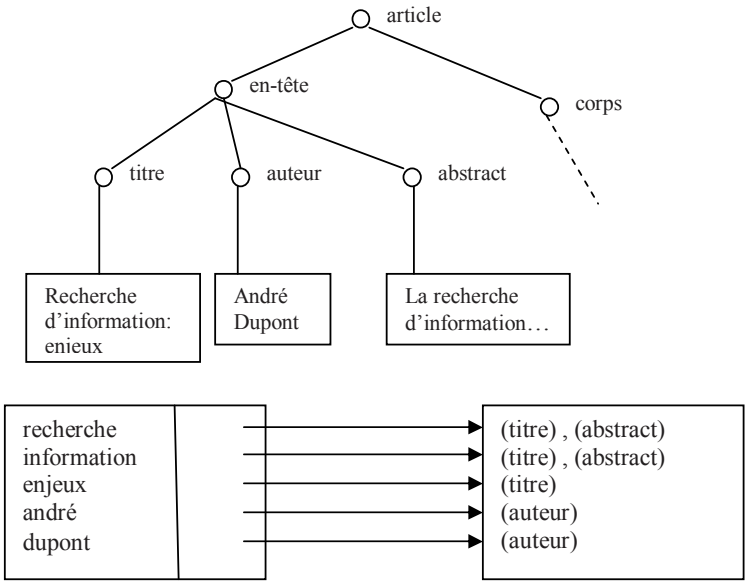


FIG. 2.6 – Exemple d'indexation basée sur des champs

Les différents champs d'un document peuvent être obtenus de plusieurs façons :

– Ils peuvent être codés en tant que méta-données dans les fichiers XML, par exemple en utilisant RDF.

– Dans le cas d'un document d'un format quelconque transformé en XML ; ils peuvent provenir du document dans son format original
– Ils peuvent être retrouvés à l'aide de différentes techniques d'extraction [94]
– Ils sont simplement extraits de la DTD ou du schéma XML associé.

2.4.3.2 Indexation basée sur des chemins

Les techniques d'indexation basées sur des chemins ont pour but de retrouver rapidement des documents ayant des valeurs connues pour certains éléments ou attributs. Il s'agit aussi de faciliter la navigation de façon à résoudre efficacement des expressions Xpath et utiliser des index pleins textes sur les contenus. En conséquence, les solutions proposées utilisent des index de chemins, c'est à dire des index donnant pour chaque valeur répertoriée d'un chemin de balises (de type Xpath) la liste des documents répondants contenant un élément atteignable par ce chemin et ayant cette valeur. On trouvera une illustration de l'indexation basée sur des chemins sur la figure 2.7.

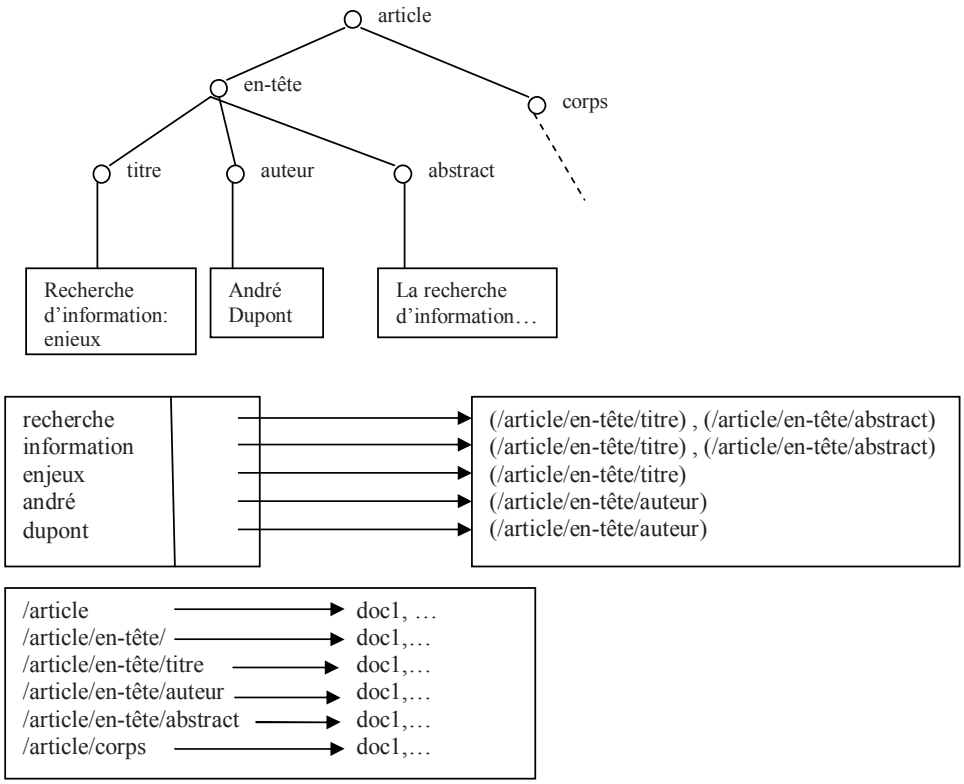


FIG. 2.7 – Exemple d'indexation basée sur des chemins

Parmi les approches utilisant une indexation basée sur des chemins, on

peut citer Natix [105] ou bien encore InfonyteDB [97]. Dans ces approches cependant, il devient difficile de retrouver les relations ancêtres-descendants entre les différents noeuds des documents. Les approches d'indexation basées sur des arbres le permettent quant à elles.

2.4.3.3 Indexation basée sur des arbres

La figure 2.8 donne un exemple d'indexation basée sur des arbres. Les noeuds de l'arbre sont numérotés dans les index de façon à pouvoir reconstruire la structure arborescente des documents.

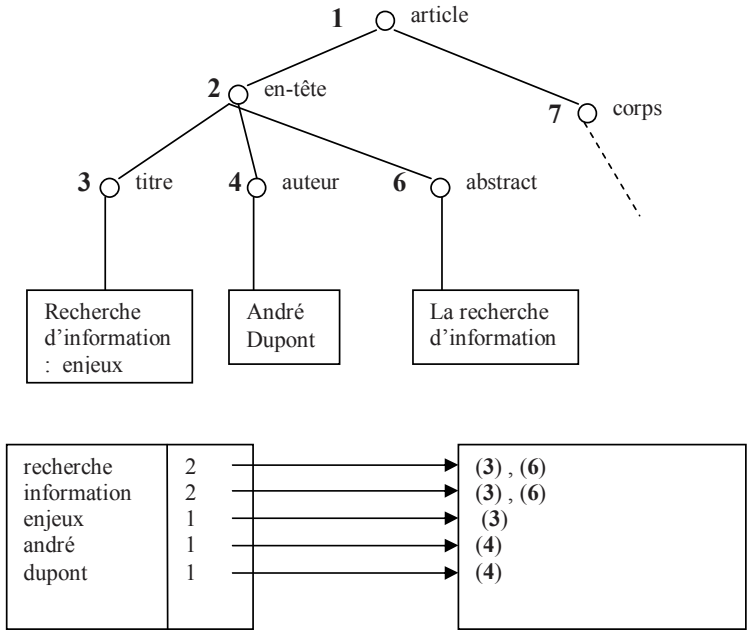


FIG. 2.8 – Exemple d'indexation basée sur des arbres

Dans [124], on trouve plusieurs schémas pour des documents SGML, schémas aussi valables pour les documents XML. Les auteurs démontrent que l'index ANOR (*inverted index for All NOdes without Replication*) est celui obtenant les meilleurs performances. Les documents structurés du corpus sont agrégés en un seul arbre de document. Cet arbre de document est ensuite interprété comme un k-arbre virtuel (certains noeuds peuvent ne pas exister) et de cette manière, un identifiant unique (UID) peut être attribué à chaque noeud. Le parent du noeud courant peut être trouvé en cherchant le noeud ayant l'UID p, calculé à partir de l'UID c du noeud courant, c'est à dire : $p = [((c-2)/k)+1]$. Chaque terme est stocké au niveau du noeud qui est le noeud le plus bas dans l'arbre contenant toutes les occurences du terme. On trouvera un exemple sur

la figure 2.9. De cette manière, chaque terme est stocké une et une seule fois dans l'arbre. Ce stockage des documents XML implique cependant une perte d'information, car certains termes ne sont pas stockés dans l'index à leur position exacte dans le document. Par exemple, la requête "fille dans E" est vraie alors que ce ne devrait pas être le cas.

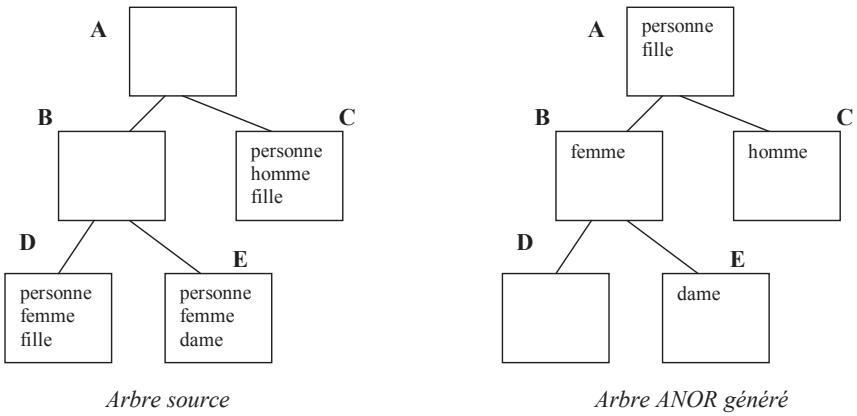
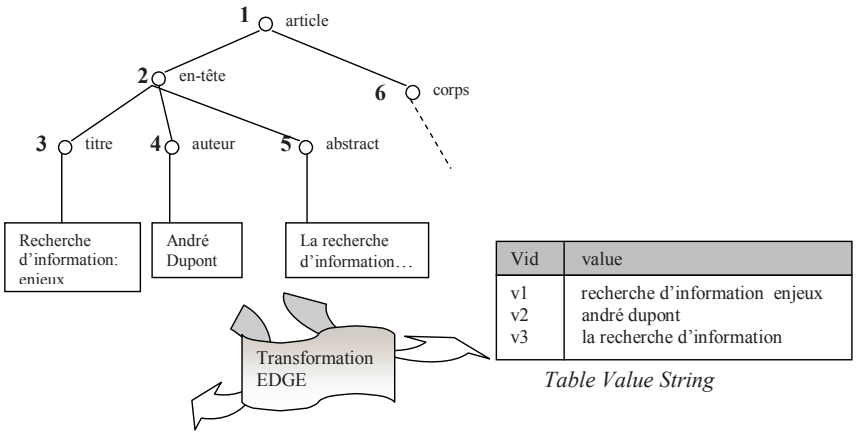


FIG. 2.9 – Exemple d'index ANOR

Le système XRS proposé dans [100, 190] utilise une architecture BUS (*Bottom Up Schema*) pour indexer et rechercher des documents XML. Une liste inverse de tous les termes apparaissant dans le contenu des éléments feuilles et les valeurs des attributs est créee. L'innovation réside non seulement dans la génération et le traitement automatique des XPath durant la recherche, mais aussi dans la possibilité de mettre à jour les indices de façon rapide.

Dans le système EDGE [68], une table appelée *EDGE* stocke la structure spécifique des documents XML, c'est à dire les arcs de la représentation en arbre des documents. Cette table, comme le montre la figure 2.10, stocke ainsi l'identifiant du noeud source et cible de chaque arc, l'ordre d'apparition des noeuds, le nom du noeud cible et le type du noeud cible (interne ou feuille). Si le noeud cible est un noeud feuille, une table séparée stocke la valeur du noeud.

L'inconvénient principal de l'approche EDGE est que de nombreuses requêtes ont des performances médiocres car elles nécessitent de nombreuses jointures sur la (très) grande table EDGE.
De manière similaire à EDGE, l'approche BINARY [68] matérialise la structure en arbre des documents XML dans des tables. L'approche BINARY crée une table séparée *Bname* pour chaque élément name. En d'autres termes, BINARY réalise une partition horizontale de la table EDGE en utilisant le nom de l'élément comme critère de partition. La structure des tables est identique



Vid	value
v1	recherche d'information enjeux
v2	andré dupont
v3	la recherche d'information

Table Value String

source	ordinal	name	datatype	target
NULL	1	article	ref	1
1	1	en-tête	ref	2
2	1	titre	ref	3
3	1	#text	string	v1
2	2	auteur	ref	4
4	1	#text	string	v2
2	3	abstract	ref	5
5	1	#text	string	v3
1	2	corps	ref	6

Table EDGE

FIG. 2.10 – Transformation d'un document XML avec l'approche EDGE

à celle employée pour la table EDGE, excepté pour l'attribut *name* qui est déplacé au niveau du schéma (voir figure 2.11).

Par comparaison à l'approche EDGE, les tables ont une taille moins importante, mais le nombre de jointures nécessaires pour des requêtes de chemin est toujours aussi grand. L'approche BINARY est cependant très efficace pour des recherches sur un élément particulier.

La structure d'index du Xpath Accelerator [93] a été conçue pour l'évaluation des expressions de chemin. L'intuition guidant le Xpath Accelerator est la suivante : en chargeant un nouveau document XML, le Xpath Accelerator exécute une traversée de la représentation en arbre du document. Durant ce parcours, des valeurs croissantes de pré-ordre ou post-ordre sont assignées aux noeuds visités, comme le montre la figure 2.12.

En stockant de plus la dimension de prédécesseur du noeud parent, un champ indiquant la présence d'attributs et le nom de balise de chaque noeud, une navigation efficace devient possible. Xpath Accelerator est particulièrement intéressant pour une navigation dans des documents XML et pour le traitement

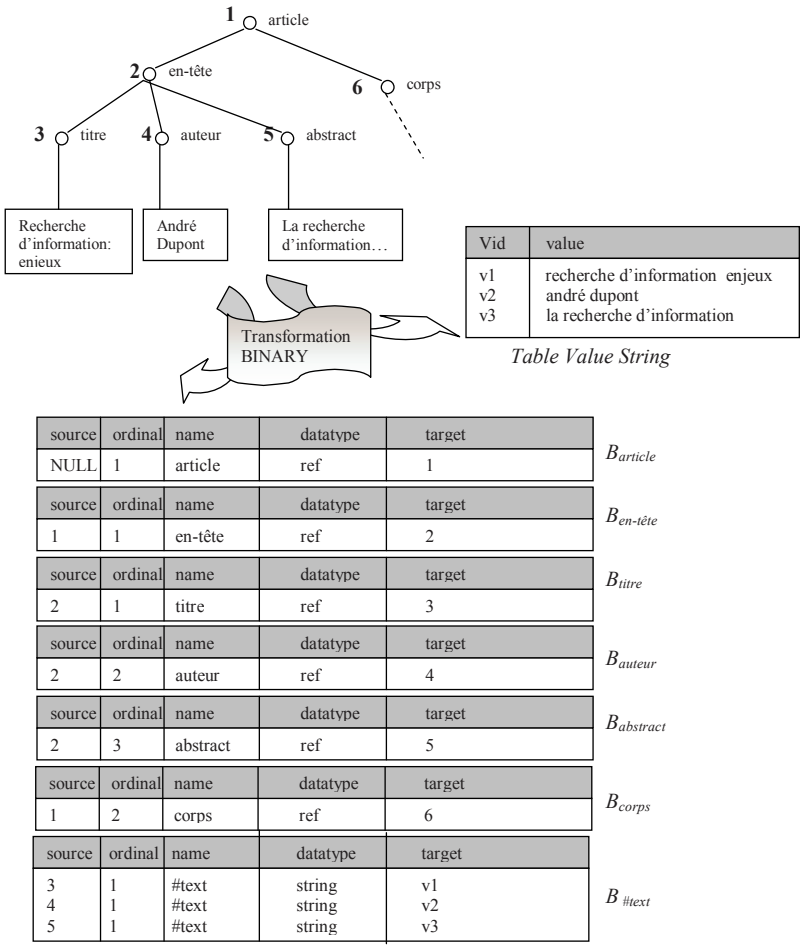


FIG. 2.11 – Transformation d'un document XML avec l'approche BINARY

d'expressions Xpath. Contrairement à d'autres approches basées sur des index de structure, Xpath Accelerator permet de répondre à des expressions Xpath qui n'ont pas pour origine la racine du document.
XISS (XML Indexing and Storage System) [126] a pour but de créer un index efficace pour la recherche de Xpath. Contrairement à l'approche Xpath Accelerator, XISS ne permet de traiter efficacement que des relations ancêtre-descendant. Cependant, les insertions sont facilitées.

2.4.4 Quelques exemples de systèmes commerciaux

La liste d'exemples que nous fournissons ici est loin d'être exhaustive, un nombre croissant d'entreprises proposant des solutions pour le stockage de documents XML.

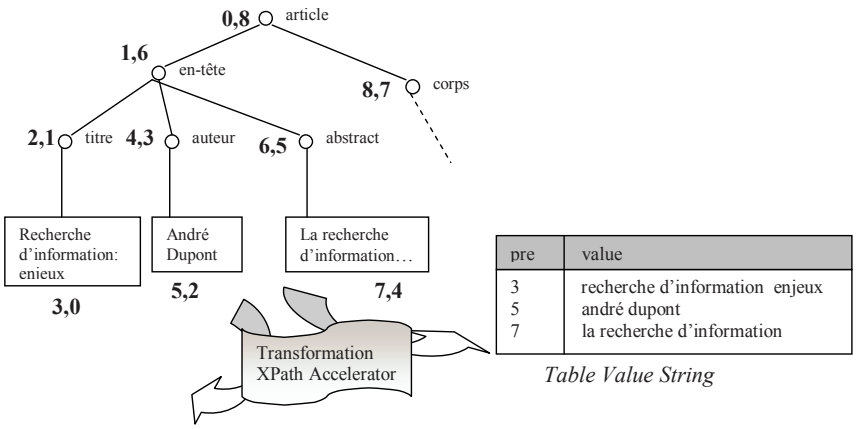


FIG. 2.12 – Transformation d'un document XML avec l'approche XPath Accelerator

Parmi les techniques de stockage XML natives, on peut citer *Xindice de Apache* [2][13], Tamino XML Server de Software A.G. [3] [196], TextML Server de IXIA Soft [4] [99], IPEDO XML Database de Ipedo [5] [98] ou GoXML de XML Global [6] [87].
Xylème Zone Server [7] [227] offre à ses utilisateurs un véritable entrepôt de données, indexées et stockées de telle sorte que la structure et la hiérarchie contenue dans un document soient prises en compte. En associant le langage de requêtes XQL et les capacités techniques du module XyView (décrit plus en détail dans la section 2.8), il est possible de rechercher des éléments précis dans de multiples documents même si ceux-ci utilisent divers schémas XML, et de récupérer un document de réponse qui aura une vue adaptée aux besoins.

Les approches issues des bases de données assurent l'ouverture à XML des données stockées dans des bases de données relationnelles existantes. On peut

[2]http ://xml.apache.org/xindice/
[3]http ://www.softwareag.com/tamino/
[4]http ://www.ixiasoft.com/
[5]http ://www.ipedo.com/html/ipedo_xml_database.html
[6]http ://www.xmlglobal.com
[7]http ://www.xyleme.com

par exemple citer e-XMLMedia XMLizer [8] [63] ou Xperanto d'IBM [9] [225], prototype issu du centre de recherche d'IBM qui sera intégré dans les futures versions de DB2.

Il existe aussi un certain nombre de systèmes souvent hybrides, étendant des SGBDR existants avec des techniques XML natives. Ces techniques sont bien sûr propriétaires et en évolution permanente.
Oracle ou IBM, par exemple, étendent leur système avec des types spécialisés pour XML. L'idée principale est que la base de donnée stocke les contenus XML avec des structures flexibles en valeur XML alors que les structures régulières sont transformées en bases de données en utilisant un des schémas de transformation que nous avons vu précedemment. Dans DB2 d'IBM[10], il est possible de stocker des documents XML dans des bases DB2 et de travailler avec ces documents structurés. Oracle 9.i[11] offre un type de données natif nommé XMLType. Ce type de données offre des fonctions d'extraction retournant des éléments DOM et des fonctions d'interrogation basées sur Xpath. Enfin, dans Sybase XML Processor [12], la gestion des documents XML est réalisée à travers d'un type de données XML appelé JXml. Il s'agit d'un type d'objet Java dédié à XML et surchargeable. Deux formes de stockage des documents XML sont disponibles : le mode *élément* permet le mapping des éléments du document XML dans des colonnes de table, et le mode *document* gère le stockage du document entier dans un champ texte : indexation et recherche plein-texte sont alors possibles. De plus, un mode hybride permet le stockage mixte de tout le document dans un champ texte et de certains éléments dans des colonnes.

On trouvera une description détaillée de tous ces systèmes dans [179].

2.4.5 Conclusion

Dans les systèmes de stockage *XML natifs*, le document garde son intégrité, et il est facile de le restituer : les performances en extraction sont donc généralement bonnes. Cependant, lors du stockage, des index sophistiqués doivent être gérés afin de permettre les recherches par le contenu (ce qui peut être presque aussi long que la déstructuration relationnelle). Ces index nécessitent des efforts d'implémentation coûteux et de nombreuses fonctions intégrées dans les systèmes de bases de données relationnelles doivent être reprogrammées...

[8] http ://www.e-xmlmedia.fr/site_francais/produits_xmlizer.htm
[9] http ://www.almaden.ibm.com/software/dm/Xperanto/index.shtml
[10] http ://www-3.ibm.com/software/data/db2/extenders/xmlext/index.html
[11] Oracle 9i - XML database : http ://otn.oracle.com/tech/xml/index.html
[12] http ://www.sybase.com/products/databaseservers/ase

L'utilisation des méthodes d'indexation proposées par la communauté des *bases de données* présente de nombreux avantages :

- utilisation de la puissance des moteurs relationnels ;
- accès rapide et direct aux éléments de données ;
- intégration simplifiée avec des systèmes d'information.

Cependant, il existe une perte de temps lors du stockage pour décomposer le document et stocker les différents composants dans des tables relationnelles, et des difficultés à recomposer le document complet par assemblage des différents éléments. De plus, pour caser le contenu XML dans des tables il faut effectuer une mise en correspondance des données par programmation et reprogrammer le processus chaque fois que le contenu ou l'application qui en assure la prise en charge sont modifiés. Enfin, ces approches sont orientées données alors que l'on souhaiterait plutôt des approches orientées documents pour pouvoir faire de la Recherche d'Information.

De nouvelles approches cherchent à combiner l'*approche orientée données* à l'*approche orientée documents*, pour profiter au mieux de toutes les caractéristiques des documents XML [89, 117, 204, 218]. Elles permettent notemment d'indexer le contenu textuel des documents et de pondérer les termes, ce qui rend ensuite possible un calcul de pertinence des éléments.

2.5 Langages de requêtes

L'interrogation des corpus de documents XML diffère de l'interrogation habituelle en RI, et ce du fait de l'information structurelle contenue dans les documents. D'un point de vue utilisateur, il existe deux façons principales d'interroger les collections de documents XML :

1. Il peut, s'il n'a pas d'idée précise de ce qu'il recherche, formuler des requêtes comparables à celles utilisées dans les moteurs de recherche traditionnels, c'est à dire des requêtes composées de simples mots-clés. On appelle ces requêtes *requêtes orientées contenu*.
2. Il peut ajouter des conditions sur l'information structurelle des documents, et préciser ainsi son besoin. Ceci présuppose cependant qu'il a une connaissance au moins partielle de la collection qu'il interroge. On parle alors de *requêtes orientées contenu et structure*.

De nombreux langages de requêtes ont été proposés dans la littérature, et nous nous proposons d'en détailler quelques uns dans cette section. Ces langages, issus de la communauté des bases de données, se concentrent principalement sur l'introduction de la dimension structurelle dans les requêtes, et traitent souvent le contenu des éléments de façon booléenne (présent / absent).
D'après [82], un langage de requêtes XML doit intégrer les fonctionnalités des

langages de requêtes pour les systèmes documentaires et pour les bases de données. Les différents langages de requêtes doivent donc supporter les fonctions suivantes :

- Sélection des arbres sur critères multiples,
- Possibilité d'effectuer toutes les opérations des types de base,
- Quantification universelle et existentielle des variables,
- Combinaison des données depuis des documents,
- Tri des résultats,
- Imbrication de requêtes,
- Possibilité d'utilisation des agrégats et fonctions associées.

De plus, l'intégration de fonctions des systèmes documentaires nécessite la prise en compte de requêtes par liste de mots-clés du type :

CONTAINS (<élément>, collection de mots clés)

Au delà des requêtes exactes, il serait aussi souhaitable de supporter des requêtes approchées sur mots-clés du type :

SIMILAR (<élément>, collection de mots clés).

Enfin, de nouvelles fonctionnalités traitant les structures sont nécessaires :

- respect de la hiérarchie et des séquences,
- agrégation de données depuis des documents,
- préservation de structures,
- construction de structures nouvelles.

Dans ce qui suit, nous donnons une description des différents langages de requêtes adaptés à XML, suivant leur ordre chronologique d'apparition (figure 2.13).

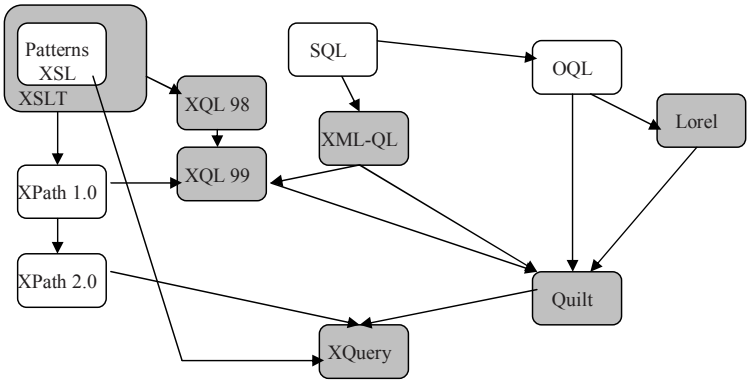


FIG. 2.13 – Historique des langages d'interrogation XML

2.5.1 Les précurseurs

Le langage UnQL [33] possède une architecture d'arbre "étiqueté". Les données peuvent être représentées sous forme d'arbres ou de structures cycliques. Le langage permet la sélection directe à travers la correspondance de sous-arbres via des patrons et est au moins aussi expressif que l'algèbre relationnelle.
On trouvera des exemples d'opérations de sélection et de jointure dans le tableau 2.4.

Select * from R1	Select t where R1 => \ t <- DB
Select A,D from R1,R2 Where R1.C=R2.C	Select Tup =>{ (A =>x, D=>z)} Where R1 => Tup => { A =>\ x, C =>\ y } <- DB, R2 => Tup =>{ C =>, D => \z} <- DB

TAB. 2.4 – Comparaison d'opérations de sélection et de jointure en SQL et UnQL

=> représente un chemin, t,x,y,z sont des sous-arbres et le symbole "\" implique une recherche de motifs (*pattern matching*).
Le modèle de données Lore a été développé pour traiter des données semi-structurées, et a récemment été étendu pour traiter des données XML. Les données d'un ou plusieurs documents XML peuvent être transcrites en Lore sous forme de graphe, avec les arêtes représentant les sous-éléments (descendants) et les liens (Xlinks et Xpointers). Le langage de requêtes LOREL a été conçu pour accéder à des données LORE dans un contexte XML [3].
La construction de base de LOREL est l'expression de chemin simple. Une expression de chemin simple est une séquence de balises séparées par des points (.) à la place des barres obliques de Xpath (par exemple Guide.Restaurant.Adresse). On notera cependant que Lorel a été conçu avant Xpath et ne s'appuie donc pas dessus dans ses spécifications. Au delà de l'expression de chemins simples, LOREL permet l'utilisation d'expressions de chemin généralisées (certaines balises sont remplacées par des symboles ou des expressions représentant notamment un joker, un nombre illimité de jokers ou des combinaisons de balises). Lorel présente de nombreux avantages : il supporte les jointures, les manipulations d'ensembles d'identifieurs d'objets et les expressions de chemin. De plus, les mises à jour sont permises. Le tableau 2.5 donne un exemple de requête.

Pour traiter l'information structurée provenant du Web en utilisant les technologies issues des bases de données, le système Strudel a été développé, associé au langage de requêtes StruQL [65]. Ce langage a quelques similarités avec Lo-

```
SELECT $R.Nom, $R.Téléphone
FROM Répertoire.Hotel $H, Guide.Restaurants $R
WHERE $H.Adresse.Ville.Rue=$R.Adresse.Ville.Rue
AND $H.Nom= " Le Lutécia "
```

TAB. 2.5 – Exemple de requête Lorel : Lister le nom des restaurants dans la rue de l'hôtel Lutécia.

rel et UnQL, notamment dans sa représentation de l'espace d'information, c'est à dire à l'aide de graphes.

2.5.2 XML-QL

XML-QL [125] est une nouvelle approche dans laquelle des requêtes basées sur SQL peuvent être formulées pour interroger des documents XML. XML-QL peut construire des documents XML et supporter des vues ordonnées ou non de documents XML. Il supporte les expressions de chemin et les expressions régulières. Les constructions essentielles de XML-QL sont les suivantes :

- les sélections : utilisation de canevas (*patterns*), c'est à dire de documents XML dans lesquels certaines données sont remplacées par des variables pour retrouver des parties de documents.
- Les éléments optionnels : les canevas retrouvent les documents ayant toutes les balises mentionnées. Dans un objet XML, certains éléments peuvent être manquants et il faut cependant être en mesure de retrouver les objets possédant optionnellement certains éléments.
- La construction des résultats : XML-QL permet de construire de nouveaux arbres en résultats de requêtes.
- Les jointures : celles-ci s'effectuent simplement par des réutilisations de variables dans des conditions.
- Les variables balises : XML-QL permet l'interrogation des métadonnées. Pour cela, il est possible d'utiliser des variables pour désigner des balises.
- Les expressions régulières : il est possible d'exprimer des expressions de chemins par imbrication des attributs.

Cependant, les opérateurs d'agrégation et les mises à jour ne sont pas supportés. On trouvera un exemple de requête XML-QL dans le tableau 2.6.

XML-QL a été parmi les premiers langages de requêtes pour XML proposé au Consortium W3C. Plus récemment, une équipe de recherche de l'INRIA[13] , a proposé une nouvelle extension de ce langage, permettant de faire des recherches " floues ", sans connaissance exacte de la structure des données. Ces recherches

[13]Institut National de Recherche en Informatique et en Automatique

```
WHERE <Hotel @Catégorie = " *** ">
        <$a>$V</$a> IN Nom, téléphone, Fax
        <Adresse>
                <Ville> Paris </Ville>
        </Adresse>
</Hotel> IN Répertoire
SELECT
<Résultat>
<$a>$V</$a>
</Résultat>
```

TAB. 2.6 – Exemple de requête XML-QL : Recherche de tous les hôtels de catégorie trois étoiles à Paris, avec leur nom, leur téléphone et leur fax.

combinent des techniques d'interrogation structurée avec des recherches d'information à base de mots-clés, reposant sur une approche de médiation [67].

2.5.3 XQL

XQL [168] a été proposé par Microsoft pour interroger des collections de documents XML. Au lieu d'utiliser des canevas XML, XQL propose d'étendre les URL pour interroger des collections de documents XML avec des expressions Xpath.
XQL permet de gérer des éléments courants, d'insérer des prédicats de comparaison, des méthodes, des expressions booléennes, de gérer des vecteurs, de formuler des requêtes à partir des éléments courants, etc. Il présente le principal avantage d'étendre directement les notations des URL mais reste cependant loin des possibilités de XML-QL, notamment en matière de restructuration des résultats. Il possède de plus une syntaxe un peu complexe. On trouvera un exemple de requête XQL dans le tableau 2.7.

```
//Restaurant ? (@catégorie[text()="***"])/Ville [text()= "Paris "]
```

TAB. 2.7 – Exemple de requête XQL : Recherche de tous les restaurants 3 étoiles dont un élément descendant Ville contient pour valeur Paris

2.5.4 QUILT

Le langage QUILT [40] a été développé dans le but d'être un langage flexible, combinant des caractéristiques pour interroger des documents et des bases de données.
Des caractéristiques navigationnelles sont disponibles pour parcourir les documents, on peut créer des variables, et des fonctions peuvent être définies pour agir sur les objets de données. En ce qui concerne l'interrogation même, elle est basée sur la construction FLWR (*for-let-where-return*), et elle permet de faire des jointures et d'utiliser des opérateurs d'agrégation. QUILT est ainsi bien plus puissant que SQL en permettant de restructurer l'information résultat d'une requête en document XML. QUILT a été soumis au W3C en 2001 par IBM et certains auteurs de XML-QL.

2.5.5 XQuery

Xquery [66] est le langage de requêtes pour XML proposé par le W3C. Xquery tire très fortement ses constructions et caractéristiques de Quilt, lui-même dérivé de Xpath, XQL, XML-QL, Lorel et Yatl .
Xquery peut être perçu comme un surensemble de SQL. Les fonctionnalités de SQL sur les tables (collections de tuples) sont étendues pour supporter des opérations similaires sur les forêts (collections d'arbres) Ces extensions ont conduit à intégrer les fonctions suivantes : projection d'arbres sur des sous-arbres, sélection d'arbres et de sous-arbres en utilisant des prédicats sur les valeurs des feuilles, utilisation de variables dans les requêtes pour mémoriser un arbre ou pour itérer sur des collections d'arbres, combinaison des arbres extraits de collection en utilisant des jointures d'arbres, réordonnnancement des arbres, imbrication des requêtes, calculs d'agrégats, et utilisation possible de fonctions utilisateur.
XQuery supporte des fonctions orientées RI : en particulier, un prédicat *contains* est intégré pour la recherche par mots-clés.Pour faciliter la recherche dans des structures mal connues, Xquery permet enfin d'exprimer des chemins indéterminés ou partiellement connus, tout comme Xpath. On trouvera dans le tableau 2.8 un exemple de requête XQuery.

2.5.6 Autres langages de requêtes

XML-GL [37] est un langage de requêtes graphique s'utilisant sur des graphes XML. Il présente le principal avantage d'être très ergonomique. Il permet de plus de retrouver des chemins pas complètement définis, supporte les opérations

```
For $R in collection (" Guide ")/Restaurant,
        $H in collection (" Répertoire)/Hôtel
where $H/Rue=$R and $H/Nom= "Le Lutécia "
return
        <RestauTel>
                <Nom> $R/Nom/text( ) </Nom>
                <Tel> $R/Téléphone/text( ) </Tel>
        </RestauTel>
```

TAB. 2.8 – Exemple de requête XQuery : lister le nom des restaurants avec leur numéro de téléphone dans la rue de l'hôtel Lutécia.

de jointure et les opérateurs d'agrégation, et permet la mise à jour de documents XML. Cependant, contrairement à XML-QL, il n'est possible de faire des requêtes que sur le contenu des tags et non sur leurs noms mêmes. C'est pour cette raison fondamentale qu'il est considéré comme moins puissant que XML-QL. La figure 2.14 montre un exemple de requête XML-GL, dans lequel on génère des paires avec les éléments $< fabricant >$ et $< véhicule >$ où $< nom\_fab >=< nom >$, $< nom\_modele >=< modele >$ et $<année>=<année>$.

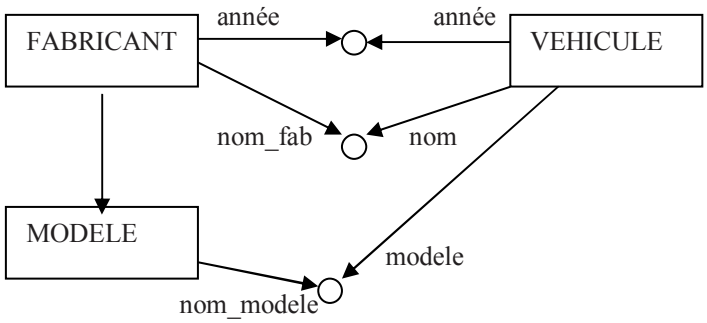


FIG. 2.14 – Exemple de requête XML-GL :Jointure

Les langages que nous avons présentés jusqu'ici proposent un prédicat de type "contains" pour exprimer des conditions sur le contenu des éléments. Cependant, ce type de prédicat ne permet pas de mesurer la similarité entre les unités d'informations et les conditions de contenu. Les approches que nous décrivons ci-dessous sont issues de la communauté de la RI et cherchent à intégrer ce dernier aspect.

Les langages de requêtes que nous avons décrits précédemment ne peuvent pas traiter des requêtes du genre " trouver les livres et les CDs avec des titres similaires ". Le langage ELIXIR (an Expressive and Efficient Language for XML

Information Retrieval) [42, 43] étend le langage XML-QL avec un opérateur de similarité textuelle. Pour répondre à une requête de similarité, l'algorithme proposé réécrit la requête ELIXIR originale en une série de requêtes XML-QL. Ces dernières génèrent des données relationnelles intermédiaires puis utilisent des techniques de bases de données relationnelles pour évaluer la requête de similarité sur ces données intermédiaires. On obtient un document XML contenant des noeuds ordonnés par similarité. D'après les auteurs, l'algorithme est adapté à la taille et des documents XML et à la complexité des requêtes.
On trouvera dans le tableau 2.9 un exemple de requête et la réponse envoyée par ELIXIR.

<book>Traditional Ukrainian cookery</> <book>Being and nothingness</> <book>Shooting Elvis</> <cd>Traditional Ukrainian folk music</> <cd>Being there</> <cd>Milk Cow blues</></>	db.xml
CONSTRUCT <item>$b</> WHERE <item.book>$b</> in " db.xml ", <items.cd>$c</> in "db.xml", $b $c.	Q
<item>Traditional Ukrainian cookery</> <item> Being and nothingness</>	réponse

Tab. 2.9 – Document XML décrivantdes livres et CDs, requête ELIXIR pour trouver des éléments ayant des titres similaires, et réponse renvoyée par le système

En partant d'un constat sur les lacunes du langage XQL (pas de pondération des résultats, pas de prédicats vagues pour mesurer la similarité et pas de correspondance sémantique entre les différents tags XML), Grossjohann [91] propose le langage XIRQL. XIRQL est une extension du langage XQL et est basé sur pDatalog, variante de Datalog dans laquelle les faits et les règles possèdent des probabilités. XIRQL possède des opérateurs permettant l'utilisation de prédicats vagues et l'abstraction des types de données et des balises, et permet une recherche orientée sur la pertinence (c'est-à-dire avec une pondération des résultats). Le langage possède cependant une syntaxe complexe, difficilement utilisable sans une interface appropriée. XIRQL a été implémenté dans le système HyreX et testé dans la campagne d'évaluation INEX 2002 [84].
Cohen et al. [48] proposent le langage EquiX, permettant de combiner la recherche de motifs, la quantification et les expressions logiques pour interroger

à la fois les données et les métadonnées des documents XML. Les requêtes peuvent être formulées avec une syntaxe abstraite basée sur des graphes ou une syntaxe formelle concrète. L'algorithme d'évaluation a un coût polynomial et la DTD décrivant les documents résultats est dérivée automatiquement de la requête.
Dans leur système XISS, Li et Moon [126] proposent un modèle pour répondre à des requêtes demandant de retrouver des éléments ou attributs communs dans des documents XML ne suivant pas la même DTD.
Dans [49], le langage Tequyla-TX est présenté. Tequyla-TX est un langage de requête typé permettant d'effectuer aussi bien des requêtes orientées bases de données que des requêtes basées sur des mots-clés. Afin de répondre aux besoins fondamentaux des applications de recherche de texte, il autorise la recherche basée sur des mots ou des caractères, ce qui est particulièrement utile pour des applications littéraires (comme l'analyse de l'utilisation des prépositions dans les textes latins).
Le langage XML Fragment [35] se propose d'interroger les documents XML sous forme XML, ce qui permet à l'utilisateur d'exprimer des besoins imprécis.
Le langage NEXI a été défini dans [206, 207] pour répondre aux besoins de la campagne d'évaluation INEX. Les requêtes étaient en effet précédemment exprimées en XML (pour 2002) ou XPath (pour 2003), mais dans le premier cas, le langage n'était pas assez puissant, et il était trop complexe dans le second cas (63% des requêtes exprimées par les participants (experts en RI) contenaient des erreurs de syntaxe !). NEXI a alors été conçu comme un sous-ensemble extensible d'XPath interprétable de manière vague (il s'agit d'un langage de requête orienté RI et non Base de données). NEXI est amené à évoluer au fil des années pour s'adapter aux différentes tâches proposées aux participants d'INEX (notamment la tâche hétérogène ou la tâche en langage naturel).
On notera enfin que le W3C a récemment proposé un Working Draft [216], qui a pour but d'étendre les caractéristiques de recherche de XQuery à la recherche plein-texte. Le langage TexQuery [8] en est une application.

2.5.7 Conclusion sur les langages de requêtes XML

Comme on peut le voir, de très nombreux langages de requêtes ont été proposés dans la littérature. La plupart sont très puissants, mais leur syntaxe, souvent dérivée de SQL, est difficilement accessible pour les novices. Des interfaces adaptées peuvent être associées, mais leur feraient perdre de leur puissance.
Le tableau 2.10 compare les principales caractéristiques des langages de requêtes les plus importants que nous venons de passer en revue.

La plupart des langages que nous venons de présenter sont basés sur une approche orientée base de données. Les conditions sur le contenu textuel des

	LOREL	**XML-QL**	**XQL**	**XML-GL**	**Xquery**
Navigation	Oui	Oui	Oui	Oui	Oui
Support Xpath	Limité	Limité	Oui	/	Oui
Sélection	Oui	Oui	Oui	Oui	Oui
Jointure	Oui	Oui	Non	Oui	Oui
Tri	Oui	Oui	Non	Oui	Oui
Construction	Non	Oui	Non	Oui	Oui
Mot-clé	Non	Non	Non	Non	Oui
Fonction	Oui	Oui	Non	/	Oui
Imbrication	Oui	Oui	Oui	Oui	Oui
Agrégation	Oui	proposée	Non		Oui

TAB. 2.10 – Tableau comparatif de différents langages de requêtes pour XML

documents sont énoncées à l'aide de prédicats de type "contains", qui ne permettent pas l'évaluation de la similarité entre la requête et les unités d'informations.
Des langages orientés RI font peu à peu leur apparition, et les propositions rencontrées dans littérature proposent d'ajouter des fonctionnalités concernant la recherche sur le contenu (utilisation d'un prédicat "*about*" pour remplacer le prédicat "*contains*", ou bien encore d'opérateurs booléens dans les conditions de contenu).
Quelle que soit l'approche utilisée (orientée *données* ou orientée *documents*), une connaissance parfaite de la structure des documents est aussi souvent nécessaire à l'utilisateur pour pouvoir formuler des requêtes. Il doit de plus spécifier l'élément qu'il désire voir retourné par le SRI, alors qu'il n'a pas forcément d'idée précise de ce qu'il recherche exactement. L'utilisation de ces syntaxes certes complètes mais aussi complexes nous amène à nous interroger sur leur utilité : dans [199], les auteurs montrent en effet que l'utilisation du langage naturel pour exprimer les requêtes peut donner des résultats comparables à ceux obtenus lorsque les requêtes sont exprimées en suivant la grammaire du langage NEXI.
Enfin, nombreuses sont les spécifications de langages, mais rares sont les implémentations concrètes...

2.6 Traitement des requêtes

Les modèles de recherche que nous présentons dans cette partie sont spécifiques aux approches orientéees RI, puisqu'ils cherchent à attribuer des scores de pertinences aux noeuds des documents XML. Les approches orientées BD ne se posent pas ce problème, le contenu des documents étant traité de façon

booléenne (présent/absent).
Dans les approches présentées dans la littérature, les modèles de RI classiques ont été adaptés pour tenir compte de l'information structurelle contenue dans les documents XML et des tailles variées des éléments (c'est à dire des granularités variées de l'information). Ces modèles cherchent à répondre à des requêtes orientées contenu et /ou à des requêtes orientées contenu et structure. Dans le premier cas, les systèmes doivent décider de la granularité idéale de l'information à renvoyer à l'utilisateur, alors que dans le second, les conditions de structure des requêtes donnent des indications sur le type d'élément à renvoyer.
Dans cette section, nous nous proposons de détailler les différentes méthodes proposées pour adapter le modèle vectoriel ou bien encore le modèle probabiliste. Nous nous attardons ensuite sur les méthodes proposant une approche spécifique pour le traitement des conditions de structure.

2.6.1 Modèle vectoriel étendu

Dans les approches issues du modèle vectoriel, une mesure de similarité de *chaque* élément à la requête est calculée, et ce à l'aide de mesures de distance dans un espace vectoriel. Les éléments sont représentés par des vecteurs de termes pondérés. Pour ce faire, la plupart des approches indexent des sous-arbres imbriqués (section 2.4.2.1), c'est à dire propagent les termes des noeuds feuilles dans l'arbre du document. Les éléments sont renvoyés à l'utilisateur par ordre décroissant de pertinence.

On trouve dans [79] une des premières adaptations du modèle vectoriel. La similarité d'un noeud n à une requête $q = \{t_1, t_2, ...t_T\}$ est exprimée selon l'équation 2.1 :

$$sim(q,n) = \alpha(T) cosm(q,n) + \sum_{k=1}^{s} \frac{cosm(q,n_k)}{\beta^{k-1}} \tag{2.1}$$

où $\alpha(T)$ est un facteur permettant de prendre en compte le type du noeud, s est le nombre de noeuds enfants n_k de n, et β est un paramètre permettant d'assurer que le nombre d'enfants n'introduit pas un biais dans la formule.
La fonction *cosm* est définie de la façon suivante :

$$cosm(q,n) = \sum_{i=1}^{T} \frac{w_i^q * w_i^n}{|n|} \tag{2.2}$$

avec w_i^q et w_i^n respectivement le poids du terme $t_i$dans la requête q et dans le noeud n, et $|n|$ le nombre de termes dans le noeud n.
La pertinence d'un noeud peut ainsi être calculée à part, puis combinée avec

la pertinence des noeuds descendants.
Le modèle peut être généralisé en permettant le traitement des requêtes orientées contenu et structure. L'idée de base est là encore d'appliquer le modèle récursivement à chaque sous-arbre de la hiérarchie pour ensuite effectuer un agrégat des scores.

Schlieder et Meuss [186] intègrent la structure des documents dans la mesure de similarité du modèle vectoriel. Leur modèle de requête est basé sur l'inclusion d'arbres : cela permet de formuler des requêtes sans connaître la structure exacte des données.
Les auteurs proposent la notion de *terme structurel*, définie comme un arbre étiqueté. *book[author]*, *book[Bradley, title[XML]]*, *author[Bradley]*, ... sont des exemples de termes structurels.
Les notions de *tf* et *idf* sont adaptées au processus de recherche dans des documents structurés. Soit E un élément de type t. Le poids $w^t_{T,E}$ d'un terme structurel T dans E est défini par :

$$w^t_{T,E} = tf_{T,E}.idf^t_T = \frac{freq_T(E)}{maxfreq(E)}.(log(\frac{|E^t|}{n_T}) + 1) \tag{2.3}$$

avec $freq_T(E)$ le nombre d'occurrences de T dans E, $maxfreq(E)$ le nombre maximal d'éléments de la collection possédant la même étiquette que E, $|E^t|$ le nombre d'éléments de type t et n_T le nombre d'éléments contenant T.
Les auteurs combinent ainsi le modèle vectoriel et le "tree matching" afin de répondre à des requêtes orientées contenu et structure. Dans le modèle proposé, seuls les éléments (c'est à dire les sous-arbres) qui ont une structure qui peut être réduite à celle de la requête (c'est à dire qu'en supprimant certains éléments du sous-arbre, on peut arriver à la requête) ont un score de pertinence non nul.

Dans [90], Grabs et Scheck proposent d'évaluer l'importance d'un terme dans un élément donné en fonction de l'importance du terme dans les éléments du même type.
Lorsque la requête est composée d'une condition sur le type d'un élément (on nommera *cat* ce type) ainsi que d'une condition sur le contenu de cet élément (requête orientée contenu et structure), la similarité d'un élément e de type *cat* à la requête q est calculée selon l'équation 2.4 :

$$RSV(e,q) = \sum_{t \in terms(q)} tf(t,e).ief_{cat}(t)^2.tf(t,q) \tag{2.4}$$

où $tf(t,e)$ est la fréquence du terme t dans l'élément e et $ief_{cat} = log\frac{N_{cat}}{ef_{cat}(t)}$, avec N_{cat} le nombre d'éléments du type *cat* et $ef_{cat}(t)$ la fréquence du terme t dans les éléments du type *cat*.
Les requêtes orientées contenu sont quant à elles traitées de la façon suivante.

Soit $SE(e)$ l'ensemble des descendants de e incluant e. $\forall\ se \in SE(e)$, $l \in path(e, se)$ est une étiquette appartenant au chemin reliant e à se, c'est à dire un type d'élément. Soit enfin $aw_l \in [0, 1]$ un facteur modélisant l'importance de l'étiquette l. La similarité d'un élément e à une requête q composée de simples mots-clés est définie de la façon suivante :

$$RSV(e, q) = \sum_{se \in SE(e)} \sum_{t \in terms(q)} tf(t, se)(\prod_{l \in path(e,se)} aw_l).ief_{cat(se)}(t)^2.tf(t, q) \tag{2.5}$$

Cette approche a été évaluée dans la campagne d'évaluation INEX 2002 et les résultats ont cependant été peu convainquants.

Le modèle JuruXML [137, 135] propose d'indexer les éléments selon leur type (un index par type d'élément) et d'appliquer ensuite le modèle vectoriel pour la pondération des éléments.
Les requêtes orientées contenu sont évaluées sur chacun des index et les résultats, qui ont été normalisés, sont ensuite fusionnés afin de fournir à l'utilisateur une liste unique de résultats.
Une requête structurée est quant à elle évaluée en trois phases. Tout d'abord, la requête originale est décomposée en ensemble de conditions de la forme $(chemin, terme)$. Ensuite, une correspondance vague entre les chemins est calculée. Soit c_i^q la condition de chemin pour le terme t_i et c_i^e le XPath du terme t_i dans l'élément e. La fonction de similarité entre les deux chemins est exprimée selon l'équation 2.6 :

$$cr(c_i^q, c_i^e) = \begin{cases} \frac{1+|c_i^q|}{1+|c_i^e|} si\ c_i^q\ est\ une\ sous-sequence\ de\ c_i^e \\ 0\ sinon \end{cases} \tag{2.6}$$

Par exemple, $cr(article/bibl, article/bm/bib/bibl/bb) = 3/6 = 0.5$.
On a enfin :

$$RSV(e, q) = \frac{\sum_{(t,c_i^q) \in q} \sum_{(t,c_i^e) \in e} w_q(t) * w_e(t) * cr(c_i^q, c_i^e)}{|q| * |e|} \tag{2.7}$$

où $w_q(t)$ et $w_e(t)$ sont les poids du terme t dans q et e, et $|q|$ et $|e|$ sont les nombres de termes dans q et e.
Cette dernière approche, évaluée dans le cadre de la campagne INEX 2004 permet d'obtenir de bons résultats par rapport à l'ensemble des participants.

Le moteur de recherche XXL [201] est lui aussi basé sur le modèle vectoriel et utilise une fonction de tri basée sur *tf* et *idf*. XXL offre des fonctionnalités pour la recherche orientée-pertinence de chemins, c'est à dire que la recherche est effectuée avec des conditions de chemins vagues. XXL repose sur une syntaxe SQL (*select-from-where*).
On trouvera d'autres exemples d'adaptation du modèle vectoriel dans [12, 53, 134, 137, 201, 35, 219, 103].

2.6.2 Modèle probabiliste

2.6.2.1 Le modèle FERMI

Le modèle de données multimedia FERMI [41] est l'un des premiers modèles à considérer la vue logique des documents et à permettre le renvoi à l'utilisateur non seulement de documents entiers, mais aussi de sous-structures de la structure logique des documents (c'est à dire des *noeuds*). Pour ce faire, le modèle propose une représentation des données se restreignant aux éléments importants pour la recherche, en négligeant par exemple les informations de présentation .
Dans le modèle, chaque document est un arbre composé d'objets structurels typés (par exemple un livre, un chapitre, une section, un paragraphe, une image,...), et dans lequel les feuilles contiennent des données mono-média. Les documents hypermédias contiennent aussi des liens entre les différents noeuds et éventuellement entre les différents documents. Des attributs sont assignés aux noeuds, et il peut s'agir d'attributs standards (comme l'auteur ou la date de création) ou bien encore la description du contenu du noeud pour l'index. Ce dernier type d'attributs est initialement assigné aux seuls noeuds feuilles, et leur contenu dépend du type de média. Par exemple, pour le texte, ils servent à décrire le contenu sémantique, alors que pour les images, un contenu spatial et perceptif est ajouté.
Les valeurs des attributs sont propagées vers le haut ou vers le bas de la hiérarchie, selon la classe de l'attribut auquel elles sont rattachées. Par exemple, les auteurs des différents noeuds sont propagés vers le haut (en utilisant des techniques de fusion), alors que la date de publication d'un document complet est propagée vers le bas. Le modèle supporte en outre le typage des noeuds, des liens et des attributs.
La recherche suit une approche logique, c'est à dire qu'elle consiste à chercher des noeuds n qui impliquent la requête q. La formulation originale du modèle est basée sur la logique de prédicats, et la stratégie de recherche est composée de deux phases, "fetch and browse" : (1) dans la première phase, on cherche les noeuds exhaustifs, et (2) dans la seconde phase, on navigue à partir de ces noeuds dans l'arborescence du document pour retrouver (éventuellement) des noeuds plus spécifiques.
Dans [120], le modèle est affiné (mais restreint à la logique des propositions) en utilisant la théorie de Dempster-Shafer [188]. La théorie de l'évidence de Dempster-Shafer est utilisée principalement parce qu'elle possède une règle de combinaison permettant d'effectuer une agrégation du score de pertinence des éléments en respectant la théorie de l'incertain. Ceci présente le principal inconvénient de laisser un fossé énorme entre le modèle de données sémantique et la logique utilisée pour son implémentation.

2.6.2.2 Le modèle d'inférence probabiliste

Pour étendre le modèle probabiliste inférentiel aux documents XML, les probabilités doivent tenir compte de l'information structurelle. Une approche est d'utiliser des probabilités conditionnelles de jointure, avec par exemple $P(d|t)$ devenant $P(d|p \ contains \ t)$, où d représente un document ou une partie de document, t est un terme et p est un chemin dans l'arbre structurel de d.

Une méthode d'augmentation basée sur le modèle probabiliste est proposée par Fuhr et al. dans [75, 84]. Cette méthode est basée sur le langage de requêtes XIRQL, et a été implémentée au sein du moteur de recherche HyRex.
Dans cette approche, les noeuds sont considérés comme des unités disjointes (section 2.4.2.1). Tous les noeuds feuilles ne sont cependant pas indexés (car d'un granularité trop fine). Dans ce cas-là les termes sont propagés jusqu'au noeud indexable le plus proche. Afin de préserver des unités disjointes, on ne peut associer à un noeud que des termes non reliés à ses noeuds descendants.
Le poids de pertinence des noeuds dans le cas de requêtes orientées contenu est calculé grâce à la *propagation* des poids des termes les plus spécifiques dans l'arbre du document. Les poids sont cependant diminués par multiplication par un facteur, nommé facteur "*d'augmentation*".
Par exemple, considérons la structure de document de la figure 2.15, contenant un certain nombre de termes pondérés (par leur probabilité d'apparition dans l'élément), et la requête "XML".

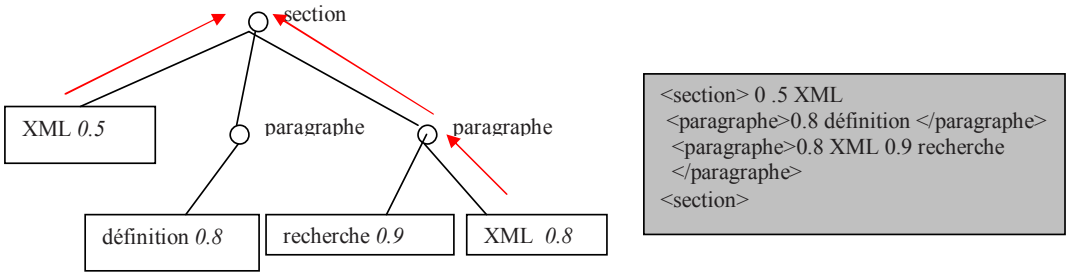


FIG. 2.15 – Modèle d'augmentation [75]

Le poids de pertinence de l'élément *section* est calculé comme suit, en utilisant un facteur d'augmentation égal à 0.7 :
P([section, XML])+ P([paragraphe[2]]). P([paragraphe[2], XML)
- P([section, XML]).P([paragraphe[2]]).P([paragraphe[2], XML)
= 0.5 + 0.7*0.8 - 0.5*0.7*0.8=0.68 .
Le noeud *paragraphe* (ayant une pertinence de 0.8 à la requête) sera donc mieux classé que le noeud *section*.
Pour les requêtes orientées contenu et structure, des probabilités d'apparition

de chaque terme de la condition de contenu dans les éléments répondant aux conditions de structure sont calculées, et des sommes pondérées de ces probabilités sont ensuite effectués.

2.6.2.3 Autres approches

Dans [104], les auteurs proposent une approche basée sur les ***modèles de langage*** pour traiter les requêtes orientée contenu. Les auteurs considèrent que comme n'importe quel élément XML peut potentiellement être renvoyé à l'utilisateur, chaque élément doit être traité comme une unité d'indexation à part entière. Par conséquent, pour chaque élément, le texte qu'il contient ainsi que le texte contenu dans ses descendants est indexé (voir approches d'indexation basées sur les sous-arbres imbriqués, section 2.4.2.1). Un modèle de langage est ensuite estimé pour chaque élément de la collection. Pour une requête donnée, les éléments sont triés par rapport à la probabilité que le modèle de langage de l'élément génère la requête. Ceci revient à estimer la probabilité P(e,q), ou e est un élément et q une requête :

$$P(e,q) = P(e).P(q|e) \tag{2.8}$$

Deux probabilités doivent donc être estimées : la probabilité a priori de l'élément $P(e)$ et la probabilité qu'il génère la requête $P(q|e)$. Pour la seconde probabilité, les auteurs considèrent que les termes de la requête sont indépendants, et utilisent une interpolation linéaire du modèle d'élément et du modèle de collection pour estimer la probabilité d'un terme de la requête. la probabilité d'une requête t_1,t_2,...t_n est ainsi calculée de la façon suivante :

$$P(t_1,...t_n|e) = \prod_{i=1}^{n}(\lambda.P(t_i|e) + (1-\lambda).P(t_i)) \tag{2.9}$$

où $P(t_i)$ est la probabilité d'observer le terme t_i dans l'élément e, $P(t_i)$ est la probabilité d'observer le terme dans la collection et λ est un paramètre de lissage.
Le calcul des probabilités peut être réduit à la formule de calcul des scores ci-dessous, pour un élément e et une requête $t_1, ...t_n$.

$$\begin{aligned} s(e,t_1,t_2,...t_n) &= \beta.log(\sum_t tf(t,e)) \\ &\quad + \sum_{i=1}^{n} log(1 + \frac{\lambda.tf(t_i,e).(\sum_t df(t))}{(1-\lambda)df(t_i).(\sum tf(t,e))}) \end{aligned} \tag{2.10}$$

où $tf(t,e)$ est la fréquence du terme t dans l'élément e, $df(t)$ est le nombre d'éléments contenant t, λ est le poids donné au modèle de langage de l'élément en lissant avec le modèle de la collection, et β est un paramètre servant à combler le fossé entre la taille de l'élément moyen et la taille de l'élément moyen

pertinent.
Dans [223], l'utilisation de la fréquence inverse d'élément *ief* est proposée pour faciliter les pondérations par élément : un nouveau poids probabiliste pour les termes est alors formulé, utilisant *ief* et la fréquence du terme dans chaque élément. Les poids des termes de la requête peuvent être étendus avec des conditions sur l'appartenance du terme à un certain élément ou chemin.
On trouvera d'autres approches basées sur les modèles de langages dans [127, 4, 192, 147, 104].

Dans [156, 154], on trouve un exemple d'utilisation des ***réseaux bayésiens*** à la recherche d'information structurée. La structure de réseau bayésien utilisée reflète directement la hiérarchie des documents, c'est à dire que les auteurs considèrent que chaque élément de la hiérarchie possède une variable aléatoire associée. La variable aléatoire associée à un élément structurel peut prendre 3 valeurs différentes dans l'ensemble $V = \{N, G, E\}$, avec N indiquant que l'élément n'est pas pertinent, G que l'élément est peu spécifique et E que l'élément possède une forte spécificité.
Pour chaque élément e et pour une requête donnée q, la probabilité $P(e = E|q)$ donne le score de pertinence *final* de l'élément, qui permet ensuite de classer les éléments selon leur degré de pertinence.
Deux autres types de variables aléatoires sont considérés. Le premier est la requête, qui est représentée par un vecteur de fréquences de termes. Le second est associé aux modèles de pertinence utilisés pour évaluer la similarité locale de l'élément à la requête et peut prendre deux valeurs : *pertinent* ou *non pertinent*.
Pour une requête donnée, un score local de pertinence est calculé pour chaque élément. Ce score dépend uniquement de la requête et du contenu de l'élément. Pour calculer ce score local, plusieurs modèles peuvent être utilisés. La fréquence des termes de la requêtes dans la requête, dans l'élément, dans le parent de l'élément et la longueur de l'élément peuvent par exemple être utilisés comme paramètres par les modèles.
La probabilité qu'un élément soit dans l'état N, G ou E dépend ensuite de l'état de l'élément parent, et du jugement par le(s) modèle(s) de pondération utilisé(s) que l'élément est pertinent ou non pertinent, comme le montre la figure 2.16.

On a alors (si on considère deux modèles de base M_1 et M_2 pour le calcul du score local de l'élément) :

$$P(e = v|q) = \sum_{v_p \in V, r_1, r_2 \in \{R, \neg R\}} \theta_{c(e), v, v_p, r_1, r_p} * P(e\ parent = v_p) * P(M_1 = r_1|q) * P(M_2 = r_2|q)$$

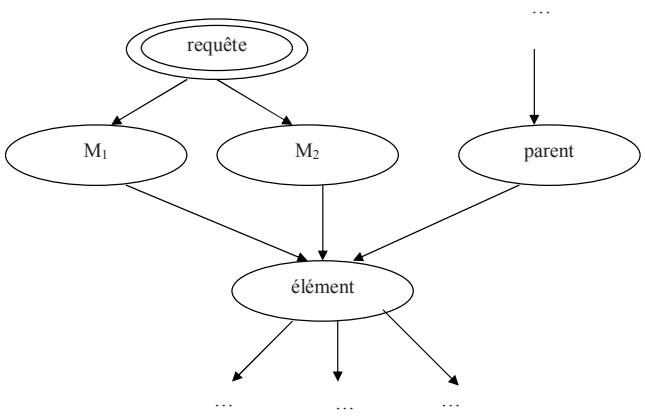


FIG. 2.16 – Modèle de réseau bayésien. L'état de l'élément dépend de l'état du parent et de la pertinence de l'élément pour les modèles M_1 et M_2

(2.11)

où $v \in V$, q est une requête composée de simples termes, et θ est un paramètre obtenu par apprentissage. Il dépend des différents états des 4 variables aléatoires (état de l'élément, état du parent, pertinence des modèles de base M_1 et M_2), et de la catégorie $c(e)$ de l'élément.
Les scores de pertinence sont calculés récursivement dans le réseau bayésien en commençant par la racine des documents.
Le modèle est étendu au traitement des requêtes orientées contenu et structure dans [212].

2.6.3 Remarques concernant le traitement de la structure

La plupart des modèles proposés dans la littérature ne proposent pas d'approches réellement orientées RI pour le traitement des conditions de structure des requêtes. Certaines approches proposent par exemple d'effectuer un simple filtre sur les résultats des conditions de contenu [192, 201]. D'autres approches, indépendantes des modèles de pondération de termes utilisés, existent cependant, et cherchent à évaluer la pertinence des conditions structurelles. Nous nous proposons de les décrire dans le paragraphe suivant.

2.6.3.1 Approches orientées RI pour le traitement de la structure

Dans [27], les auteurs proposent le langage FXpath (*Fuzzy XPath*), possédant les caractéristiques suivantes :

- une *correspondance d'arbres floue*, ce qui permet de renvoyer à l'utilisateur une liste triée d'éléments et non un ensemble non-ordonné comme le fait XPath
- des *prédicats flous*, permettant à l'utilisateur de spécifier des conditions de sélection imprécises et approximatives (introduction d'un prédicat *NEAR* et d'un prédicat *CLOSE*),
- une *quantification floue*, permettant la spécification d'opérateurs linguistiques comme opérateurs d'agrégation (par exemple *tout*, *au moins un*, *la plupart*, ...)

D'autres approches cherchent elles-aussi à effectuer la correspondance entre l'arbre du document et l'arbre de la requête [186, 135, 229].
Dans [229], l'auteur définit la notion de proximité à l'aide de distances. Dans des documents structurés, la distance peut être définie en terme de nombres de mots entre des termes de noeuds feuilles ou en termes de noeuds entre les noeuds. La distance des noeuds peut être quantifiée grâce à la distance horizontale (nombre de noeuds du même niveau entre les noeuds) et à la distance verticale (nombre d'unités logiques qui peuvent être groupées pour aller d'un noeud à un autre).

2.6.3.2 Le problème des corpus hétérogènes

L'interrogation de corpus hétérogènes (c'est à dire composés de documents suivant des DTD différentes) reste un problème ouvert : les conditions de structures exprimées par les utilisateurs dans la requête ne correspondent pas forcément exactement aux DTD des documents présents dans le corpus, mais ces derniers pourraient pourtant être pertinents pour l'utilisateur. Alors que les approches que nous avons présentées jusqu'ici cherchent à vérifier des *correspondances syntaxiques* entre les arbres de la requête et des documents, les approches pour les corpus hétérogènes cherchent quant à elles à vérifier des *correspondances sémantiques*.
Une première solution est d'utiliser un lexique, un thésaurus ou une ontologie pour faire correspondre les conditions de structures exprimées dans la requête avec les types d'éléments effectivement présents dans la collection [201].
D'autres approches, comme celle proposée par Denoyer et al. dans [57] ou Abiteboul et al. dans [2] visent à proposer un format médian dans lequel tous les documents du corpus (et éventuellement les requêtes) peuvent être transformés pour ensuite appliquer des techniques traditionnelles de traitement des requêtes structurées.

Depuis 2004, une tâche visant à proposer des solutions pour l'interrogation de corpus hétérogènes a été introduite dans INEX [198], et permettra d'évaluer ces différentes approches.

2.6.4 Conclusion

Comme nous venons de le voir, l'adaptation des modèles de RI traditionnels à la RI structurée n'est pas un problème trivial. Un premier problème est rencontré au niveau de la pondération des termes, qui devrait idéalement tenir compte de l'importance du terme au sein de l'élément, du document, et de la collection.
Un second problème concerne l'attribution des scores de pertinence aux noeuds des documents XML. Certaines approches calculent les scores des noeuds en propageant les termes dans l'arbre du document [4, 192, 104], alors que d'autres déduisent les scores de pertinence des noeuds en propageant la pertinence des noeuds feuilles [75, 84, 111, 170, 12]. Quelle que soit l'approche utilisée, la pertinence d'un noeud est fonction de la pertinence de ses descendants, et éventuellement de la pertinence de ses ancêtres [136, 156].
Un dernier problème concerne enfin le traitement des conditions de structure, pour lesquelles des méthodes basées sur la correspondance d'arbre ont été développées. Notons enfin que si les approches proposées cherchent à attribuer une pertinence aux éléments de structure, elles partent du principe que l'utilisateur a clairement exprimé le type des éléments qu'il désire voir retournés. Cela ne devrait pourtant pas être toujours le cas : les conditions de structure devrait permettre à ce dernier de préciser son besoin, et non forcément de restreindre le champ des recherches. Des solutions doivent alors être proposées pour la *sélection et le tri* des éléments de structure.

2.7 Evaluation

2.7.1 La campagne d'évaluation INEX

INEX (*Initiative for the Evaluation of XML Retrieval*) est à ce jour la seule campagne d'évaluation des différents SRI pour la recherche d'information sur des documents XML. Elle a lieu en 2005 pour la quatrième année consécutive. Le but principal d'INEX est de promouvoir l'évaluation de la recherche sur des documents XML en fournissant une collection de test, des procédures d'évaluation et un forum pour permettre aux différentes organisations participantes de comparer leurs résultats.
La collection de test consiste en un ensemble de documents XML, requêtes et jugements de pertinence. Les requêtes et les jugements de pertinence associés sont obtenus grâce à la collaboration des participants.

2.7.1.1 Collection

La collection INEX est composée d'articles scientifiques provenant de la IEEE Computer Society, balisés au format XML. La collection, d'environ 500 Mo, contient plus de 12000 articles, publiés de 1995 à 2002, et provenant de 18 magazines ou revues différents.
Les articles sont généralement composés d'une en-tête (<*fm*>), d'un corps (<*body*>) et d'annexes (<*bm*>). Chacun de ces éléments se rédécompose : par exemple, le corps est composé de section <*sec*> elles-mêmes composées de paragraphes <*p*> et les annexes sont composées de référence bibliographiques <*bibl*> et éventuellement de curriculum vitae <*vt*>. On trouvera un extrait d'un document de la collection dans le tableau 2.11.

Un article moyen est composé d'environ 1500 éléments, et la profondeur moyenne des documents est de 6.9. Au total, la collection contient 8 millions de noeuds et 192 balises différentes.

```
<?xml version="1.0" ?>
<!DOCTYPE article SYSTEM "/usr/projects/inex/2004/inex/dtd/xmlarticle.dtd">
<article>
    <fno>A3036</fno>
    <fm>
        <hdr>
            <hdr1>
                <ti>IEEE ANNALS OF THE HISTORY OF COMPUTING</ti>
            </hdr1>
            <hdr2>
                <obi>
                    <volno>Vol. 18</volno>
                    <issno>No. 3</issno>
                </obi>
                <pdt> <yr>1996</yr> </pdt>
                <pp>pp. 36-42</pp>
                </hdr2>
        </hdr>
        <tig>
            <atl> Women in Computing : Historical Roles, the Perpetual
            Glass Ceiling, and Current Opportunities </atl>
        </tig>
        <au sequence="first">
            <fnm>AMITA</fnm>
            <ref aid="a3036a1" type="prb">GOYAL</ref>
        </au>
        <abs>
            <p> <it>
            Over the course of history, women have slowly begun to hold influential
            roles in the computing industry. Although progress has been made,
            the precipitous journey is not yet complete. This paper presents a
            historical analysis of the entrance and role of women in the computing
            industry, a discussion on the existence and impact of the glass ceiling,
            and a detailed and informative collection of programs and opportunities
            established to abet women in succeeding in the industry. The information
            compiled in this work will prove useful not only to the women already
            employed in the industry but also to women contemplating entrance.
            </it> </p>
        </abs>
    </fm>
```

```
<bdy>
    <sec>
        <st>Introduction</st>
        <ip1>
        Over the course of history, the demographics of the workforce have changed
        drastically. Women have slowly emerged as able participants in the
        workforce and have even progressed to hold influential roles and positions.
        Women have accounted for 60% of the total labor force growth between
        1982 and 1992, experiencing their highest labor force participation
        rate of 57.8% in 1992. At this time, of the 100 million women 16
        years and older in the United States, 58 million are active in the labor force
        <ref rid="biba303626" type="bib">[26]</ref> .
        </ip1>
        ...
    </sec>
    <sec>
        <st>Historical Roles</st>
        <ip1>
        Women were the prominent early users of computers. Some even say that
        the first computer user was a woman ! During World War II, most
        men were in the armed forces, affording women the opportunity to be
        the early pioneers. In those days, <it>calculators</it> or
        <it>computers</it> was the term given to people, primarily women, who
        were doing hand calculations using desk calculators. As women moved into
        programming, they usually became application programmers,
        programming scientific problems in math and physics and working on
        applying numerical methods to computers. Women were
        often stereotyped as being good candidates for programming :
        "Programming requires lots of patience, persistence and a capacity
        for detail and those are traits that many girls have"
        <ref rid="biba303616" type="bib">[16]</ref> .
        </ip1>
    </sec>
</bdy>
<bm>
    <bib>
        <bibl>
            <h>References</h>
            <bb id="biba30361">
```

```
                <au>
                    <fnm>A.</fnm>
                    <snm>Adam</snm>
                </au>
                <atl>"Women and Computing in the UK,"</atl>
                <ti>Comm. ACM,</ti>
                <obi>
                    <volno>vol. 38,</volno>
                    <issno>no. 1,</issno>
                </obi>
                <pp>p. 43,</pp>
                <pdt>
                    <yr>1995.</yr>
                </pdt>
            </bb>
            ...
        </bibl>
    </bib>
    <vt id="a3036a1">
        <p>
            <fig>
                <art file="a3036a1.gif" w="150" h="187" tw="150" th="187"/>
            </fig>
            <b>Amita Goyal</b>
            is an assistant professor in the Information Systems Department at
            Virginia Commonwealth University. She received her BS in computer
            science and MS and PhD in information systems, all from the University
            of Maryland at College Park. Her research interests include
            distributed database systems, women in technology, and information
            systems curricula. In June 1995, Dr. Goyal served as program chair
            for EDSIG's Worldwide Conference on Information Systems Education
            (WISE 1995). Dr. Goyal is a member of the Association of NeXTSTEP
            Developers, Inc. (ANDI), the International Association for
            Mathematical and Computer Modelling (IAMCM), and the Information
            Resources Management Association (IRMA).
            </p>
        </vt>
    </bm>
</article>
```

TAB. 2.11 – Exemple de document XML de la collection INEX

2.7.1.2 Requêtes

Les requêtes (ou Topics) sont crées par les différents participants et doivent être représentatives des demandes de l'utilisateur moyen sur la collection. Les topics se divisent en deux catégories principales :

- Les CO (*Content Only*) : ce sont des requêtes en langage naturel, comme celles utilisées dans TREC. Les mots-clés de la requête peuvent être éventuellement groupés sous forme d'expressions et précédés par les opérateurs '+' (signifiant que le terme est obligatoire) ou '-' (signifiant que le terme ne doit pas apparaître dans les éléments renvoyés à l'utilisateur).
- Les CAS (*Content And Structure*) : ces requêtes contiennent des contraintes sur la structure des documents, comme par exemple des conditions de contenu de tel ou tel élément.

Pour chaque *Topic*, différents champs permettent d'expliciter le besoin de l'auteur : le champ *Title* donne la définition formelle de la requête, le champ *Keywords* contient un ensemble de mots-clés qui ont permis l'exploration du corpus avant la formulation définitive de la requête, et les champs *Description* et *Narrative*, explicités en langage naturel, indiquent les intentions de l'auteur [193]. La formulation des requêtes est étroitement liée à la tâche de recherche associée. Nous donnons donc quelques exemples de requêtes dans la section suivante.

2.7.1.3 Tâches

La tâche principale d'INEX est la tâche de recherche *ad-hoc*. Comme en recherche d'information traditionnelle, la recherche ad-hoc est considérée dans INEX comme une simulation de l'utilisation d'une bibliothèque, où un ensemble statique de documents est interrogé avec des besoins utilisateurs, c'est à dire des requêtes. Les requêtes peuvent contenir à la fois des conditions structurelles ou de contenu, et en réponse à une requête, des éléments (et non forcément des documents) peuvent être retrouvés à partir de la bibliothèque. La tâche ad-hoc se divise en trois sous-tâches : les tâches CO, SCAS et VCAS.

Tâche CO La tâche CO (*Content Only Task*) a pour but de répondre avec des éléments/documents XML à des requêtes utilisateur CO, c'est à dire des requêtes contenant de simples mots-clés. Aucune indication de structure dans la requête ne peut aider les SRI à déterminer la granularité de l'information à renvoyer.
On trouvera un exemple de requête CO dans le tableau 2.12.

```
<inex_topic topic_id="98" query_type="CO">
        <title> "Information Exchange" +"XML" "Information Integration"</title>
        <description> How to use XML to solve the information exchange (information
        integration) problem, especially in heterogeneous data sources ? </description>
        <narrative> Relevant documents/components must talk about techniques of
        using XML to solve information exchange (information integration) among
        heterogeneous data sources where the structures of participating data sources
        are different although they might use the same ontologies about the same
        content. </narrative>
        <keywords> Information exchange, XML, information integration,
        heterogeneous data sources </keywords>
</inex_topic>
```

TAB. 2.12 – Exemple de requête CO, issue du jeu de test 2003

Tâche SCAS La tâche SCAS (*Strict Content And Structure Task*) consiste à répondre avec des éléments/documents XML aux topics CAS de manière stricte, c'est à dire en respectant toutes les conditions sur la structure et le contenu énoncées dans les requêtes. Le champ *Title* des requêtes de la tâche SCAS est basé sur une syntaxe XPath. On trouvera un exemple de requête CAS pour la tâche SCAS dans le tableau 2.13.

```
<inex_topic topic_id="64" query_type="CAS">
        <title> //article[about(./,'hollerith')] // sec[about(./, 'DEHOMAG')] </title>
        <description> In articles discussing Herman Hollerith find sections that
        mention DEHOMAG </description>
        <narrative> Relevant sections deal with DEHOMAG (Deutsche Hollerith
        Maschinen Gesellschaft) in documents that discuss work or life of Herman
        Hollerith </narrative>
        <keywords> Hollerith, DEHOMAG, Deutsche Hollerith Maschinen Gesellschaft
        </keywords>
</inex_topic>
```

TAB. 2.13 – Exemple de requête CAS, issue du jeu de test 2003

Tâche VCAS La tâche VCAS (*Vague Content And Structure Task*) utilise elle-aussi des requêtes CAS, mais pour lesquelles les participants peuvent répondre de manière vague, c'est à dire avec des éléments/documents qui satisfont globalement les requêtes. Le champ *Title* des requêtes de la tâche SCAS est basé sur le langage de requêtes NEXI [206, 207], l'extension de XPath utilisée

en 2003 pour les requêtes CAS étant considérée comme trop complexe [207] : 63% des requêtes exprimées par les participants (experts en RI) contenaient des erreurs de syntaxe ! On trouvera un exemple de requête CAS pour la tâche VCAS dans le tableau 2.14.

```
<inex_topic topic_id="149" query_type="CAS">
        <title> //article[about(.//(abs|kwd),"genetic algorithm")]
        // bdy//sec[about(., simulated annealing)] </title>
        <description>Find sections about simulated annealing in article that mention
        genetic algorithms. </description>
        <narrative> I have come across the Constrained Shortest Path problem in
        connection with a route planing program. I have become aware of a technique
        called Simulated Annealing known from combinatorial optimization for heuristic
        solutions to NP-hard problems that I wish to use in the route plaing program.
        I have noticed a tendency for authors that mention SA in combination with
        Genetic Algorithms so I expect the keyword 'genetic' to appear in relevant
        articles. For the section to be relevant it has to discuss usage of
        Simulating Annealing or refer to results relevant to the techique.
        </narrative>
        <keywords> genetic, simulated annealing, optimization </keywords>
</inex_topic>
```

Tab. 2.14 – Exemple de requête CAS, issue du jeu de test 2004

Autres tâches En 2004, quatre nouvelles tâches ont été proposées aux participants :

- la tâche de "*relevance feedback*", qui a pour but d'expérimenter l'utilisation du contenu ET de la structure comme informations de base pour la formulation d'une nouvelle requête,
- la tâche de *langage naturel*, dans laquelle les utilisateurs formulent leurs requêtes en langage naturel, et donc sans avoir besoin d'apprendre un langage complexe,
- la tâche *interactive* qui a pour but d'étudier le comportement des utilisateurs face à des corpus XML et donc de cerner au mieux leurs besoin,
- et la tâche *hétérogène*, qui propose aux participants de nouvelles collections, afin de développer des approches indépendantes des DTDs.

2.7.1.4 Jugements de pertinence

L'évaluation de la pertinence des SRI passe par une première phase de validation des documents renvoyés par les SRI. Chaque élément/document est jugé

à la main (par les participants) pour chaque requête, en utilisant le système de jugement en ligne [155, 158]. En 2002, une première échelle de pertinence à deux dimensions a été proposée, basée sur le *degré de pertinence* et la *couverture* des éléments. Depuis la campagne d'évaluation 2003, les dimensions de pertinence et de couverture ont été remplacées par les dimensions d'*exhaustivité* et *spécificité*. La notion d'exhaustivité décrit jusqu'à quel point l'élément discute du sujet de la requête. Une échelle à 4 niveaux est proposée :

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

La notion de spécificité décrit jusqu'à quel point l'élément de focalise sur le sujet de la requête. Une nouvelle échelle à quatre niveaux est proposée :

- Pas spécifique : le sujet de la requête n'est pas un thème de l'élément
- Marginalement spécifique : le sujet de la requête est un thème mineur de l'élément
- Assez spécifique : le sujet de la requête est un thème majeur de l'élément
- Très spécifique : le sujet de la requête est le seul thème de l'élément

L'utilisation d'une échelle à deux dimensions est motivée par le besoin de refléter la pertinence relative d'un élément par rapport à ses descendants. Par exemple, un élément peut être plus exhaustif que chacun de ses descendants pris séparemment car il couvre tous les aspects (ou plutôt l'union des aspects) discutés dans chacun de ses descendants. De la même manière, des éléments peuvent être plus spécifiques que leurs parents, car ces derniers couvrent plus de sujets, y compris des sujets non pertinents.

Il y a 10 valeurs possibles sur l'échelle, puisque comme pour la couverture et la pertinence, les deux dimensions ne sont pas tout à fait orthogonales (par exemple, lorsque l'élément n'est pas exhaustif, il ne peut pas être spécifique, et inversement). La combinaison des deux dimensions est utilisée pour identifier les éléments pertinents. Le degré de pertinence d'un élément jugé par les participants est donné par la paire (e, s), avec $(e, s) \in ES$ et $ES = \{(0,0), (1,1), (1,2), (1,3), (2,1), (2,2), (2,3), (3,1), (3,2), (3,3)\}$.

On trouvera un exemple de d'arbre XML et de jugements de pertinence associés dans la figure 2.17.

2.7.1.5 Evaluation

L'évaluation de la performance des différents systèmes proposés par les participants utilise des méthodes basées sur les mesures de rappel et précision, en cherchant à prendre en compte la structure des documents XML et la pos-

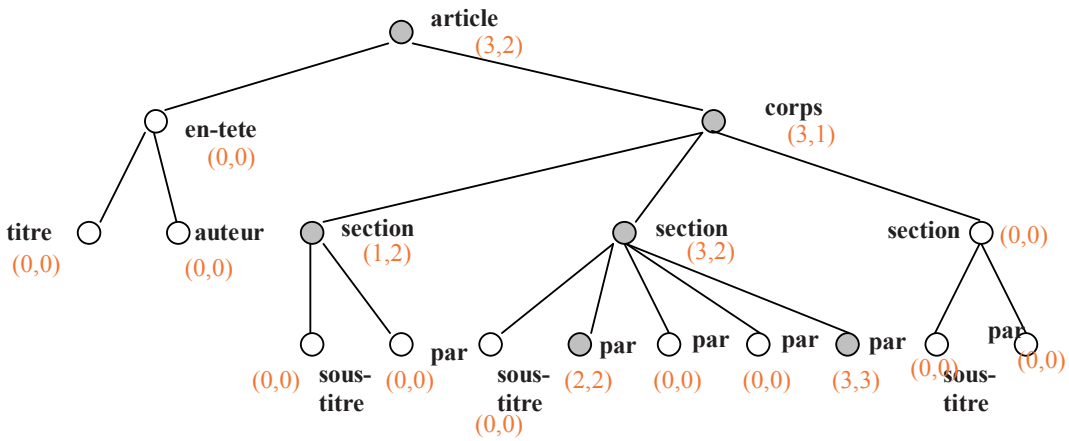


FIG. 2.17 – Exemple de jugements de pertinence

sible imbrication des résultats. Ces mesures sont décrites plus en détail dans la section suivante.

Les résultats et réflexions issus des premières campagnes d'évaluation INEX et des workshops qui ont suivi sont disponibles dans [74, 76, 77].

2.7.2 Mesures d'évaluation

Parallèlement à la mise en place de la campagne d'évaluation INEX, une reflexion (nécessaire) sur les mesures d'évaluation a été conduite et un certain nombre de mesures ont été proposées. Ces mesures étendent les mesures traditionnelles utilisées en RI dans le but de traiter les besoins supplémentaires induits par la recherche d'information dans des documents XML. Un des principaux problèmes, comme nous allons le voir, est le retour possible d'éléments imbriqués. Par exemple, une section et l'un de ses paragraphes peuvent être retournés à l'utilisateur à différents rangs dans la liste des résultats.
Avant de définir ces différentes mesures, notons qu'elles se basent toutes sur les hypothèses suivantes de comportement de l'utilisateur : (1) l'utilisateur suit la liste ordonnée des résultats qui lui sont renvoyés, en commençant par le premier élément, et (2) un élément pertinent est encore pertinent même si l'utilisateur a déjà vu la même information plus haut dans la liste de résultats.

La performance d'un système a été définie comme la capacité d'un système à retrouver des éléments à la fois exhaustifs et spécifiques au sujet de la requête. Une première mesure suivant ce critère d'évaluation a été définie lors de la campagne d'évaluation 2002 [74, 110]. Cette mesure applique la mesure de *precall* [231, 162] (voir section 1.4) aux éléments de documents XML et calcule

la probabilité $P(rel|retr)$ qu'un élément vu par l'utilisateur est pertinent :

$$P(pert|retr)(x) = \frac{x.n}{x.n + esl_{x.n}} \tag{2.12}$$

où $esl_{x.n}$ est la longueur supposée de recherche (*expected search length* [51]), c'est à dire le nombre attendu d'éléments non pertinents retrouvés jusqu'à ce qu'un point de rappel x soit atteint, et n est le nombre de documents pertinents dans la collection étant donnée une certaine requête.
Pour appliquer la mesure ci-dessus, les deux dimensions de pertinence (exhaustivité et spécificité) doivent être agrégées en une seule valeur. Deux types de fonctions d'agrégation utilisant les jugements de pertinence définis dans la section 2.7.1.4 ont été définies :

- une agrégation "stricte" pour évaluer si un SRI est capable de retrouver des éléments très spécifiques et très exhaustifs

$$f_{strict}(e,s) = \begin{cases} 1 \ si \ e = 3 \ et \ s = 3 \\ 0 \ sinon \end{cases} \tag{2.13}$$

- une agrégation "généralisée" pour évaluer les éléments selon leur degré de pertinence

$$f_{generalisee}(e,s) = \begin{cases} 1 \ si \ (e,s) = (3,3) \\ 0.75 \ si \ (e,s) \in \{(2,3),(3,\{2,1\})\} \\ 0.5 \ si \ (e,s) \in \{(1,3),(2,\{2,1\})\} \\ 0.25 \ si \ (e,s) \in \{(1,2),(1,1)\} \\ 0 \ si \ (e,s) = (0,0) \end{cases} \tag{2.14}$$

En 2004, d'autres fonctions d'agrégation ont été introduites. L'équation 2.14 accorde une préférence à la notion d'exhaustivité, attribuant de bons scores à des éléments exhaustifs mais pas forcément spécifiques. Ces éléments sont généralement de grande taille (comme des articles entier par exemple), ce qui implique que de bons résultats peuvent être obtenus en renvoyant les documents dans leur entier et non des parties de documents. Afin de résoudre ce problème, une fonction d'agrégation "généralisée" orientée spécificité a été définie [109] :

$$f_{sog}(e,s) = \begin{cases} 1 \ si \ (e,s) = (3,3) \\ 0.9 \ si (e,s) = (2,3) \\ 0.75 \ si \ (e,s) \in \{(1,3),(3,2)\} \\ 0.5 \ si \ (e,s) = (2,2) \\ 0.25 \ si \ (e,s) \in \{(1,2),(3,1)\} \\ 0.1 \ si \ (e,s) \in \{(2,1),(1,1)\} \\ 0 \ si \ (e,s) = (0,0) \end{cases} \tag{2.15}$$

Parallèlement, deux classes de fonctions d'agrégation ont été définies : on parle maintenant de fonctions orientées spécificité et de fonctions orientées exhaustivité. Les fonctions orientées spécificité (equations 2.16 et 2.17) considèrent

uniquement les éléments ayant le plus haut degré de spécificité, tandis que les fonctions orientées exhaustivité (équations 2.18 et 2.19) ne considèrent que les éléments ayant le plus haut degré d'exhaustivité [107].

$$f_{s3\_e321}(e,s) = \begin{cases} 1 \ si \ e \in \{3,2,1\} \ et \ s = 3 \\ 0 \ sinon \end{cases} \tag{2.16}$$

$$f_{s3\_e32}(e,s) = \begin{cases} 1 \ si \ e \in \{3,2,\} \ et \ s = 3 \\ 0 \ sinon \end{cases} \tag{2.17}$$

$$f_{e3\_s321}(e,s) = \begin{cases} 1 \ si \ s \in \{3,2,1\} \ et \ e = 3 \\ 0 \ sinon \end{cases} \tag{2.18}$$

$$f_{e3\_s32}(e,s) = \begin{cases} 1 \ si \ s \in \{3,2,\} \ et \ e = 3 \\ 0 \ sinon \end{cases} \tag{2.19}$$

Toutes ces mesures sont ensuite combinées pour calculer une précision moyenne, qui a été utilisée pour établir les classements officiels des participants lors de la campagne 2004 (alors qu'en 2002 et 2003 seules les équations 2.13 et 2.14 ont été utilisées).
Ces mesures présentent cependant un inconvénient majeur : elles ne prennent pas en compte l'imbrication (*overlap*) des éléments et évaluent le retour d'un élément pertinent sans prendre en compte le fait qu'il ait été déjà peut-être vu entièrement ou en partie par l'utilisateur. Par exemple, un système A renvoyant une section pertinente et aussi un de ses paragraphes pertinent obtient les mêmes performances qu'un système B renvoyant deux éléments pertinents non imbriqués.
En 2003, une nouvelle mesure a été fournie pour essayer de résoudre ce problème [76, 144]. Cette mesure incorpore la taille des éléments et le concept d'imbrication dans les mesures de rappel et précision (équations 2.20 et 2.21). Au lieu de mesurer le rappel et la précision après qu'un certain nombre d'éléments aient été retrouvés, la taille totale de l'élément retrouvé est utilisée comme paramètre de base, alors que l'imbrication est traitée en ne considérant que les parties de l'élément qui n'aient pas déjà été vues (on considère alors que l'information pertinente est répartie uniformément au sein d'un élément).

$$rappel_o = \frac{\sum_{i=1}^{k} e(c_i).\frac{|c'_i|}{|c_i|}}{\sum_{i=1}^{N} e(c_i)} \tag{2.20}$$

$$precision_o = \frac{\sum_{i=1}^{k} s(c_i).|c'_i|}{\sum_{i=1}^{k} |c'_i|} \tag{2.21}$$

Les éléments $c_1, ..., c_k$ des équations 2.20 et 2.21 forment une liste triée de résultats, N est le nombre total d'éléments dans la collection,$e(c_i)$ et $s(c_i)$ sont les valeurs d'exhaustivité et spécificité de l'élément c_i, $|c_i|$ est la taille de l'élément et $|c'_i|$ est la taille de l'élément qui n'a pas été précédemment vu

par l'utilisateur. Comme cette mesure traite les deux dimensions de pertinence séparément, de nouvelles fonctions ont été définies pour fournir une normalisation séparée de l'exhaustivité et la spécificité [144].
Parallèlement, B. Piwowarski et P. Gallinari ont proposé dans [157] la mesure ERR (*Expected Ratio of Relevant Units*). La définition de cette mesure est basée sur le comportement hypothétique d'un utilisateur. Trois hypothèses sont faites sur le comportement de ce dernier :

- L'utilisateur consulte le contexte structurel (parent, enfants, frère) d'un élément retourné. Cette hypothèse est relative à la structure intrinsèque des documents
- La spécificité d'un élément influence le comportement de l'utilisateur
- L'utilisateur n'utilise aucun lien, c'est à dire qu'il ne va pas naviguer vers un autre document. Cette hypothèse est valide dans le cadre de la campagne INEX, mais pourrait être aisément supprimée pour traiter des corpus de documents hypertextes.

La mesure ERR est alors l'espérance du nombre d'élements pertinents qu'un utilisateur voit quand il consulte la liste des premiers éléments retournés par l'espérance du nombre d'éléments pertinents qu'un utilisateur voit s'il explore tous les éléments du corpus. Cette mesure est normalisée et peut donc être moyennée sur plusieurs requêtes.

Cependant, les mesures décrites ci-dessus ne prennent pas en compte un problème essentiel de l'évaluation : *la surpopulation de la base de rappel* [109]. Cette surpopulation est due aux règles d'inférence utilisées lors de l'élaboration des jugements de pertinence [155] : si un noeud est jugé pertinent, ses ancêtres doivent aussi être jugés pertinents, même si leur degré de pertinence est moindre (et ce notamment à cause de la propagation de l'exhaustivité dans l'arbre du document). Par conséquent, un taux de rappel idéal ne peut être obtenu que par les systèmes référençant tous les composants de la base de rappel, y compris les éléments imbriqués.
Afin de solutionner ce problème, Gabriella Kazai *et al.* établissent dans [109] la définition d'une base de rappel idéale, qui supporterait la procédure d'évaluation suivante : les éléments de la base de rappel idéale *doivent* être retournés par les systèmes, les éléments proches de ceux contenus dans la base de rappel idéale *peuvent* être vus comme des succès partiels, mais les autres systèmes *ne doivent pas* être pénalisés s'ils ne les renvoient pas. Les mesures XCG sont proposées pour répondre à ces besoins. Les mesures XCG (*XML Cumulated Gain*) sont des extensions du "gain cumulatif" proposé par Järvelin et Kekäläinen dans [102]. Les mesures de gain cumulatif ont été développées pour évaluer les systèmes selon le degré de pertinence des documents retournés. La motivation derrière XCG est d'étendre les mesures de gain culumatif au problème des éléments imbriqués. Les premiers tests de fiabilité de la mesure sont encourageants [108], mais le comportement de la mesure doit encore être évaluée dans le cadre de la campagne d'évaluation INEX.

Comme nous venons de le voir, les problèmes soulevés par l'évaluation des SRI strucurés sont nombreux et loin d'être résolus. Ceci s'explique par la "jeunesse" des recherches dans le domaine, l'évaluation de la RI structurée étant née avec la campagne d'évaluation INEX. De plus, la définition précise des tâches utilisateurs modélisées dans INEX permettrait de fixer de nombreuses problématiques. La tâche interactive initiée en 2003 permet d'étudier le comportement des utilisateurs et pourra donc être utilisée dans ce but. En décembre 2004, lors du 3ème workshop INEX, trois tâches utilisateurs ont été identifiées :

1. trouver les éléments les plus spécifiques dans un chemin,
2. trouver autant de contenu pertinent que possible,
3. trouver autant d'élements pertinents que possible.

De nouvelles mesures, beaucoup plus simples, ont été associées à ces tâches [72]. Parmi elles, on peut citer : le nombre d'éléments les plus spécifiques retournés sur le nombre d'éléments retournés, la somme des contenus pertinents retrouvés sur la somme des tailles des contenus retrouvés, le nombre de sous-arbres non imbriqués sur le nombre d'éléments retrouvés, ...

2.8 Interface et Visualisation

Les interfaces pour les systèmes gérant des documents XML peuvent prendre en considération trois facettes du processus de recherche : l'indexation des documents, l'interrogation et la visualisation des résultats.

Le module XyView de Xyleme Zone Server [227] permet de construire une vue unique et adaptée aux besoins d'utilisateur, et ce, quelle que soit la variété des DTDs ou schémas utilisés pour tous les documents. Xyview est utilisé pour créer une DTD virtuelle, appelée une vue abstraite qui décrit au mieux la variété de structures. XyView associe automatiquement les balises de la vue abstraite avec celles des différentes DTDs. Il est aussi possible de rajouter ou modifier les balises proposées par XyView afin de rendre cette vue abstraite la plus pertinente possible. Une fois la vue abstraite créée, les utilisateurs peuvent formuler des requêtes sur de multiples documents ayant chacun sa structure propre comme s'ils partageaient tous la même structure.

En ce qui concerne la formulation des requêtes, des formulaires peuvent s'avérer intéressants. On peut par exemple citer XForms [61] et XML Forms language [118]. Dans le système XYZFind [62], l'interface pour la formulation de requêtes est adaptative : le système peut engager un dialogue avec l'utilisa-

teur pour l'aider dans sa formulation.

Pour afficher des résultats de recherche complexes, la meilleure méthode serait peut-être d'utiliser une représentation en "accordéon" (comme par exemple la représentation des répertoires sous Windows Explorer). Le système XMLFS [15] se présente à l'utilisateur comme n'importe quel système de gestion de fichiers. XMLFS crée automatiquement une organisation en répertoires de collections de documents XML. Cette organisation, basée sur le contenu et le contexte, permet à l'utilisateur de naviguer à travers le système de fichiers selon son domaine d'intérêt. La différence entre XMLFS et un système de fichiers traditionnel est que XMLFS montre les fichiers organisés selon une hiérarchie dynamique construite à la volée. On trouvera un exemple de navigation dans la figure 2.18.

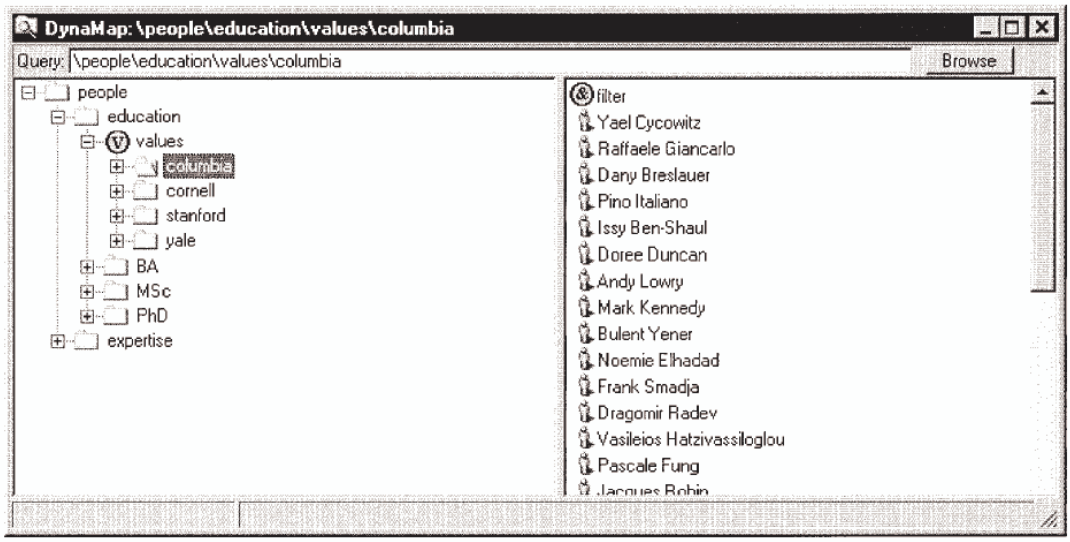


FIG. 2.18 – Exemple de navigation XML avec le système XMLFS

Le but des SRI pour les documents structurés, est, comme nous l'avons vu, de renvoyer des parties de documents les plus spécifiques et exhaustives possibles à l'utilisateur. Pour les besoins de la tâche interactive d'INEX, deux interfaces pour la présentation des résultats du moteur de recherche HyreX ont été conçues [131]. La première interface renvoit une liste ordonnée de résultats, et lorsque l'utilisateur clique sur l'un de ces résultats, son contenu apparait dans la partie droite de l'écran, alors que l'arbre du document auquel il appartient ainsi que sa position surlignée dans l'arbre apparaissent dans la partie gauche. Cependant, les résultats sont de granularités variées, et les retourner dynamiquement à l'utilisateur peut conduire à la désorientation de ce dernier [170]. Afin de réduire ce phénomène, une stratégie de présentation pourrait

consister à retourner à l'utilisateur des super-éléments, composés de nombreux éléments pertinents, au lieu de renvoyer ces éléments pertinents directement. Les résultats sont alors affichés par document, ce qui permet à l'utilisateur de voir les éléments dans leur contexte, et non plus en tant que liste ordonnée d'éléments indépendants. C'est ce que propose la seconde interface conçue pour la tâche interactive. Cette interface a été conçue pour couvrir deux aspects de la recherche dans des corpus de documents structurés : la dépendance structurelle et hiérarchique entre les différents éléments, et la granularité variée des éléments retournés à l'utilisateur. Les résultats sont regroupés par document et un résumé de chaque document est présenté à l'utilisateur. Ce résumé est composé du titre du document, de ses auteurs, d'un rectangle gris indiquant son degré de pertinence et d'une barre rouge de longueur variée indiquant le nombre d'éléments pertinents dans le document. Le document est ensuite représenté par une carte appelé *TreeMap* [101]. Cette carte est en fait un rectangle coupé alternativement horizontalement et verticalement afin de représenter les différents niveaux du document. Par exemple, le rectangle peut être coupé horizontalement pour les noeuds de premier niveau, puis verticalement pour les noeuds de second niveau, et ainsi de suite. Afin d'éviter l'obtention de représentations trop denses, les concepteurs utilisent la notion de carte partielle, dans laquelle les noeuds non retrouvés (c'est à dire les noeuds non pertinents) ne sont pas affichés.

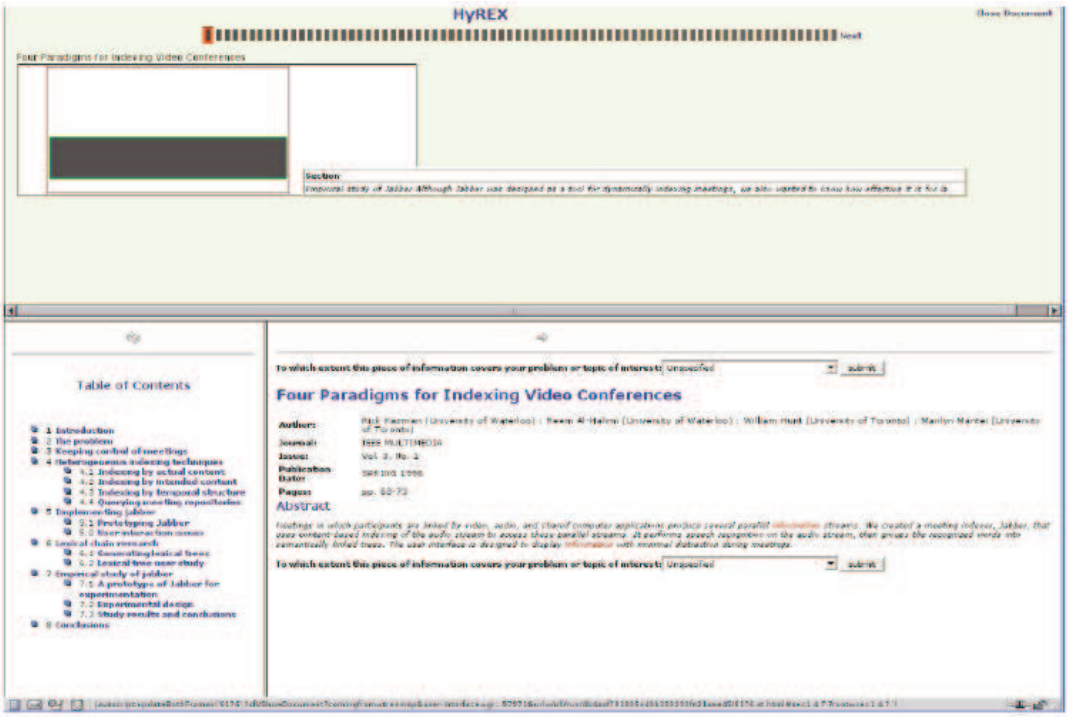


FIG. 2.19 – Interface de visualisation pour la tâche interactive d'INEX 2004

2.9 Conclusion

Devant le nombre croissant de documents semi-structurés et plus particulièrement de documents XML mis à disposition des utilisateurs, de nouveaux systèmes de recherche d'information utilisant au mieux leurs différentes caractéristiques doivent être développés. La dimension structurelle apportée au contenu textuel des documents permet de considérer l'information avec une autre granularité que le document tout entier. Le but pour les SRI est alors de renvoyer les parties de documents (ou unités d'information) les plus spécifiques et exhaustives à la requête utilisateur. Ces unités d'information doivent être auto-explicatives, c'est à dire qu'elles ne dépendent d'aucune autre pour être comprises par l'utilisateur.
Ce dernier peut formuler deux types de requêtes pour interroger des corpus de documents semi-structurés : (1) des requêtes contenant des conditions de structure et de contenu, pour lesquelles il doit avoir une connaissance au moins partielle de la DTD des corpus qu'il interroge, et dans lesquelles il spécifie le type de l'unité d'information qu'il désire voir renvoyée, et (2) des requêtes formées de simples mots-clés, pour lesquelles le SRI doit décider de la granularité de l'information à renvoyer.
Dans les deux cas, les approches proposées par la recherche d'information traditionnelle ne suffisent pas à intégrer la dimension structurelle, pourtant indispensable. De nouveaux modèles doivent donc être proposés pour l'indexation, l'interrogation et la recherche. Dans ce chapitre, nous avons présenté les différents modèles proposés dans la littérature pour répondre à ces problématiques. Ces modèles ont cependant certaines limites, et le modèle que nous présentons dans le chapitre 3 vise à répondre à certaines d'entre elles.

Deuxième partie

Un modèle flexible pour la Recherche d'Information structurée

Chapitre 3

XFIRM : XML Flexible Information Retrieval Model

3.1 Introduction

Nous avons présenté dans le deuxième chapitre un état de l'art des travaux pour la recherche d'information structurée. Afin de pouvoir retrouver de l'information pertinente au sein des documents XML, de nouvelles méthodes pour l'indexation, l'interrogation et la recherche ont été proposées. Tous ces modèles cherchent à utiliser l'information structurelle des documents pour retrouver les unités d'information les plus spécifiques et exhaustives au besoin de l'utilisateur.
La communauté des Bases de Données a été la première à proposer des solutions pour la recherche d'information structurée, notamment en ce qui concerne l'indexation des documents et l'interrogation des corpus. Cependant, ces approches, orientées données, cherchent à répondre façon *exacte* aux besoins de l'utilisateur. Les approches proposées par la communauté de la Recherche d'Information cherchent quant à elles à proposer des résultats qui correspondent *au mieux* aux besoins. Nos travaux s'inscrivent dans ce cadre, malgré quelques emprunts aux fonctionnalités des bases de données.
Afin de répondre à certaines limites des approches proposées dans la littérature (présentées dans la section 3.2), nous proposons XFIRM (*XML Flexible Information Retrieval Model*), un modèle flexible pour la recherche dans des documents semi-structurés, ayant pour but de répondre au mieux au critère de spécificité et exhaustivité demandé par l'utilisateur. Notre modèle évalue les requêtes grâce à une technique de propagation de la pertinence des noeuds dans l'arbre des documents. Afin de permettre cette propagation, nous proposons tout d'abord un modèle logique et physique générique de représentation des données. Un langage d'interrogation est ensuite défini afin de permettre à

l'utilisateur d'exprimer son besoin de manière plus ou moins précise, en introduisant ou non des conditions structurelles dans les requêtes. Notre méthode de propagation calcule un premier score de pertinence pour les noeuds feuilles (et ce grâce à la pondération des termes des feuilles) et propage ensuite cette pertinence dans l'arbre du document, en prenant en compte les importances diverses des descendants d'un noeud, mais aussi son contexte, grâce à la prise en compte de la pertinence de ses ancêtres. Les conditions de structure des requêtes peuvent quant à elles être traitées de manière stricte ou vague, grâce aux diverses propagations effectuées dans l'arbre des documents. Notre modèle apporte ainsi de la flexibilité dans la recherche à plusieurs niveaux : la représentation des documents (et par conséquent la structure d'index) est générique et permet de traiter des collections de documents hétérogènes, le langage permet à l'utilisateur d'exprimer son besoin selon plusieurs degrés de précision, et les conditions de contenu et les éventuelles conditions de structure des requêtes peuvent être traitées de manière vague.

Ce chapitre est organisé comme suit : la section 3.2 présente nos motivations, et dresse quelques limites des modèles proposés dans la littérature pour l'indexation, l'interrogation et la recherche dans des corpus de documents structurés. La section 3.3 présente de manière générale le modèle que nous proposons pour pallier ces limites. La section 3.4 présente le modèle logique de représentation des documents sur lequel se base notre proposition, et la section 3.5 présente le langage d'interrogation associé : ce langage autorise l'utilisateur à exprimer son besoin selon divers degrés de précision. Nous présentons ensuite la méthode de recherche que nous utilisons, en détaillant de manière séparée les recherches basées sur les seules conditions de contenu (section 3.6) et les recherches basées sur des conditions de structure et de contenu (section 3.7). Enfin, la section 3.8 décrit l'architecture du prototype que nous avons développé pour valider la faisabilité de notre approche.

3.2 Motivations

Les principales limites des travaux présentés dans le chapitre précédent se résument comme suit :

1. Tout d'abord, en ce qui concerne l'indexation des documents XML, certaines solutions proposées sont non-extensibles, c'est à dire que les index proposés sont dépendants de la structure des documents, qui est reflétée dans leur schéma [63, 20, 59]. Si ces approches sont capables de traiter la structure avec efficacité, elles nécessitent cependant une connaissance a-priori de la structure des documents, et des documents possédant une structure différente de celle de la collection originale ne peuvent pas être

ajoutés aux index.

Les solutions extensibles n'ont au contraire pas besoin de la DTD des documents pour les indexer, mais souvent des fonctionnalités manquent aux index pour répondre à des conditions de structure précises ou encore à des conditions de contenus relatives à des éléments de structure [124, 68, 93, 126].

Pour pallier les inconvénients présentés ci-dessus, nous proposons un modèle de représentation des données générique et orienté RI. Ce modèle nous permettra d'implémenter plusieurs modèles de recherche et d'indexer et de traiter des collections de documents hétérogènes, c'est à dire possédant des DTDs différentes. Notre modélisation, basée sur l'approche XPath Accelerator [93], nous permet en outre de conserver toute l'information structurelle des documents et de naviguer aisément dans leur représentation en arbre.

2. Les langages proposées dans la littérature pour l'interrogation des corpus de documents structurés offrent pour la plupart de puissantes fonctionnalités [125, 40, 91, 66]. Ils requièrent cependant de la part de l'utilisateur une connaissance poussée de la structure des documents qu'il interroge, ainsi que la spécification de l'élément qu'il désire voir retourner par le système. La majorité des langages est aussi basée sur une approche orientée base de données, et le contenu des documents est alors traité de façon booléenne. Il a pourtant été démontré en RI que la prise en compte du poids des mots-clés dans un document est primordiale, voire nécessaire.

   Nous proposons dans nos travaux un langage permettant à l'utilisateur d'exprimer son besoin selon divers degrés de précision. Ce langage possède une syntaxe simple, ne reposant pas sur SQL. L'utilisateur peut formuler son besoin à base de simples mots-clés, sans précision aucune sur la structure, et laisser le modèle décider de la granularité appropriée de l'information à renvoyer. Il peut aussi s'il le souhaite formuler des contraintes sur la structure des documents, en introduisant éventuellement la notion de hiérarchie entre les différentes conditions de structure.

3. Les modèles de recherche proposés dans la littérature cherchent à adapter les modèles utilisés en RI traditionnelle ([79, 12, 53, 134, 137, 201, 35, 219, 103] pour le modèle vectoriel, [122] pour le modèle booléen, [120, 78, 127, 4, 192, 147, 104, 156] pour le modèle probabiliste), et ce afin de retrouver les unités d'information les plus pertinentes à une requête utilisateur. Cependant, contrairement à la RI traditionnelle, la pertinence dans le cadre de la RI structurée est exprimée selon deux dimensions : l'*exhaustivité* et la *spécificité*. Les modèles de recherche devraient donc

prendre en compte ces deux dimensions de manière explicite, ce qui n'est pas forcément le cas des approches proposées dans la littérature. Les expérimentations que nous avons menées dans [185] montrent cependant que la recherche des éléments pertinents peut difficilement s'effectuer en deux phases séparées (une phase pour rechercher l'information exhaustive et une autre pour rechercher l'information spécifique à l'intérieur de cette information exhaustive), et que l'information structurelle présente dans les documents doit être utilisée au mieux, même pour les requêtes composées de simples mots-clés. Enfin, la plupart des approches présentées dans la littérature traitent les conditions de structure en effectuant un filtre des résultats sur ces dernières. Les conditions structurelles présentes dans les requêtes doivent pourtant pouvoir être traitées de manière vague, afin de proposer à l'utilisateur qui ne connaît pas nécessairement parfaitement la structure des documents qu'il interroge des solutions alternatives à son besoin.

Pour répondre à ces différents besoins, nous proposons un modèle de propagation de la pertinence permettant de retrouver les unités d'information les plus exhaustives et spécifiques à une requête. Lorsque la recherche porte sur des requêtes à base de simples mots-clés, notre modèle décide de la granularité appropriée de l'information à renvoyer à l'utilisateur, en introduisant la notion d'informativité dans le calcul de la pertinence des éléments. Dans le cas de requêtes possédant des conditions de structures, nous intégrons la structure dans notre modèle de pertinence. Diverses fonctions de propagation sont utilisées afin d'effectuer une correspondance vague entre l'arbre de la requête et l'arbre du document, c'est à dire afin de traiter des structures qui ne sont pas forcéments identiques.

3.3 Présentation générale du modèle XFIRM

Le modèle XFIRM (*XML Flexible Information Retrieval Model*) que nous proposons pour la recherche d'information dans des documents semi-structurés est un modèle de RI orienté pertinence, basé sur une méthode de propagation de la pertinence. Il repose sur un modèle de représentation des documents nous permettant de conserver à la fois toute l'information structurelle et toute l'information textuelle des documents. Nous considérons qu'un document semi-structuré est un *arbre*, composé de *noeuds internes*, de *noeuds feuilles* et d'*attributs*. La structure arborescente des documents est modélisée grâce aux *arcs* reliants ces composants, chaque noeud interne pouvant posséder plusieurs enfants. L'information textuelle des documents est quant à elle conservée au sein des noeuds feuilles.
Le langage d'interrogation que nous proposons permet à l'utilisateur d'expri-

mer son besoin selon divers degrés de précision. Ce dernier peut par exemple formuler des requêtes à base de simples mots-clés (qui peuvent former des expressions et qui peuvent être reliés par des opérateurs booléens). Ce type de requête peut par exemple être utilisé quand l'utilisateur n'a pas la moindre idée de l'unité d'information qu'il désire voir retourner. Il peut aussi s'il le souhaite formuler des contraintes sur la structure des documents. Ces conditions de structure peuvent lui permettre d'indiquer le type des unités d'informations qu'il désire voir retournées par le système (on parlera d'*élément cible*). Il peut enfin inclure la notion de hiérarchie entre les différentes conditions de structure. Lorsque la requête contient des conditions de structure, elle peut, comme les documents, être assimilée à un arbre.
Le modèle de recherche repose sur un modèle de propagation de la pertinence. Un premier score de pertinence est calculé pour les noeuds feuilles des documents, et ce score est ensuite propagé dans l'arbre du document. Afin de répondre au critère de spécificité des unités d'informations, ce score est diminué durant la propagation.
Pour les requêtes composées de simples mots-clés, afin de déterminer la granularité appropriée de l'information à renvoyer, nous utilisons la taille des noeuds comme une indication de leur importance durant la propagation et nous situons enfin chaque noeud dans son contexte en prenant en compte la pertinence du document dans son entier.
Les requêtes composées de conditions de structure sont décomposées en requêtes élémentaires de type *nom_element[condition contenu]* et chacune de ces requêtes est traitée de manière indépendante : on évalue la similarité des noeuds feuilles à la condition de contenu et une première propagation est effectuée pour répondre à la contrainte de structure. Les éventuelles conditions de hiéarchie de la requête initiale sont ensuite traitées en effectuant des propagations de la pertinence des noeuds résultats des requêtes élémentaires vers les unités d'information faisant partie de l'ensemble des éléments cibles. Si ces éléments cibles ne sont pas précisés par l'utilisateur, ils sont identifiés automatiquement. Les différentes fonctions de propagation permettent d'effectuer une correspondance stricte ou vague entre l'arbre de la requête et l'arbre des documents. Lorsqu'une correspondance vague entre l'arbre de la requête et l'arbre du document est effectuée, des documents possédant une structure différente de celle la requête peuvent être renvoyés à l'utilisateur, même si leur pertinence est plus faible que celle des documents pour lesquels toutes les conditions de structure sont respectées. Par exemple, un document possédant la structure /a/b/c sera pertinent pour une requête /a/d/c, mais aussi pour une requête /a/c/b.

3.4 Modèle de représentation des documents

Notre modèle de propagation de la pertinence se base sur la structure arborescente des documents XML, en propageant les scores de pertinence des noeuds à travers l'arbre des documents. Nous présentons ici le modèle de représentation des documents que nous utilisons pour modéliser les documents XML. Ce modèle nous permet de naviguer dans la structure en arbre des documents XML et de représenter le contenu de cette structure.

3.4.1 Modèle de représentation

Un document structuré ds est un *arbre*, défini par les ensembles N, F, A et L.

$$\textbf{Document Structuré} : ds = (N,\ F,\ A,\ L)$$

avec $N = \{n_1, n_2, ...\}$ l'ensemble des noeuds internes, $F = \{nf_1, nf_2, ...\}$ l'ensemble des noeuds feuilles, $A = \{a_1, a_2, ...\}$ l'ensemble des attributs et L est un ensemble d'*arcs orientés*. Notons que cette représentation est une simplification du modèle de données de Xpath et Xquery présenté dans [66], dans lequel un noeud peut être un *document*, un *élément*, un *attribut*, du *texte*, un *espace de noms*, une *instruction* ou alors un *commentaire*.
Un arc orienté est une paire (u, v) formée de deux éléments des ensembles N, F ou A tels que :

- u est parent de v
- chaque $n_i \in N$ appartient au moins une fois à L en tant que premier composant d'une paire formant un arc
- chaque $n_i \in N$, $nf_i \in F$, $a_i \in A$ excepté le noeud racine appartient une et une seule fois à L en tant que second composant d'une paire formant un arc

Les noeuds sont ainsi reliés entre eux par des arcs qui forment les relations parent/enfant. Tous les noeuds excepté le noeud racine ont exactement un noeud parent.

Dans l'exempe de la figure 3.1, on a $N = \{n_1, n_2, ...n_{17}\}$, $F = \{nf_1, nf_2, ..., nf_{11}\}$, $A = \{a_1\}$, et $L = \{(n_1, n_2), (n_1, n_8), (n_1, a_1), ..., (n_3, nf_1), ...\}$.
Tout $n_i \in N$ est le point de départ d'un *sous-arbre* de l'arbre ds.
Un *chemin* dans l'arbre est une suite ordonnée de noeuds n_i. Pour arriver à chaque nf_i, il n'y a qu'un seul chemin possible à partir de la racine.

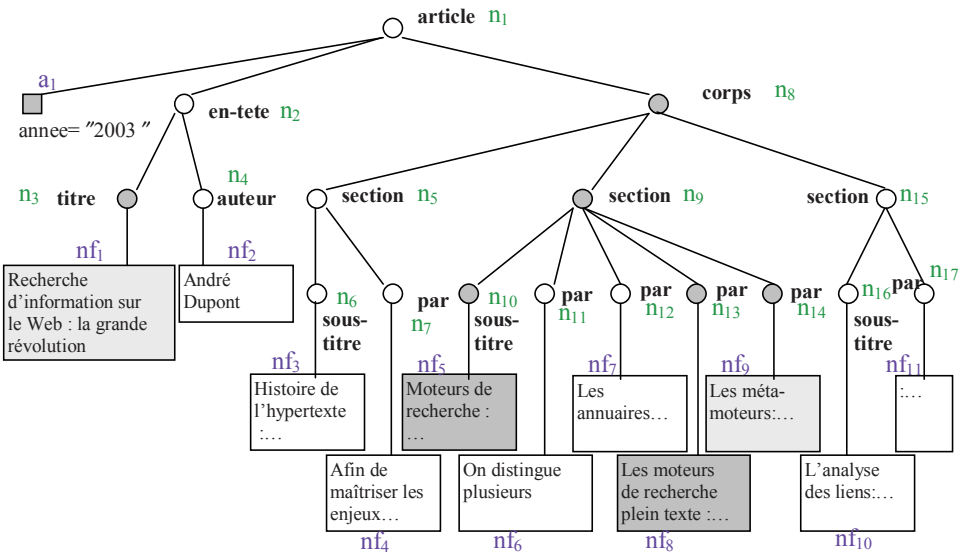


FIG. 3.1 – *Représentation du document article.xml*

Cette représentation générique de la structure des documents nous permet de gérer des collections de documents hétérogènes (possédant des DTDs différentes). La représentation physique des documents que nous utilisons permet de d'optimiser la navigation dans la structure arborescente des documents. On trouvera une présentation de cette représentation dans la section 3.8 présentant le prototype que nous avons développé.
Comme le montre la figure 3.1, l'information textuelle des documents est située au niveau des noeuds feuilles.
Un noeud feuille nf_i est composé de termes t_j et de leur poids w_j^i dans le noeud feuille.

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

Les termes t_i sont extraits de la partie textuelle des documents en utilisant des techniques traditionnelles couramment utilisées en RI.

3.4.2 Pondération

Le calcul du poids des termes au sein des noeuds feuilles n'est pas un problème trivial. Ce poids doit modéliser l'importance du terme dans le noeud feuille, mais aussi au sein du document et de la collection. Le calcul de w_j^i dépend du modèle de pondération considéré. Ce calcul peut être fonction de :

– tf_j^i la fréquence du terme t_j dans le noeud feuille nf_i

– idf_j la fréquence inverse de document pour le terme t_j, définie par :

$$idf_j = log(\frac{|D|}{|d_j|}) \tag{3.1}$$

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

– ief_j la fréquence inverse d'élément pour le terme t_j, qui est une adaptation de la formule idf_j a la granularité de l'information que nous traitons (on évalue le poids d'un terme dans un noeud feuille et non plus dans un document). ief_j est défini de la façon suivante :

$$ief_j = log(\frac{|F_c|}{|nf_j|}) \tag{3.2}$$

où $|F_c|$ est le nombre total de noeuds feuilles de la collection et $|nf_j|$ est le nombre de noeuds feuilles de la collection contenant le terme t_j

– l_i la taille du noeud feuille nf_i (c'est à dire le nombre de termes qu'il contient)

– Δl la taille moyenne des noeuds feuilles de la collection

Nous avons testé plusieurs fonctions de pondération pour le calcul de w_j^i, présentées dans le chapitre 4. On trouvera la fonction de calcul de poids optimale dans les paragraphes 3.6.1 et 3.7.2.

3.5 Langage de requêtes

Les caractéristiques du langage de requêtes que nous proposons sont les suivantes [181] :

– *Syntaxe simple*, ne reposant pas sur SQL ; notre langage peut être vu comme une simplification du langage XPath ;

– Formulation de requêtes à base de *simples mots-clés*, sans précision aucune sur la structure : ce type de requête pourra par exemple être utilisé lorsque l'utilisateur n'a pas la moindre idée de l'unité d'information qu'il désire voir retournée ;

– Possibilité de formuler des *contraintes sur la structure des documents*, sans nécessairement donner le type de l'unité d'information à retourner (contrairement à des langages comme XQuery [66]) ;

– Possibilité de formuler des requêtes plus complexes, en introduisant la notion de *hiérarchie* entre les différentes contraintes de structure, mais sans pour autant devoir donner des chemins absolus : le langage permet l'expression de chemins vagues.

– Possibilité d'étendre les requêtes grâce à un *dictionnaire des noms de balises* des différents noeuds rencontrés dans le corpus. Ceci sert particulièrement dans le cas de corpus composés de documents suivant des

DTDs différentes ou dans le cas de requêtes pour lesquelles l'utilisateur ne connaît pas exactement le nom des éléments qu'il recherche [201].

3.5.1 Le langage de requêtes XFIRM par l'exemple

Le langage de requête XFIRM propose à l'utilisateur de formuler son besoin selon quatre degrés de précision.
S'il recherche simplement de l'information et que le type de l'unité d'information renvoyée lui importe peu pourvu qu'elle réponde à son besoin, il pourra formuler sa requête avec de *simples mots-clés* (***degré de précision P1***). Ces mots-clés pourront éventuellement être reliés par des opérateurs (opérateurs booléens ET, OU, NON et opérateurs d'importance, '+' signifiant que le terme est impératif et '-' signifiant que le terme n'est au contraire pas souhaité). La recherche sur des expressions est aussi possible, en encadrant les expressions de " ". Ce type de requête constitue une forme de recherche habituelle dans les moteurs de recherche "traditionnels". On trouvera ci-dessous quelques exemples de requêtes de type P1, aussi appelées *requêtes orientées contenu*.
P1.1 : internet google
P1.2 : +internet - "moteur de recherche"
P1.3 : internet OU (toile ET réseau)

Si l'utilisateur désire donner des *conditions sur la structure* des documents, il peut exprimer son besoin en donnant le nom d'un élément, et éventuellement préciser son besoin sur cet élément en ajoutant des conditions sur son contenu ou la valeur de ses attributs. Ces requêtes de ***précision P2*** peuvent être combinées entre elles par des opérateurs booléens. Par exemple, les requêtes :
P2.1 : section[]
P2.2 : section[internet recherche]
P2.3 : titre["moteurs de recherche"] ET section[@num=1]
signifient que l'utilisateur souhaite obtenir un élément de type *section* (dans le cas de P2.1), un élément *section* parlant de *internet* et de *recherche* (dans P2.2), ou une unité d'information contenant à la fois un élément *titre* sur "moteurs de recherche" et un élément *section* ayant un attribut *num* de valeur 1 (P2.3).
L'élément retourné à l'utilisateur est donc l'élément spécifié dans la requête si la requête est composée d'une seule opérande (P2.1 ou P2.2) ou alors une unité d'information répondant à toutes les conditions s'il s'agit d'une requête contenant des opérateurs booléens (P2.3).

Les requêtes de ***précision P3*** permettent d'ajouter la notion de *hiérarchie* entre les différentes conditions de structures (requêtes de type P2), qui sont

alors séparées par le signe "//". Par exemple, les requêtes :
P3.1 : //article[] // titre["moteurs de recherche"] ET section[internet google]
P3.2 : //article[@date-publi=2000] // corps[internet]// section[@num=1]
signifient que l'utilisateur souhaite obtenir respectivement un noeud *article* ayant pour descendant un élément *titre* contenant les termes "*moteurs de recherche*" et un élément *section* parlant de *internet* et de *google* (P3.1), un noeud *article* dont l'attribut *date-publi* vaut 2000 ayant pour descendant un élément *corps* contenant le mot *internet* et étant lui-même ancêtre d'un noeud *section* ayant un attribut *num* de valeur 1 (P3.2).

Dans les requêtes de type P3, les noeuds retournés à l'utilisateur sont par défaut ceux spécifiés dans la première requête de type P2 (*article* dans les exemples P3.1 et P3.2). Si l'utilisateur a une idée plus précise de ce qu'il recherche, il pourra spécifier l'unité d'information qu'il désire voir retournée. Dans la suite, nous nommerons cette unité d'information *élément cible*. Cet élément cible est spécifié grâce au signe "ec :" précédant une requête de type P2. Ainsi la requête de ***précision P4*** :
P4.1 : //article[@date-publi=2000]// ec : corps[] // par[google] ET sous-titre ["moteurs de recherche"]
signifie que l'utilisateur souhaite obtenir un noeud *corps* ayant pour ancêtre un noeud *article* dont l'attribut *date-publi* vaut 2000 et pour descendant un noeud *par* parlant de *google* et un noeud *sous-titre* contenant l'expression "moteurs de recherche".

Les requêtes de type P2, P3 ou P4 sont aussi appelées *requêtes orientées contenu et structure*. La syntaxe de ces requêtes permet à l'utilisateur de formuler des *expressions de chemin vagues* dans l'expression de ses conditions. Il peut par exemple exprimer la requête *article//section* (il sait alors qu'un noeud *article* a pour descendant un noeud section), sans indiquer nécessairement le chemin d'accès précis (*article/corps /section*).
Un dictionnaire des balises est utilisé par défaut dans le traitement des requêtes. Il est utile dans le cas où l'utilisateur fait des recherches dans un corpus contenant des documents suivant des DTD différentes ou des documents ayant des balises pouvant être considérées comme équivalentes. Par exemple, dans la requête P4.1, la balise *titre* pourra être remplacée par la balise *sous-titre*, car elles sont considérées comme équivalentes dans le dictionnaire. On trouvera un exemple de traitement des requêtes avec ce dictionnaire dans la section 3.6.

Le langage de requêtes XFIRM peut ainsi être vu comme une extension de XPath à la recherche textuelle. Contrairement aux langages basés sur une syntaxe FLWR (For Let Where Return) [40, 66] le langage n'oblige pas l'utilisateur à spécifier le type de l'unité d'information qu'il désire voir retournée.

La granularité de l'information à renvoyer est alors décidée par le système. De plus, pour faciliter la recherche dans des structures mal connues, le langage permet d'exprimer des chemins indéterminés ou partiellement connus et permet de combiner de façon booléenne les conditions de structure.

3.5.2 Grammaire du langage de requête

La syntaxe du langage XFIRM est décrite dans la grammaire du tableau 3.1.

requête : := <P1> | <P2> | <P3> | <P4>

P1 : := <expressionRéduite> <suiteExpressions>
expressionRéduite : := <termes> <suiteExpressionRéduite>
| "(" <termes> ")" <suiteExpressionRéduite>
suiteExpressionRéduite : := <expressionRéduite> | vide
suiteExpressions : := <opérateurBooléen> <P1> | vide
termes : :=<opérateurAdditif> <motsClés>
motsClés : := terme<suiteTermes> | "" " terme<suiteTermes>" " "<suiteTermes>
suiteTermes : :=vide | <termes>
opérateurAdditif : := " + " | " - " | vide
opérateurBooléen : := " OU " | " ET " | " NON " | vide

P2 : := <expressionStructure><suiteExpressionStructure>
expressionStructure : :=nomElement "[" <condition> "]"
condition : := "@" nomAttribut "=" terme | P1 | vide
suiteExpressionStructure : := <operateurBooléen> <expressionStructure> | vide

P3 : := "//" <P2><suiteP3>
suiteP3 : :=<P3> | vide

P4 : := <suiteP3><elementCible><suiteP3>
ElementCible : := "// ec : " <P2>

Légende : vide : expression terminale représentant l'ensemble vide
terme : expression terminale représentant un mot clé
nomElément : expression terminale représentant un nom de balise
nomAttribut : expression terminale représentant un nom d'attribut
ec : expression terminale indiquant la présence d'un élément cible

Tab. 3.1 – Grammaire BNF du langage de requête XFIRM

3.6 Evaluation des requêtes orientées contenu

Le but du traitement des requêtes orientées contenu (c'est à dire des requêtes de type P1) est de retrouver des sous-arbres de taille minimale répondant de manière exhaustive à la requête. Ce traitement est effectué comme présenté ci-dessous :

1. une première étape consiste à évaluer la similarité des noeuds feuilles de l'index à la requête (on parle alors de calcul du score des noeuds feuilles),
2. et une seconde étape consiste à rechercher les sous-arbres pertinents et informatifs. Pour ce faire, la dimension d'informativité des sous-arbres est évaluée :
   - en propageant vers le haut le score des feuilles dans l'arbre du document, et ce en privilégiant les noeuds les plus porteurs d'informations,
   - et en propageant vers le bas le score du document dans sa globalité, afin de tenir compte du contexte du sous-arbre dans l'évaluation de sa pertinence

3.6.1 Calcul du score des noeuds feuilles

Les requêtes de type P1 sont composées de termes et d'expressions, éventuellement reliés par des opérateurs booléens. Quel que soit le contenu de ces requêtes, nous nous ramenons à la représentation suivante :

$$q = \{(t_1, w_1^q), \ldots, (t_T, w_T^q)\} \tag{3.3}$$

où t_i est soit un terme unique soit une expression, et w_i^q est le poids de t_i dans la requête q.
Les scores des noeuds feuilles identifiés dans l'arbre du document sont calculés grâce à la fonction de similarité $RSV_m(q, nf)$ (*Retrieval Status Value*), où m est le modèle de RI considéré (équation 3.4).
Si la requête est composé de termes et des poids associés, on a :

$$RSV_m(q, nf) = \sum_{i=1}^{T} w_i^q * w_i^{nf} \tag{3.4}$$

où w_i^q et w_i^{nf} sont respectivement le poids du terme i dans la requête q et le noeud feuille nf, le calcul de ces poids dépendant du modèle m de pondération considéré.
Nous considérons ainsi que les termes sont implicitement reliés par le booléen OU. Les autres conditions booléennes (ET et NON) sont traitées en amont grâce à un filtrage sur les ensembles résultats associés à chaque terme. Le préfixe '+' des requêtes est enfin utilisé comme le booléen ET, alors que le préfixe '-' est

utilisé comme le booléen NON. Un terme précédé de '+' doit obligatoirement être présent dans le résultat renvoyé à l'utilisateur, alors qu'un terme précédé de '-' ne doit au contraire pas être présent.

Plusieurs fonctions de calcul du poids des termes dans les noeuds feuilles et la requête ont été évaluées et sont présentées dans le chapitre 4, section 4.4.1 et dans [183]. La fonction nous permettant d'obtenir des performances optimales est une adaptation de la formule *tf-idf* à la granularité de l'information que nous traitons (on ne parle plus de documents mais de noeuds feuilles). Les poids des termes dans la requête et les noeuds feuilles sont alors les suivants :

$$w_i^q = tf_i^q * ief_i \tag{3.5}$$
$$w_i^{nf} = tf_i^{nf} * ief_i \tag{3.6}$$

où tf_i^q et tf_i^{nf} sont respectivement la fréquence du terme i dans la requête q et dans le noeud feuille nf, et ief_i défini dans l'équation 3.2.

3.6.2 Propagation de la pertinence des noeuds feuilles

Une valeur de pertinence est ensuite calculée pour chaque noeud de l'arbre de document, en utilisant les poids des noeuds feuilles qu'il contient. Les termes apparaissant près de la racine d'un sous-arbre paraissent plus porteurs d'information pour le noeud associé que ceux situés plus bas dans le sous-arbre. Il semble ainsi intuitif que plus grande est la distance entre un noeud et son ancêtre, moins il contribue à sa pertinence. Nous modélisons cette intuition par l'utilisation dans la fonction de propagation du paramètre $dist(n, nf_k)$, qui représente la distance entre le noeud n et un de ses noeuds feuille nf_k dans l'arbre du document, c'est à dire le nombre d'arcs séparant les 2 noeuds. Il parait aussi intuitif que plus un noeud possède de noeuds feuilles pertinents, plus il est pertinent. Nous introduisons alors dans la formule de propagation le paramètre $|F_n^p|$, qui est le nombre de noeuds feuilles descendants de n ayant un score non nul. La valeur de pertinence p_n d'un noeud est alors calculée selon la formule 3.7 :

$$p_n = |F_n^p|. \sum_{nf_k \in F_n} \alpha^{dist(n,nf_k)-1} * (RSV_m(q, nf_k)) \tag{3.7}$$

où F_n est l'ensemble des noeuds feuilles nf_k descendants de n, et $\alpha \in]0..1]$ est un paramètre permettant de quantifier l'importance de la distance séparant les noeuds dans la formule de propagation.
Les noeuds sont ensuite renvoyés à l'utilisateur par ordre décroissant de pertinence à la requête.

Illustrons cette propagation avec le document de la figure 3.2 et la requête '*moteurs de recherche*' composée de trois termes implicitement reliés par le booléen OU. Sur cette figure, et pour plus de simplicité dans la suite des exemples, les noeuds feuilles sont numérotés de nf_1 à nf_{11} et les noeuds internes de n_1 à n_{17}, selon leur ordre d'apparition dans la lecture séquentielle du document.

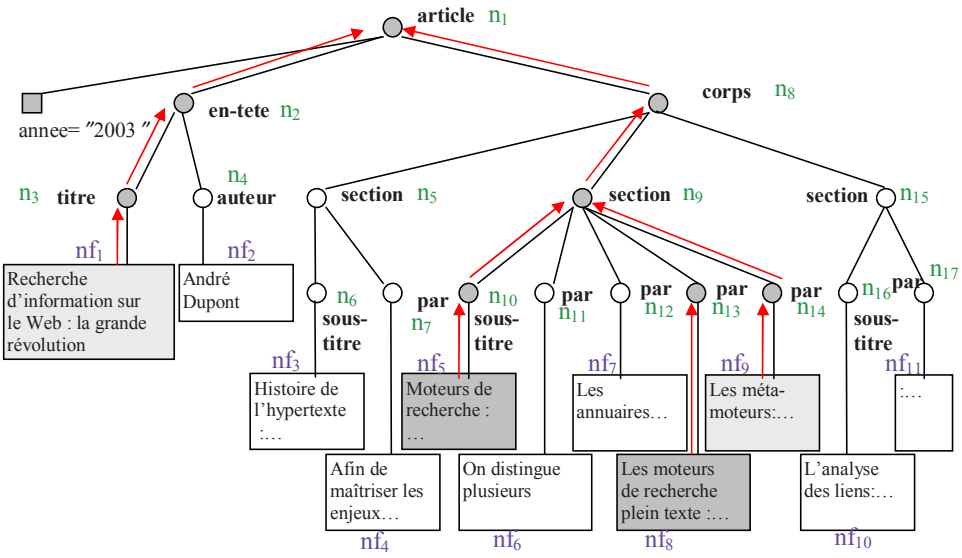


FIG. 3.2 – Exemple de propagation de la pertinence dans un arbre XML

Les noeuds feuilles nf_1, nf_5, nf_8 et nf_9 ont un score de similarité non nul avec la requête. La pertinence de leurs noeuds parents respectifs est égale à $1.\alpha^0 * RSV(q, nf_k) = RSV(q, nf_k)$, c'est à dire à leur propre score. De la même façon, tous les noeuds parents (et non ancêtres) de noeuds feuilles auront une pertinence égale au score de similarité de leur noeud feuille descendant (ou à la somme des scores de leurs noeuds feuilles descendants dans le cas de noeuds possédant des contenus mixtes).
Nous avons ensuite :
$p_{n_2} = 1 * [\alpha^1 * RSV(q, nf_1) + \alpha^1 * RSV(q, nf_2)] = \alpha * RSV(q, nf_1)$
$p_{n_9} = 3 * [\alpha^1 * RSV(q, nf_5) + \alpha^1 * RSV(q, nf_6) + \alpha^1 * RSV(q, nf_7) + \alpha^1 * RSV(q, nf_8) + \alpha^1 * RSV(q, nf_9)]] = 3\alpha(RSV(q, nf_5) + RSV(q, nf_8) + RSV(q, nf_9))$
$p_{n_8} = 3 * [\alpha^2 * RSV(q, nf_5) + \alpha^2 * RSV(q, nf_8) + \alpha^2 * RSV(q, nf_9)]$
$= 3\alpha^2(RSV(q, nf_5) + RSV(q, nf_8) + RSV(q, nf_9))$
$p_{n_1} = 4 * [\alpha^2 * RSV(q, nf_1) + \alpha^3(RSV(q, nf_5) + RSV(q, nf_8) + RSV(q, nf_9))]$

Plusieurs valeurs de α ont été testées dans les expérimentations présentées dans le chapitre 4, section 4.4.2. La valeur $\alpha = 0.6$ semble être la valeur permettant d'obtenir le meilleur compromis entre exhaustivité et spécificité.

3.6.3 Ajout de la dimension d'informativité au calcul de la pertinence

Le modèle de pertinence que nous venons de définir considèrerait un noeud contenant les seuls termes de la requêtes comme pertinent, car très exhaustif (il contient les informations requises dans la requête) et très spécifique (tout son contenu concerne la requête). Cependant, un tel noeud, même s'il est considéré comme réponse idéale, n'est pas informatif (c'est à dire qu'il n'apporte pas d'information à l'utilisateur). Nous nous proposons donc d'ajouter la dimension d'*informativité* d'un noeud dans le calcul de sa pertinence.
La définition de la dimension d'informativité que nous proposons repose sur deux intuitions :

- la *longueur* du noeud (c'est à dire le nombre de termes qu'il contient) peut être un paramètre déterminant dans le calcul de son informativité, mais tout le problème est de savoir comment et où introduire ce paramètre. Comme le montrent les expérimentations présentées au chapitre 4, l'utilisation de la longueur des éléments au niveau du calcul du score des noeuds feuilles ne semble pas être utile ;
- le *contexte* du noeud (c'est à dire ses noeuds ancêtres et plus particulièrement son noeud racine) permet de mieux situer son contenu par rapport à la requête, et donc de mieux déterminer s'il est porteur d'information.

Pour répondre à la première intuition, nous avons évalué l'introduction du paramètre longueur des noeuds après la propagation, en introduisant un seuil sur la longueur des éléments ou en comparant leur taille avec la taille médiane et moyenne des noeuds pertinents. Les expérimentations présentées dans le chapitre 4, section 4.4.3 et dans [183] montrent qu'il est préférable d'utiliser cette longueur *durant* la propagation, comme une indication sur l'importance de l'information que véhiculent les noeuds feuilles. Nous parlerons alors de *propagation pondérée par la taille des noeuds feuilles*, et nous décrivons ce processus dans la section ci-dessous.
Concernant la seconde intuition, nous avons d'abord introduit le contexte des noeuds en les triant en fonction de la pertinence des documents associés. De meilleurs résultats sont obtenus en introduisant le concept de *pertinence contextuelle*, qui consiste à prendre en compte le poids global du document dans le calcul de la pertinence d'un noeud, et donc de tenir compte du contexte des noeuds pour évaluer leur informativité. Notre proposition est présentée dans la section 3.6.3.2.

3.6.3.1 Propagation pondérée par la taille des noeuds feuilles

De manière intuitive, on peut penser que le concepteur d'un document utilise les noeuds de petite taille pour faire ressortir des informations importantes. Ces noeuds peuvent ainsi donner des indications précieuses sur la pertinence de leurs noeuds ancêtres. Un noeud *titre* dans une *section* par exemple permet de situer avec précision le sujet de son noeud ancêtre *section*. Pour répondre à cette intuition, nous proposons d'augmenter l'importance des noeuds de petite taille durant la propagation.
Soit l_k la taille du noeud feuille nf_k et Δl la taille moyenne d'un noeud feuille.

- Si un noeud feuille nf_k est de petite taille (c'est à dire de taille inférieure à la moyenne) la pertinence p_{par} de son noeud parent *par* doit être faible.
- Mais il doit jouer un rôle plus important que les autres noeuds feuilles dans le calcul de la pertinence de ses noeuds ancêtres *anc*.

De manière synthétique, nous introduisons dans le calcul de la pertinence p_n d'un noeud n (définie dans l'équation 3.7) le paramètre $\beta(nf_k)$:

$$p_n = |F_n^p|. \sum_{nf_k \in F_n} \alpha^{dist(anc,nf_k)-1} * \beta(nf_k) * RSV(q, nf_k) \tag{3.8}$$

avec F_n l'ensemble des noeuds feuilles nf_k descendants de n, et $|F_n^p|$ le nombre de noeuds feuilles descendants de n ayant un score non nul.
Nous utilisons pour β la formule suivante :

$$\beta(nf_k) = \begin{cases} l_k/\Delta l \ si \ dist(n, nf_k) = 1 \ et \ l_k < \Delta l \\ log(\Delta l/l_k) \ si \ dist(n, nf_k) > 1 \ et \ l_k < \Delta l \\ 1 \ sinon \end{cases} \tag{3.9}$$

Les expérimentations concernant l'évaluation de β sont présentées dans le chapitre 4, section 4.4.3.3 et dans [183].

3.6.3.2 Pertinence contextuelle

Dans le paragraphe précédent, la dimension d'informativité que nous avons introduite prend en compte l'importance variable des noeuds feuilles dans la propagation. Le contexte des noeuds (c'est à dire leurs noeuds ancêtres) peut aussi jouer un rôle prépondérant dans le calcul de cette informativité. En effet, le concepteur d'un document suit une certaine unité dans ses idées, même si le contenu du document est hétérogène. La pertinence des unités d'informations du document est alors liée à la pertinence de cette unité de pensée à la requête. De même, un noeud appartenant à un document fortement pertinent doit être mieux classé qu'un noeud se trouvant dans un document de pertinence moindre. Dans le cadre de notre modèle, on parlera de *pertinence contextuelle* d'un noeud. Cette pertinence contextuelle est calculée grâce à une

rétropropagation de la pertinence du noeud racine (c'est à dire du document) vers les noeuds internes.

La pertinence p_n d'un noeud n est alors définie de la façon suivante :

$$\begin{aligned} p_n &= \rho * |F_n^p|. \sum_{nf_k \in F_n} \alpha^{dist(n,nf_k)-1} * \beta(nf_k) * RSV(q, nf_k) \\ &\quad +(1-\rho) * |F^p|. \sum_{nf_k \in F} \alpha^{dist(racine,nf_k)-1} * \beta(nf_k) * RSV(q, nf_k) \\ &= \rho * |F_n^p|. \sum_{nf_k \in F_n} \alpha^{dist(n,nf_k)-1} * \beta(nf_k) * RSV(q, nf_k) \\ &\quad +(1-\rho) * p_{racine} \end{aligned} \tag{3.10}$$

avec F_n et F respectivement l'ensemble des noeuds feuilles nf_k descendants de n et l'ensemble des noeuds feuilles nf_k du document, $|F_n^p|$ et $|F^p|$ respectivement le nombre de noeuds feuilles descendant de n ou du document et ayant un score non nul, $RSV(q, nf_k)$ calculé d'après 3.6, $\beta(nf_k)$ calculé d'après 3.9 et $\rho \in [0..1]$ est un paramètre servant de pivot et permettant d'ajuster l'importance de la pertinence du noeud racine lors de la rétropropagation.
Les expérimentations présentées dans le chapitre 4 montrent que le contexte du noeud est important pour le calcul de son informativité, mais ne doit cependant pas avoir une place prépondérante par rapport à la propagation "simple". $\rho = 0.9$ est ainsi la valeur donnant les meilleures précisions moyennes sur les tests que nous avons menés.

Le modèle que nous proposons pour le traitement des requêtes ne contenant que des conditions de contenu repose donc sur dimension d'informativité de la pertinence d'un noeud, calculée par propagation de la pertinence de ses noeuds descendants (et cette propagation est fonction de la taille des noeuds) et par rétropropagation de son contexte.

3.7 Evaluation des requêtes orientées contenu et structure

Les requêtes contenant des conditions de contenu et de structure sont de type P2, P3, ou P4. Les traitements de ces trois types de requêtes sont étroitement liés, comme nous l'exposons dans la section suivante. Le traitement d'une requête de type P3 ou P4 consiste au traitement des requêtes de type P2 qui la compose et à la recomposition de l'arbre initial de la requête à partir de ces résultats.

3.7.1 Decomposition de la requête

Les requêtes les plus précises (de type $P4$ ou $P3$) sont construites à partir des requêtes de type $P2$. Ainsi, les requêtes de type P_3 ou P_4 se décomposent comme indiqué dans les formules 3.11 et 3.12.

$$P3 = //P2_1//P2_2// \ldots //P2_n \tag{3.11}$$

$$P4 = //P2_1//P2_2// \ldots //ec : P2_i// \ldots //P2_n \tag{3.12}$$

Les requêtes de type $P3$ et $P4$ peuvent être assimilées à des arbres (puisqu'elles contiennent la notion de hiérarchie). On parlera alors d'*arbre de la requête*.
Illustrons leur décomposition avec la requête suivante :
//article[@annee=2003]// ec : section[]// par[annuaire] ET titre[moteurs de recherche]
Cette requête se décompose en requêtes de type $P2$ de la façon suivante :
$P2_1 = article[@annee = 2003]$
$P2_2 = section[]$
$P2_3 = par[annuaire]\ ET\ titre[moteurs\ de\ recherche]$

Une requête $P2_i$ de type $P2$ peut ensuite être décomposée en sous-requêtes élémentaires $SRE_{i,j}$ reliées entre elles par des opérateurs booléens et de la forme :

$$SRE_{i,j} = \begin{cases} b\_n[q] \\ b\_n[] \\ b\_n[n\_a = v] \end{cases} \tag{3.13}$$

où :

- b_n est le nom de balise du noeud n,
- $q = \{(t_1, w_1^q), (t_2, w_2^q), \ldots (t_T, w_T^q)\}$ est un ensemble de mots-clés et leur poids dans la requête, c'est à dire une requête de type $P1$
- n_a est le nom d'attribut de l'attribut a avec a estAttribut de n
- v est la valeur désirée de a

Nous avons alors par exemple : $P2_3 = (SRE_{3,1} = par[annuaire])\ ET\ (SRE_{3,2} = titre[moteurs\ de\ recherche])$, où *annuaire* et *moteurs de recherche* sont des requêtes de type P1.

Dans ce qui suit, nous décrivons le processus nécessaire au traitement d'une requête de type P3 ou P4. Après découpage des requêtes en sous-requêtes élémentaires, le traitement est effectué comme suit :

1. traitement des sous-requêtes élémentaires
2. traitement des requêtes de types P2 à partir des résultats des sous-requêtes élémentaires

3. traitement des conditions de hiérarchie de la requête à partir des résultats des requêtes de type P2.

3.7.2 Traitement des sous-requêtes élémentaires $SRE_{i,j}$

Comme nous venons de le voir, le traitement de la structure dans les requêtes passe d'abord par le traitement des sous-requêtes élémentaires $SRE_{i,j}$ formant les requêtes de type $P2$.
L'ensemble de paires *(noeud, pertinence)* $R_{i,j}$ résultat d'une $SRE_{i,j}$, (définie dans 3.13) est calculé de la façon suivante :
(1) Si $SRE_{i,j} = b\_n[q]$, (c'est le cas par exemple de $SRE_{3,1}$ dans notre exemple)

$$R_{i,j} = \{(n, p_n)/n \in construct(b\_n) \ et \ p_n = F_k(RSV_m(q, nf_k), dist(n, nf_k))\} \tag{3.14}$$

où :
- p_n est le score de pertinence du noeud n
- la fonction $construct(b\_n)$ permet de créer l'ensemble de tous les noeuds ayant pour nom de balise b_n ou ayant un nom de balise considéré comme équivalent (d'après le dictionnaire des noms de balises créé au moment de l'indexation) et possédant au moins un noeud feuille descendant dont le score de similarité à la requête q est non nul,
- la fonction $F_k(RSV_m(q, nf_k), dist(n, nf_k))$ permet de propager et d'agréger les scores de pertinence des noeuds feuilles nf_k descendants de n pour former le score de pertinence du noeud n. Les scores sont calculés d'après 3.4, et la propagation des scores se fait en fonction des distances $dist(n, nf_k)$ qui séparent le noeud n des noeuds feuilles nf_k dans l'arbre du document (c'est à dire le nombre d'arcs dans l'arbre du document nécessaires pour joindre n et nf_k).

(2) Si $SRE_{i,j} = b\_n[]$, (c'est le cas par exemple de $SRE_{2,1}$)

$$R_{i,j} = \{(n, 0)/n \in construct(b\_n)\} \tag{3.15}$$

c'est à dire l'ensemble des noeuds ayant b_n comme nom de balise
(3) Si $SRE_{i,j} = b\_n[n\_a = v]$, (c'est le cas par exemple de $SRE_{1,1}$)

$$R_{i,j} = \{(n, 1) \quad / \quad n \in construct(b\_n), \ a \in construct(n\_a) \ estAttribut \ de \ n \ et \ valeur(a) = v)\} \tag{3.16}$$

On attribue un score de 1 (qui est le score maximal d'un noeud répondant directement aux conditions de contenu) aux noeuds vérifiant les conditions portant sur la valeur des attributs. Nous considérons en effet ces conditions comme des conditions portant sur des données et non sur du texte et nous traitons les valeurs des attributs en effectuant des correspondances *exactes* (au sens BD).

Afin d'illustrer ce traitement, considérons la requête $SRE_{3,2}$ = *titre[moteurs de recherche]* et le document *article.xml* (figure 3.3).

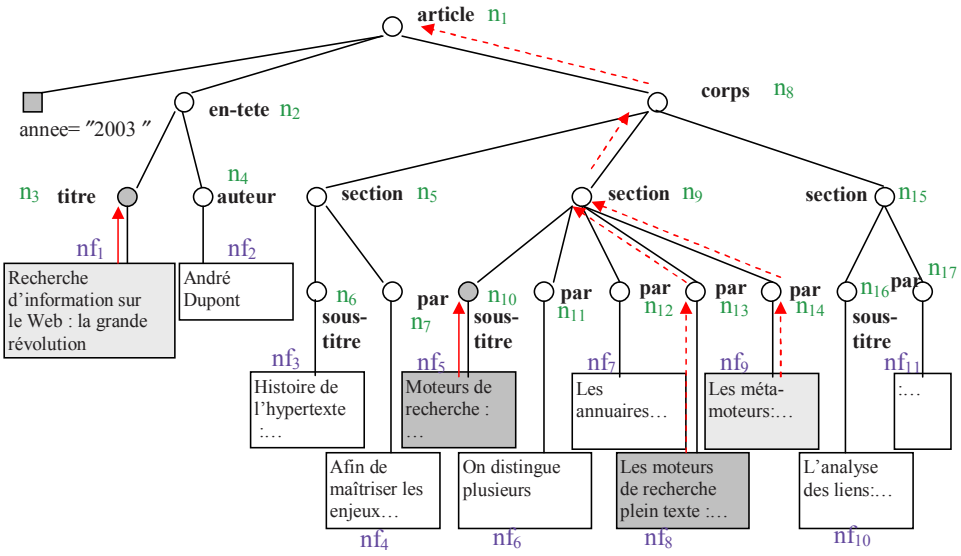


FIG. 3.3 – Exemple de traitement d'une sous-requête élémentaire

Une première étape consiste à calculer un score de pertinence des noeuds feuilles de l'index par rapport à la requête "*moteurs de recherche*". Une fois ces scores calculés, les scores non nuls sont propagés dans l'arbre du document, jusqu'à ce que des noeuds de type *titre* soient trouvés. Dans notre exemple, les noeuds feuilles nf_1, nf_5, nf_8 et nf_9 ont un score de similarité à la requête non nul, et *construct(titre)*=$\{n_3, n_{10}\}$. Lors de la création de l'ensemble *construct(titre)*, l'index Dictionnaire est utilisé sur la balise *titre*, ce qui nous permet d'inclure le noeud n_{10} portant la balise *sous-titre*.
L'ensemble $R_{3,2}$ résultat de $SRE_{3,2}$ est alors $\{(n_3, p_{n_3}), (n_{10}, p_{n_{10}})\}$,
où p_{n_3}=$F(RSV_m$ *(moteurs de recherche,* nf_1*), dist(*n_3*,* nf_1*))*
et $p_{n_{10}}$=$F(RSV_m$*(moteurs de recherche,* nf_5*), dist(*n_{10}*,* nf_5*))*.

De manières équivalente aux requêtes composées de simples conditions de contenu, le score de pertinence des noeuds feuilles $RSV_m(q, nf)$ est évalué selon l'équation 3.4. Dans les expérimentations présentées au chapitre 4 et dans [180], plusieurs formules pour le calcul du poids des termes des noeuds feuilles et de la requête ont été expérimentées, et comme pour les requêtes contenant des seules conditions de contenu, une simple formule *tf-ief* permet d'obtenir des performances optimales (voir équation 3.6).

Dans l'exemple que nous avons présenté ci-dessus, la fonction F_k *(*RSV_m*(q,* nf_k*), dist(n,*nf_k*))* ne fait que propager les scores des noeuds feuilles nf_1 et nf_5.

Son objectif est cependant aussi d'agréger les poids des noeuds feuilles lorsque plusieurs noeuds feuilles possèdent un même ancêtre répondant aux conditions de structure. Par exemple, pour répondre à une sous-requête *section[moteurs de recherche]* les scores de pertinences des noeuds feuilles nf_5, nf_8 et nf_9 doivent être agrégés et éventuellement diminués pour former le score de pertinence du noeud n_9.
Plusieurs fonctions ont été évaluées pour $F_k(RSV_m(q, nf_k), dist(n, nf_k))$ et sont présentées dans [184] et dans le chapitre 4. La fonction 3.17 permet d'obtenir les meilleurs résultats :

$$F_k(RSV_m(q, nf_k), dist(n, nf_k)) = \sum_{nf_k \in F_n} \alpha^{dist(n,nf_k)-1} * RSV(q, nf_k) \quad (3.17)$$

avec $\alpha \in]0..1]$ permettant d'ajuster l'importance de la distance entre les noeuds durant la propagation.
Cette formule est comparable à celle utilisée pour les requêtes orientées contenu. L'introduction du paramètre $|F_n^p|$ représentant le nombre de noeuds feuilles descendants de n et ayant un score non nul ne permet cependant pas d'améliorer les performances, contrairement aux résultats obtenus pour les requêtes orientées contenu. Nos expérimentations ont en outre montré que lors de la propagation, la distance entre les noeuds a une importance moindre dans le cas de requêtes orientées contenu et structure que dans le cas de requêtes orientées contenu seulement. $\alpha = 0.9$ nous permet en effet d'obtenir des performances optimales.

3.7.3 Traitement des requêtes de type P2

L'évaluation des requêtes de type P2 consiste au traitement des *conditions booléennes* de la requête. Une fois que les requêtes $SRE_{i,j}$ ont été traitées, les requêtes $P2_i$ de type $P2$ sont reconstituées grâce aux opérateurs commutatifs $\oplus_{ET}$ et $\oplus_{OU}$ définis ci-dessous.

Définition 1 : Soient deux ensembles de paires *(noeud, pertinence)* $A = \{(n, p_n)\}$ et $B = \{(m, p_m)\}$.

$$A \oplus_{ET} B = \{(l, p_l) \quad / \quad l \text{ est le plus proche ancetre commun de } m \text{ et } n, \text{ ou } l = m(\text{respectivement } n) \text{ si } m \text{ (resp. } n) \text{ est ancetre de } n \text{ (resp. } m), \forall m, \ n \text{ appartenant au meme document et } p_l = agreg_{ET}(p_n, p_m, dist(l, n), dist(l, m))\} \quad (3.18)$$

$$A \oplus_{OU} B = \{(l, p_l) \quad / \quad l = n \in N \text{ et } p_l = p_n$$

$$ou\ l = m \in M\ et\ p_l = p_m$$
$$ou\ l = n = m\ et\ p_l = agreg_{OU}(p_n, p_m)\} \quad (3.19)$$

Où $agreg_{ET}(p_n, p_m, dist(l,n), dist(l,m)) = p_l$ et $agreg_{OU}(p_n, p_m) = p_l$ définissent la façon dont les pertinences p_n et p_m des noeuds n et m sont agrégées pour former une nouvelle pertinence p_l.

Soit l'ensemble résultat R_i d'une requête $P2_i$. Alors :

$$Si\ P2_i = SRE_{i,j}, alors\ R_i = R_{i,j} \quad (3.20)$$
$$Si\ P2_i = SRE_{i,j}\ ET\ SRE_{i,k}, alors\ R_i = R_{i,j} \oplus_{ET} R_{i,k} \quad (3.21)$$
$$Si\ P2_i = SRE_{i,j}\ OU\ SRE_{i,k}, alors\ R_i = R_{i,j} \oplus_{OU} R_{i,k} \quad (3.22)$$

Le résultat d'une requête $P2_i$ est donc un ensemble R_i composé de paires formées de noeuds l et du poids de pertinence p_l qui leur est associé.

Afin d'illustrer ce traitement des requêtes de type P2, considérons la requête $P2_3$ issue de notre exemple : $P2_3$ = *(*$SRE_{3,1}$*=par[annuaire]) ET (*$SRE_{3,2}$*= titre[moteurs de recherche])*. L'ensemble résultat de la requête $SRE_{3,1}$ est $R_{3,1} = \{(n_{12}, p_{n_{12}})\}$, et l'ensemble résultat de la requête $SRE_{3,2}$ est composé de deux noeuds : $R_{3,2} = \{(n_3,\ p_{n_3}),\ (n_{10},\ p_{n_{10}})\}$. L'ensemble R_3 résultat de $P2_3$ sera alors composé de deux noeuds et des pertinences associées, comme le montre la figure 3.4 ci-dessous.

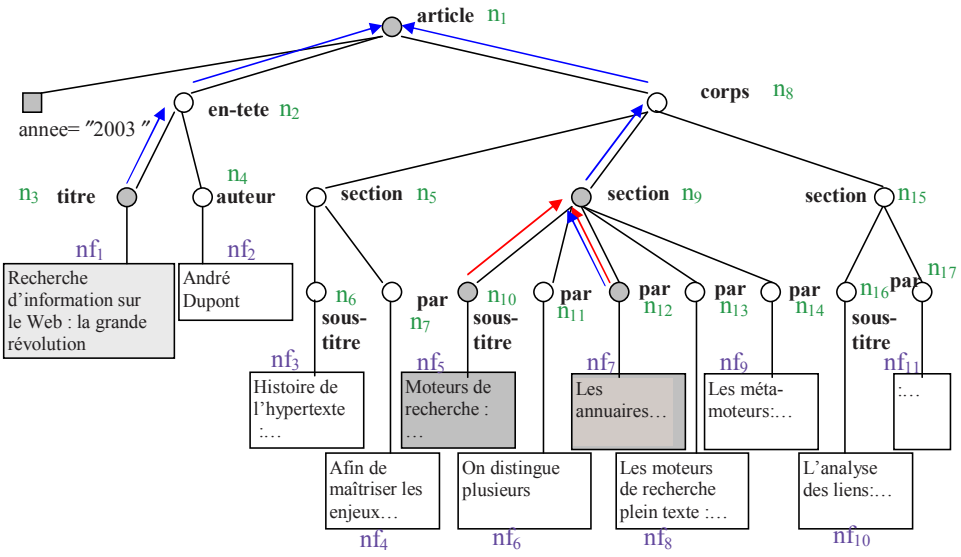


FIG. 3.4 – Exemple de traitement d'une requête de type P2

$R_3 = R_{3,1} \oplus_{ET} R_{3,2} = \{(n1, p_{n_1}), (n_9, p_{n_9})\}$,
où p_{n_1}=$agreg_{ET}(p_{n_3},\ p_{n_{12}},\ dist(n_1,n_3),\ dist(n_1,n_{12}))$=$agreg_{ET}(p_{n_2},\ p_{n_{11}}, 2, 3)$ et

p_{n_9}=$agreg_{ET}$(p_{n_10}, $p_{n_{12}}$, dist(n_9,n_{10}), dist(n_9,n_{12}))= $agreg_{ET}$($p_{n_{10}}$, $p_{n_{12}}$,1 ,1).

La fonction $agreg_{ET}(p_n, p_m, dist(l,n), dist(l,m)) = p_l$ calcule un nouveau score de pertinence pour le noeud l à partir de deux pertinences p_n et p_m et de la distance qui sépare l de n et m.
Plusieurs fonctions ont été évaluées pour $agreg_{ET}$. Ces fonctions sont présentées dans [184] et dans le chapitre 4 , et la fonction 3.23 permet d'obtenir les meilleurs résultats :

$$agreg_{ET}(p_n, p_m, dist(l,n), dist(l,m)) = \frac{p_n}{dist(l,n)} + \frac{p_m}{dist(l,m)} \tag{3.23}$$

La fonction $agreg_{OU}(p_n, p_m)$ est quant à elle une simple fonction *Somme*.

3.7.4 Traitement des requêtes de type P3

Le traitement des requêtes de type $P3$ ($P3 = //P2_1//P2_2//...P2_n$) consiste à évaluer les conditions hiérarchies de la requête. On utilise pour ce faire les ensembles résultats des requêtes de type $P2$, qui sont combinés grâce à l'opérateur non-commutatif Δ défini ci-dessous :

Définition 2 : Soient deux ensembles de paires *(noeud, pertinence)* $R_i = \{(n, p_n)\}$ et $R_{i+1} = \{(m, p_m)\}$

$$R_i \Delta R_{i+1} = \{(n, p'_n)\} \tag{3.24}$$

avec

$$p'_n = \begin{cases} p_n + prop\_ag(dist(m,n), p_n, p_m) \ si \ n \in R_i \ est \ Ancetre \ de \ m \in R_{i+1} \\ p_n \ sinon \end{cases} \tag{3.25}$$

Où $prop\_ag(dist(m,n), p_n, p_m) -> p'_n$ permet d'agréger les pertinences p_m du noeud m et p_n du noeud n en fonction de la distance qui sépare les deux noeuds, pour obtenir la nouvelle pertinence p'_n du noeud n.

L'ensemble résultat R d'une requête de type $P3$ est alors défini ainsi :

$$R = R_1 \Delta (R_2 \Delta (R_3 \Delta \ldots)) \tag{3.26}$$

ce qui revient en fait à propager de bas en haut dans l'arbre du document les poids des noeuds résultats des sous-requêtes $P2_2$ à $P2_n$ vers les noeuds résultats de $P2_1$, qui constitueront l'ensemble renvoyé à l'utilisateur.

3.7.5 Traitement des requêtes de type P4

Alors que pour les requêtes de type P3, les scores des noeuds sont propagés de bas en haut dans l'arbre du document, dans le cas de requêtes de type P4, ces scores peuvent être propagés de haut en bas, et ce à cause de la présence d'un élément cible, qui indique le type de noeud à renvoyer à l'utilisateur. Ceci nécessite la définition de l'opérateur non-commutatif $\bigtriangledown$ défini ci-dessous :

Définition 3 : Soient deux ensembles de paires *(noeud, pertinence)* $R_i = (n, p_n)$ et $R_{i+1} = (m, p_m)$

$$R_i \bigtriangledown R_{i+1} = \{(m, p'_m)\} \tag{3.27}$$

avec

$$p'_m = \begin{cases} p_m + prop\_ag(dist(m,n), p_n, p_m) \ si \ m \in R_{i+1} \ estDescendant \ de \ n \in R_i \\ p_m \ sinon \end{cases} \tag{3.28}$$

Ainsi, l'opérateur $\bigtriangledown$ est utilisé pour propager de haut en bas dans l'arbre du document les poids des noeuds résultats de sous requêtes $P2_1$ à $P2_{i-1}$ vers les noeuds résultats de $P2_i$, qui constituent les éléments cibles demandés par l'utilisateur.
L'ensemble résultat R d'une requête de type $P4$ est alors défini en trois étapes :

1. Propagation des poids des noeuds des ensembles $R_{i+1}, \ldots R_n$ de bas en haut vers les noeuds de l'ensemble constitué des éléments cibles R_i :

$$SR_1 = R_i \Delta (R_{i+1} \Delta (R_{i+2} \Delta \ldots)) \tag{3.29}$$

2. Propagation des poids des noeuds des ensembles $R_1, \ldots, R_{i-1}$ de haut en bas vers les noeuds de l'ensemble constitué des éléments cibles R_i :

$$SR_2 = (((R1 \bigtriangledown R2) \bigtriangledown R3) \bigtriangledown \ldots) \bigtriangledown R_i \tag{3.30}$$

3. Union des deux ensembles créés précédemment :

$$R = SR_1 \ U \ SR_2 \tag{3.31}$$

L'ensemble résultat R de la requête *//article[@annee=2003]// ec : section[]// par[annuaire] ET titre[moteurs de recherche]* est ainsi obtenu de la façon suivante. Nous avons, à l'issu des étapes précédentes, $R_1 = \{(n_1,1)\}$, $R_2 = \{(n_5,0), (n_9,0), (n_{15},0)\}$, $R_3 = \{(n_1,p_{n_1}), (n_9,p_{n_9})\}$.
On a alors $SR_1 = R_2 \ \Delta \ R_3 = \{(n_5,0),(n_9,p'_{n_9})(n_{15},0)\}$.
Notons que la paire (n_1, p_{n_1}) faisant partie de l'ensemble R_3 est ignorée, car n_1 n'est pas un noeud descendant de n_5,n_9 ou n_{15}.
On a ensuite : $SR_2 = R_1 \bigtriangledown R_2 = \{(n_5,p''_{n_5}), (n_9,p''_{n_9}), (n_{15},p''_{n_{15}})\}$, avec $p''_{n_5} =$

propag (dist(n_5,n_1), p_{n_5} , p_{n_1})= prop_ag(2, 0 ,1)=p''_{n_9}=$p''_{n_{15}}$.
Finalement, R= $\{(n_9,\ p'_{n_9}+p''_{n_9}),\ (n_5,p''_{n_5}),\ (n_{15},p''_{n_{15}})\ \}$, comme le montre la figure 3.5 ci-dessous.

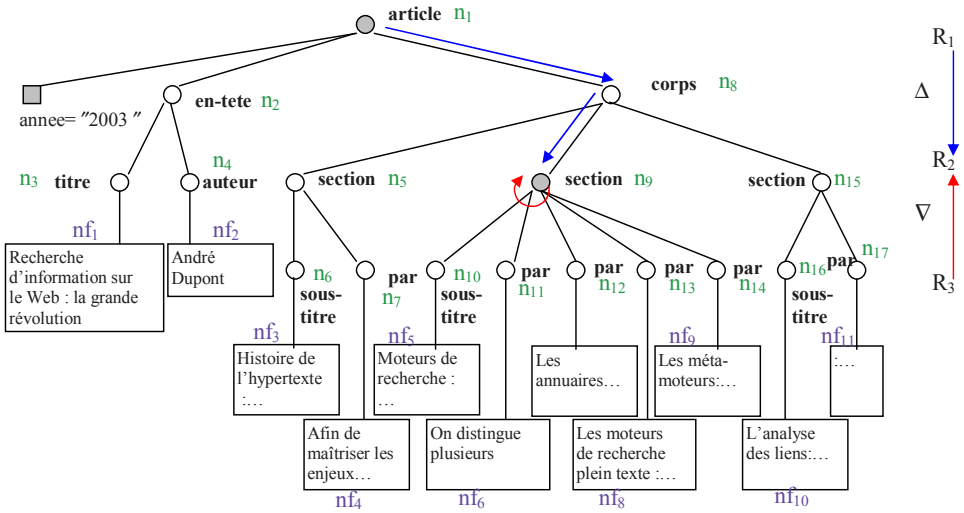


FIG. 3.5 – Exemple de traitement d'une requête de type P4 : comparaison de l'arbre du document et de l'arbre de la requête

La fonction *prop_ag(dist(n,m), p_n, p_m)* utilise elle-aussi la distance qui sépare les noeuds dans l'arbre du document pour propager les poids des noeuds et calculer de nouvelles pertinences. Cette fonction, combinée à l'opérateur U (union) de l'équation 3.31, permet d'ajuster l'importance donnée à la structure dans la requête : (i) les requêtes peuvent être traitées de manière *stricte*, et alors toutes les conditions sur la structure doivent être respectées, (ii) ou alors de manière *vague* et dans ce cas certaines conditions pourront ne pas être respectées. Dans ce dernier cas, les éléments répondant de manière stricte à la requête possèdent tout de même un meilleur score de pertinence, et sont donc renvoyés en premier dans la liste des résultats.
Plusieurs fonctions ont été évaluées pour *prop_ag(dist(n,m), p_n, p_m)*, et sont présentées dans le chapitre 4 et dans [184]. La fonction 3.32 nous permet d'obtenir les meilleurs précisions moyennes :

$$prop\_ag(dist(n,m), p_n, p_m) = \frac{p_n + p_m}{dist(n,m)} \tag{3.32}$$

Remarque Le modèle que nous venons de présenter pour le traitement des requêtes composées de conditions de structure et de contenu, permet, grâce à plusieurs propagations dans l'arbre des documents, de déterminer un score de resssemblance entre ce dernier et l'arbre de la requête. La façon dont les fonctions de propagation sont ajustées permet de répondre aux conditions de structure de manière plus ou moins stricte, et ce selon la tâche utilisateur à

laquelle on souhaite répondre.
Par exemple, considérons le document de la figure 3.6 et la requête *//a[contenu1] // i[contenu2] // ec : c[contenu3].*

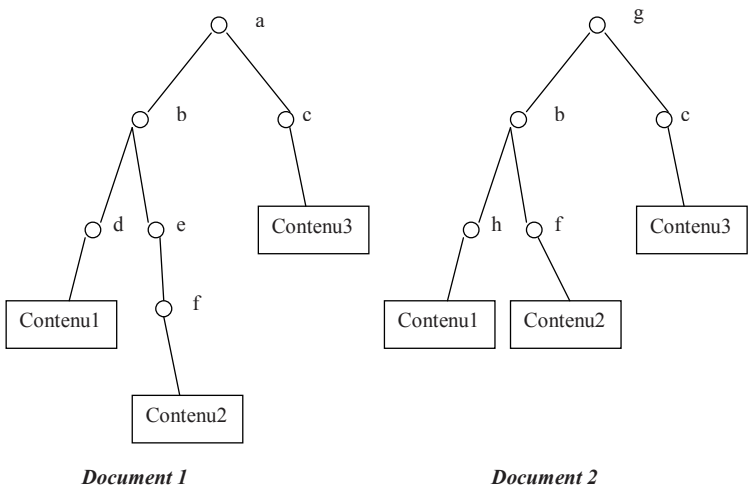


FIG. 3.6 – Exemple de traitement vague de la structure des documents

Les noeuds $/a/c$ du document 1 et $/g/c$ du document 2 auront un score de pertinence non nul, même si aucun des deux ne respecte toutes les conditions de la requête. Le noeud $/a/c$ du document 1 sera cependant mieux classé dans la liste des résultats.
Si l'on considère maintenant la requête $//b[contenu1]//ec : f[contenu2]$, le noeud $/g/b/f$ du document 2 sera mieux classé que le noeud $/a/b/e/f$ du document 1, et ce à cause de la distance qui sépare b et f dans les arbres des documents.
De la même façon, un noeud dont le chemin ne respecterait pas tout l'*ordre* des conditions hiérarchique de la requête aurait un score de pertinence non nul, mais serait moins bien classé qu'un noeud dont le XPath respecterait cet ordre. Par exemple le noeud $/a/b/d$ du document 1 aura un score de pertinence non nul à la requête *//a[]//d[]//b[contenu2].*

Nous avons réalisé de nombreuses expérimentations concernant la façon dont ces conditions de structure doivent être traitées, et les résultats de ces expérimentations sont présentés dans le chapitre 4.

3.8 Prototype

3.8.1 Architecture générale

L'ensemble des modules proposés a donné lieu au développement d'un prototype permettant l'indexation et l'interrogation de collections de documents XML. Le prototype est réalisé entièrement en langage Java (1.3) en utilisant des API telles que l'API SAX de Xerces pour parser les documents XML et JDBC pour l'accès aux bases de données. L'architecture du prototype est présentée dans la figure 3.7.

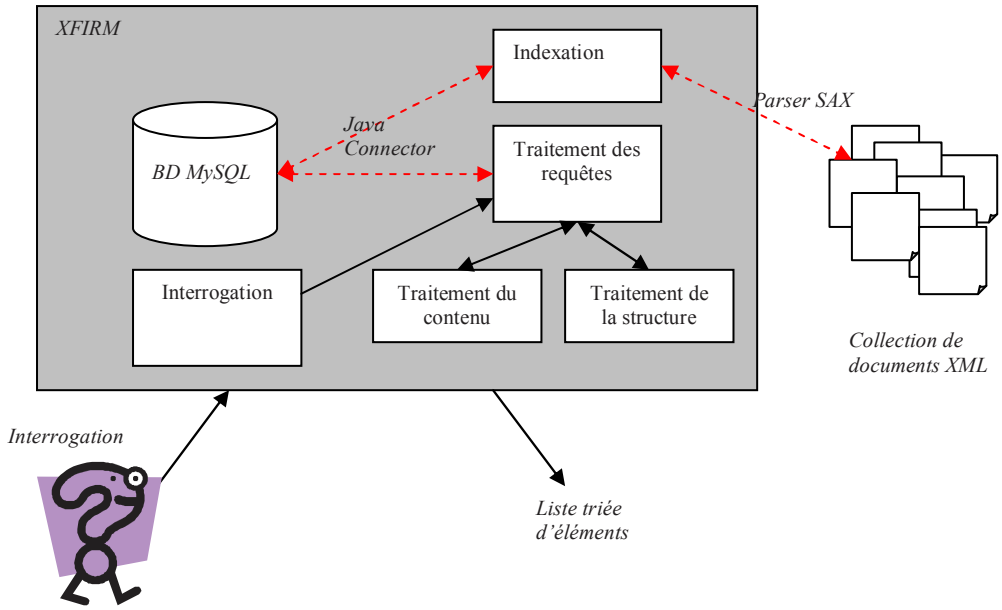


FIG. 3.7 – Architecture générale du système XFIRM

La base de données MySQL stockant les index est l'élément central de cette architecture. On trouvera une description détaillée de cette base dans le paragraphe 3.7.2. En complément de cette base, l'architecture comprend :

1. un module d'*indexation*, qui parse la collection de documents XML, lemmatise les termes et supprime les mots vides, et crée les tables de l'index. Ce module permet en outre une indexation incrémentale, ce qui permet de mettre à jour les index lors de l'insertion d'un nouveau document dans la collection.
2. un module d'*interrogation*, qui gère les requêtes utilisateurs (exprimées en langage XFIRM) en les découpant en sous-requêtes
3. un *module de traitement des requêtes*, reposant lui-même sur un module de traitement du contenu et un module de traitement de la structure. C'est

ce module qui renvoie à l'utilisateur une liste triée d'éléments répondant à sa requête.

3.8.2 Schéma de stockage

3.8.2.1 Modèle de représentation la structure arborescente des documents

Afin de pouvoir facilement naviguer dans l'arbre et déterminer rapidement les relations ancêtres-descendants ainsi que permettre l'accès rapide à un noeud, nous proposons la représentation suivante des noeuds et des attributs, basée sur l'approche XPath Accelerator [93].
Noeud : $n_i = (pre, post, parent, attribut)$
Noeud feuille : $nf_i = (pre, post, parent, \{(t_1, w_1^i), (t_2, w_2^i), \ldots (t_n, w_n^i)\})$
Attribut : $a_i = (pre, val)$
Un noeud est défini grâce à ses valeurs de pré-ordre et post-ordre (*pre* et *post*), la valeur de pré-ordre de son noeud parent (*parent*), et selon que ce soit un noeud interne ou un noeud feuille, par un champ indiquant la présence d'attributs (*attribut*) ou les termes t_j qui le composent avec leurs poids w_j^i respectifs. Un attribut est défini par la valeur de pré-ordre du noeud auquel il se rattache (*pre*) et par sa valeur (*val*).
Les valeurs de pré-ordre et post-ordre sont assignées aux noeuds comme suit : en chargeant un nouveau document, on effectue un parcours séquentiel de la représentation en arbre du document structuré. Un parcours préfixé permet d'assigner à chaque noeud visité une valeur croissante de pré-ordre (*pre*) avant que ses noeuds descendants ne soient aussi récursivement visités de gauche à droite. D'une manière inverse, la valeur de post-ordre (*post*) d'un noeud lui est assignée lors d'un parcours postfixé, c'est à dire une fois que tous ses noeuds descendants ont été visités de gauche à droite.
La figure 3.8 illustre l'assignement des valeurs de pré-ordre et post-ordre aux noeuds du document XML *article.xml* (voir tableau 2.1).

Si l'on transpose tous les noeuds dans un espace à deux dimensions basé sur les coordonnées de pré-ordre et post-ordre, on peut exploiter les propriétés suivantes illustrées par l'exemple de la figure 3.9. Etant donné un certain noeud n (le noeud */article[1]/ corps[1]/section[2]* dans l'exemple) :

- tous les ancêtres de n sont au-dessus à gauche de la position de n dans le plan
- tous ses descendants sont en dessous à droite
- tous les noeuds le précédant dans la lecture séquentielle du document sont en-dessous à gauche

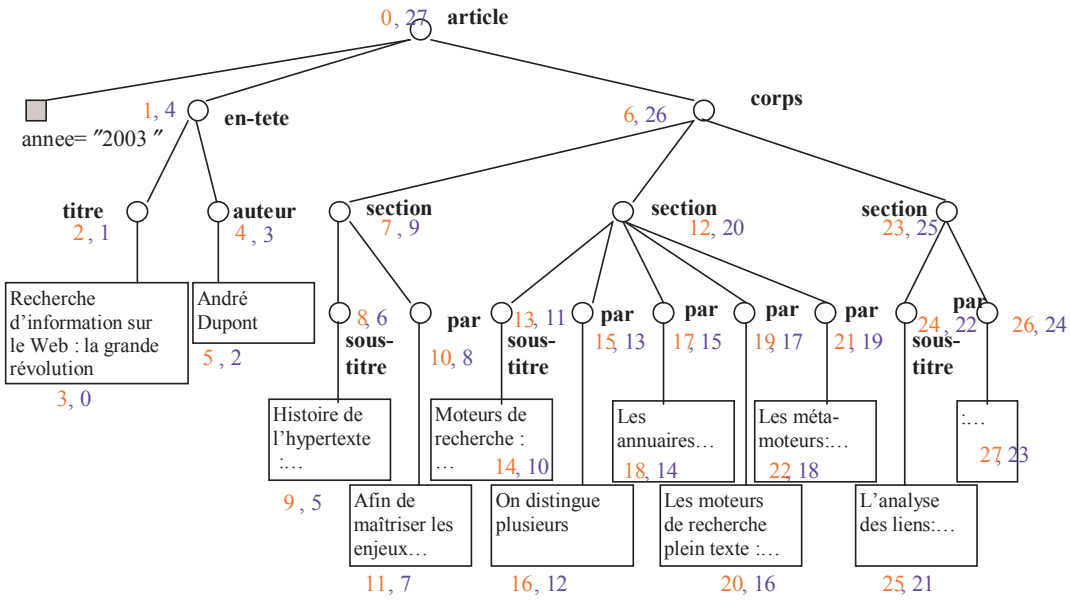


FIG. 3.8 – Valeurs de pré-ordre et de post-ordre assignées aux noeuds du document XML *article.xml*

– la partition du plan au dessus à droite comprend tous les noeuds successeurs dans la lecture séquentielle du document.

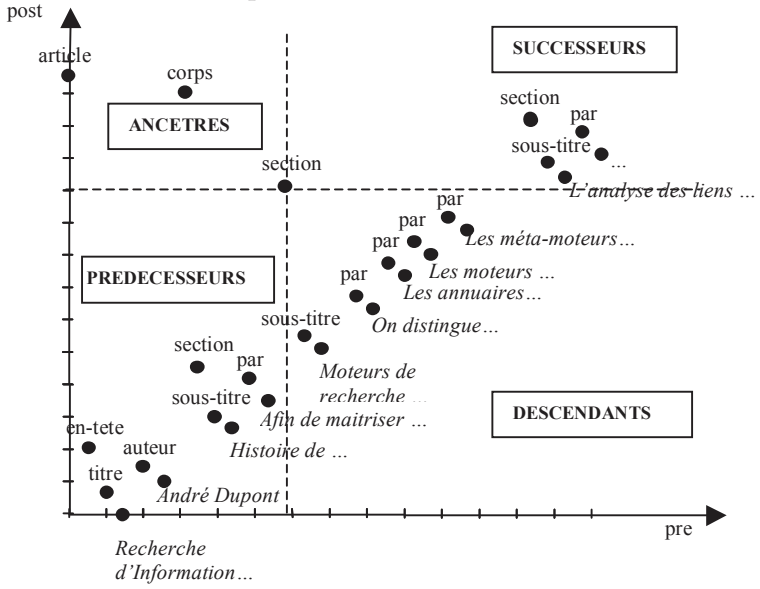


FIG. 3.9 – Représentation du document *article.xml* dans un espace à deux dimensions basé sur les coordonnées de pré-ordre et post-ordre

Ainsi, les requêtes XPath [45] du type : *Child*, *Descendant*, *Parent*, *Ancestor*, *following*, *preceding*, *following-sibling*, *preceding-sibling* sont rapidement traitées. Par exemple :

Un noeud *n'* est ancêtre de *n* si *pre(n') < pre(n)* et *post(n')> post(n).*
Outre le traitement des expressions XPath, cette représentation des noeuds est particulièrement intéressante pour une navigation dans la structure des documents. Contrairement à d'autres approches basées sur des index de structure, elle permet de répondre à des expressions XPath qui n'ont pas pour origine la racine du document, et ce en élaguant l'arbre représentant le document. Elle permet de plus de reconstruire rapidement le XPath correspondant à un noeud. Enfin, elle permet de gérer des collections de documents hétérogènes (possédant des DTDs différentes) grâce à une représentation générique de la structure de ces derniers.

3.8.2.2 Indexation

Le choix des noeuds à indexer (c'est à dire de l'information structurelle à conserver) est d'une importance capitale pour les performances du modèle de recherche, puisqu'il détermine l'unité d'information minimale qui pourra être renvoyée à l'utilisateur. Ce choix est effectué au début du processus d'indexation. Dans notre modèle, plusieurs scénarios sont possibles :

- tous les noeuds sont indexés ;
- le choix des noeuds à indexer est fait manuellement selon la ou les DTD(s) des documents ou grâce à des statistiques sur la collection, et une liste d'éléments non indexables est créée.

La sélection manuelle ou automatique des noeuds a pour but de séparer le contenu orienté données du contenu texte des documents XML. Ces derniers possèdent en effet généralement les deux types d'information, et dans la plupart des cas, seul le contenu texte satisfait le besoin en information de l'utilisateur. Le contenu orienté données est alors non seulement une réponse non souhaitée par l'utilisateur, mais apparaît aussi comme du bruit ayant un effet négatif sur les résultats de la recherche. Par exemple, chaque volume d'une revue contient un index avec des mots-clés. Le contenu de cet index peut être retourné pour certaines requêtes et est pourtant non pertinent par rapport au besoin de l'utilisateur. Supprimer les noeuds qui a priori ne sont pas utiles pour la recherche permet de réduire la taille des index et donc d'améliorer le temps de traitement des requêtes. L'information contenue dans les noeuds non indexés n'est cependant pas perdue : elle est propagée dans l'arbre du document jusqu'à ce qu'un noeud "indexable" soit rencontré.
Lors des expérimentations présentées dans le chapitre 4, deux index correspondant aux deux situations ci-dessus ont été créés. L'index permettant de restituer la structure complète des documents nous permet d'obtenir de meilleures performances en termes de précisions moyenne, ce qui tend à prouver que *toute* la structure des documents est importante dans notre modèle pour le calcul de la pertinence des noeuds.

L'information textuelle contenue au niveau des noeuds feuilles "indexables" est lemmatisée. La lemmatisation peut être effectuée avec l'algorithme de Porter [160] pour les documents de langue anglaise ou bien en effectuant des troncatures pour les autres langues. Une liste de mots vides est utilisée pour supprimer les termes qui n'apportent pas de sens au contenu des éléments, comme par exemple les pronoms ou les déterminants.
Les noms de balises ainsi que les noms et valeurs d'attributs ne subissent quant à eux aucun traitement avant d'être indexés.
On construit en outre un *dictionnaire des noms de balises*, qui permet de regouper les balises de la collection ayant la même sémantique. L'utilisation de ce dictionnaire permet d'étendre les requêtes des utilisateurs et d'établir des liens entre des documents suivants des DTDs différentes. Par exemple, les balises *titre* et *sous-titre* peuvent être considérées comme équivalentes.

3.8.2.3 Structure de la base

Comme les SRI traditionnels, XFIRM propose la construction de structures d'index pré-calculées qui sont utilisées pour l'évaluation des différentes conditions de recherche énoncées dans les requêtes. Ces index sont basés sur la modélisation des noeuds que nous avons présentée ci-dessus.
Les index sont stockés sous forme de tables dans une base de données relationnelle MySQL. Afin d'obtenir les différents index, les documents à indexer sont parcourus à l'aide d'un parseur de type SAX On trouvera un schéma générique de la base sur la figure 3.10.

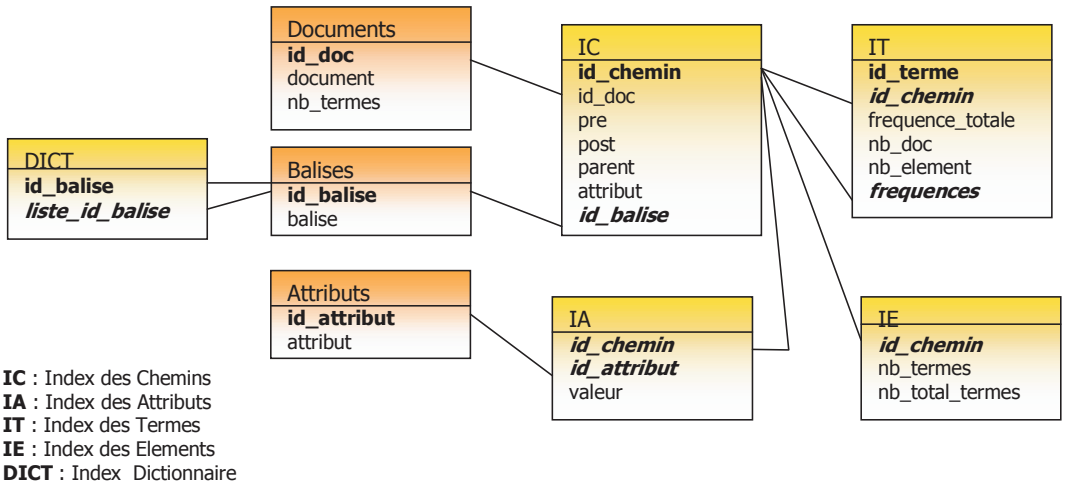


Fig. 3.10 – Schéma de la base de données contenant les index

Trois tables génériques, utilisées par les index principaux, sont présentes dans la base de données : la table *Documents*, la table *Balises* et la table

Attributs. Le schéma de ces tables est détaillé dans le tableau 3.2.

Table	Description
Documents	*Documents(doc_id, document, date, nb_termes)* *doc_id* est l'identifiant unique de chaque document, *document* est le nom de fichier du document, *date* est la date d'insertion dans l'index du document, et *nb_termes* est le nombre total de termes du document
Balises	*Balises(balise_id, balise)* *balise_id* est l'identifiant unique de chaque nom de balise et *balise* est le nom de la balise
Attributs	*Attributs(att_id, attribut)* *att_id* est l'identifiant unique de chaque nom d'attribut et *attribut* est le nom de l'attribut

Tab. 3.2 – Tables génériques du modèle physique de XFIRM

Les index principaux, au nombre de cinq, sont les suivants :
- *L'index des chemins*(IC) permet de reconstituer la structure des documents ;
- *L'index des termes (IT)* donne pour chaque terme de la collection les éléments associés et permettra de calculer diverses mesures de pertinence en fonction du modèle de recherche choisi : il correspond en fait à un fichier inverse traditionnel ;
- *L'index des éléments (IE)* décrit le contenu de chaque noeud feuille, et permettra de faire des évaluations de pertinence sur des noeuds précis ;
- *L'index des attributs (IA)* donne pour chaque attribut ses différentes valeurs ;
- et enfin le *dictionnaire (DICT)* permet de regrouper les balises de la collection ayant la même sémantique. En effet, la qualité des recherches sur des données semi-structurées peut être améliorée en utilisant la sémantique du nom des éléments [201]. L'utilisation du dictionnaire permet d'étendre les requêtes des utilisateurs et d'établir des liens entre des documents suivants des DTDs différentes. Par exemple, les balises *titre* et *sous-titre* peuvent être considérées comme équivalentes.

On trouvera enfin dans le tableau 3.3 la description détaillée de ces index principaux.

Notons de plus que les termes contenus dans l'IT sont lemmatisés. La lemmatisation peut être effectuée avec l'algorithme de Porter [160] pour les documents de langue anglaise ou bien en effectuant des troncatures pour les autres langues. Une liste de mots vides est utilisée pour supprimer les termes qui n'apportent pas de sens au contenu des éléments, comme par exemple les pronoms ou les déterminants.

Index	Description
IC	*Chemins (chemin_id, doc_id, pre, post, parent, attribut, balise_id)* *chemin_id* est l'identifiant unique de chaque chemin, *doc_id* est l'identifiant du document concerné, *pre* et *post* sont les valeurs de prédécesseurs et successeurs, *parent* est la valeur de prédécesseur du parent de l'élément, *attribut* est un booléen indiquant la présence d'attribut pour l'élément concerné, et *balise_id* est l'identifiant de la balise de l'élément concerné. Si le champ *balise_id* est nul pour un certain *chemin_id*, l'élément est alors un élément feuille de type #PCDATA et on trouvera son contenu dans l'index des éléments.
IT	*TermesElements (terme_id, terme, total_fréquence, nb_doc, nb_elt, fréquences)* *terme_id* est l'identifiant unique de chaque terme, *terme* est le terme lui même, *total_fréquence* est la fréquence totale du terme dans la collection, *nb_doc* est le nombre total de documents dans lesquels le terme apparaît, *nb_elt* est le nombre total d'éléments (c'est à dire de chemins) dans lesquels le terme apparaît et fréquences est un champ de type BLOB (Binary Long Object) contenant pour chaque élément où le terme apparaît (élément représenté par *chemin_id*) le nombre d'occurrences du terme, ainsi que les positions auxquelles il apparaît. Par exemple, la chaîne " 2 1 2/ 21 2 4 8 " indique que le terme t est présent 1 fois dans l'élément 2 à la position 2 et 2 fois dans l'éléments 21 aux positions 4 et 8.
IE	*ElementsTermes (chemin_id, nb_termes, nb_total_termes)* *chemin_id* est l'identifiant de chaque chemin, *nb_termes* est le nombre de termes uniques inclus dans l'élément concerné, *nb_total_termes* est le nombre de termes inclus dans l'élément concerné
IA	*ValeursAttributs (chemin_id, attribut_id, valeur)* *chemin_id* est le noeud auquel se rattache l'attribut, *attribut_id* est l'identifiant de l'attribut (en référence à la Table Attributs) et *valeur* est une chaîne de caractère contenant la valeur de l'attribut.
DICT	*Dictionnaire (balise_id, ListeBalise_id)* *balise_id* est un identifiant de balise et *ListeBalise_id* est une liste d'identifiants de balise ayant une sémantique proche de *balise_id*.

Tab. 3.3 – Index du modèle physique de XFIRM

Les structures de stockage que nous venons de présenter contiennent toutes les informations nécessaires pour appliquer différents modèles de RI, tant sur des requêtes portant seulement sur le contenu des documents que des requêtes plus précises portant aussi sur leur structure. Les différents index étant stockés dans une base de données, toutes les fonctions usuelles des bases de données (comme les jointures, les projections ou le tri) ne sont pas à réimplémenter. De plus, la mise à jour des index dans le cas de suppression ou d'insertion de documents est relativement simple.

3.9 Conclusion

Dans ce chapitre, nous avons présenté XFIRM, un modèle flexible pour la recherche d'information dans des documents structurés. Le but de notre modèle est de renvoyer à l'utilisateur les unités d'information (c'est à dire les noeuds des documents XML) les plus spécifiques et exhaustives répondant à son besoin en information.
Ce modèle repose sur un modèle de représentation générique des données, permettant de stocker l'arborescence des documents XML tout en gardant les fonctionnalités orientées RI traditionnelles. Le modèle de représentation permet en outre l'implémentation de nombreux modèles de recherche ainsi que le traitement de collections hétérogènes (c'est à dire ne suivant pas la même DTD). Nous avons proposé un langage de requête associé, qui autorise l'utilisateur à exprimer son besoin selon divers degrés de précision. Si l'utilisateur a un besoin peu défini ou qu'il ne connaît pas du tout la structure des documents qu'il interroge, il pourra exprimer son besoin à travers de simples mots-clés, et il laissera le système décider de la granularité appropriée de l'information à renvoyer. Si au contraire l'utilisateur a un besoin précis, il pourra introduire des conditions de structure dans sa requête, éventuellement reliées de manière à exprimer une hiérarchie.
Le modèle de recherche que nous proposons repose sur une méthode de propagation des pertinences dans l'arbre du document. Le traitement des requêtes diffère selon leur type :

- pour les requêtes orientées *contenu*, nous nous sommes attachés à modéliser la notion d'informativité d'un noeud. Cette informativité dépend non seulement de la pertinence des descendants du noeud (et plus particulièrement des plus petits d'entre eux) mais aussi de la pertinence de son contexte, puisque les noeuds sont organisés en document, et que les documents suivent une certaine unité de pensée, même s'ils possèdent un contenu hétérogène.
- pour les requêtes orientées *contenu et structure*, nous avons proposé plusieurs fonctions de propagation, qui nous permettent d'effectuer une com-

paraison entre l'arbre de la requête et l'arbre des documents. Ces fonctions de propagation, selon la tâche utilisateur à laquelle on cherche à répondre, permettent d'ajuster la façon (stricte ou vague) dont sont traitées les conditions de structure. Lorsqu'une correspondance vague entre l'arbre de la requête et l'arbre du document est effectuée, des documents possédant une structure différente de celle la requête peuvent être renvoyés à l'utilisateur, même si leur pertinence est plus faible que celle des documents pour lesquels toutes les conditions de structure sont respectées. Par exemple, un document possédant la structure /a/b/c sera pertinent pour une requête /a/d/c, mais aussi pour une requête /a/c/b. Lorsque l'utilisateur ne spécifie pas le type de l'élément qu'il désire voir renvoyer (pas d'élément cible), nous cherchons les noeuds les plus proches ancêtres communs des noeuds qui répondent aux conditions de structure (requête de type P2) ou bien les noeuds répondant à la première condition de structure des requêtes de type P3 (noeuds situés le plus haut dans la hiérarchie des documents).

Notre modèle apporte ainsi de la flexibilité dans la recherche à plusieurs niveaux : la structure d'index est générique et permet de traiter des collections de documents hétérogènes, le langage permet à l'utilisateur d'exprimer son besoin selon plusieurs degrés de précision, et les éventuelles conditions de structure des requêtes peuvent être traitées de manière vague. Les résultats obtenus par nos propositions sont présentés dans le chapitre suivant. Ils montrent les bonnes performances de notre approche par rapport aux approches proposées dans la littérature.

Chapitre 4

Expérimentations et résultats

4.1 Introduction

Dans ce chapitre, nous présentons les expérimentations effectuées pour évaluer l'apport des différentes propositions faites au chapitre 3. Les évaluations portent sur le modèle de recherche proposé pour les requêtes orientées contenu (de type P1) et les requêtes orientées contenu et structure (de type P2 à P4). Nous avons à cet effet organisé nos expérimentations en deux grandes parties. La première partie concerne les évaluations effectuées sur les requêtes orientées contenu. Nous avons évalué les points suivants dans notre modèle de propagation de la pertinence :

- impact de la formule de pondération des termes utilisée pour le calcul du score de pertinence des noeuds feuilles (équation 3.4) ;
- impact du paramètre distance dans la fonction de propagation (équation 3.7) ;
- impact de la longueur des noeuds dans le calcul de la dimension d'informativité ;
- impact du contexte des éléments dans le calcul de la dimension d'informativité.

Suite à ces expérimentations, nous commentons les jugements de pertinence utilisés dans le cadre de la campagne d'évaluation INEX ainsi que le principal problème auquel nos résultats sont soumis, à savoir le problème de l'imbrication des noeuds.
La seconde partie de nos évaluations concerne les requêtes orientées contenu et structure. Pour ces requêtes, les points suivants ont été évalués :

- impact de la formule de pondération des termes utilisée pour le calcul du score de pertinence des noeuds feuilles ;
- impact du paramètre distance dans les fonctions de propagation ;
- comparaison de la gestion stricte ou vague des conditions de structure.

Nous nous proposons ensuite d'évaluer l'impact de l'unité d'indexation minimale choisie sur notre modèle ainsi que la faisabilité de notre approche sur une collection de données hétérogènes (c'est à dire ne suivant pas la même DTD).

Dans ce chapitre, nous commençons par décrire de manière plus détaillée la collection de test utilisée pour nos expérimentations, à savoir la collection INEX, ainsi que les jeux de requêtes associés aux campagnes d'évaluations 2003 et 2004 (section 4.2). La section 4.3 présente nos conditions expérimentales, et les sections 4.4 et 4.5 décrivent nos expérimentations, respectivement pour les requêtes orientées contenu (de type P1) et les requêtes orientées contenu et structure (de type P2 à P4), et ce selon les canevas d'expérimentations décrits ci-dessus. Nous étudions dans la section 4.6 l'impact de l'unité d'indexation minimale choisie. La section 4.7 compare nos résultats avec les résultats des différents participants à INEX. Enfin, nous présentons dans la section 4.8 les expérimentations que nous avons menées pour la tâche hétérogène de la campagne d'évaluation INEX 2004.

4.2 Collection de test

Nos expérimentations utilisent les différents outils fournis par les deux dernières campagnes d'évaluation INEX (2003 et 2004), à savoir une collection de test, des requêtes et jugements de pertinence associés, ainsi que des mesures d'évaluation. Le fonctionnement de la campagne d'évaluation INEX a été décrit précédemment décrit au chapitre 2. Nous détaillons ici les tâches sur lesquelles nous avons menées nos expérimentations ainsi que les mesures que nous utilisons pour évaluer notre modèle.

4.2.1 Requêtes et jugements de pertinence

Afin de mener à bien nos expérimentations, nous avons utilisé deux types de requêtes INEX :

- les requêtes CO associées à la tâche de recherche CO (*Content Only task*),
- et les requêtes CAS, associées aux tâches de recherche SCAS (*Strict Content and Structure Task*) et VCAS (*Vague Content and Structure task*).

4.2.1.1 Tâche CO

La tâche CO a pour but de répondre avec des éléments/documents XML à des requêtes utilisateur contenant de simples mots-clés. Aucune indication de structure dans la requête ne peut aider les SRI à déterminer la granularité de l'information à renvoyer.
Dans nos expérimentations, nous utilisons les ensembles de requêtes fournis pour les campagnes d'évaluation 2003 et 2004. En 2003, la tâche CO était composée de 36 requêtes (avec 32 jugements de pertinence associés), et en 2004, 40 requêtes ont été mises à disposition des participants (avec 34 jugements de pertinence associés).

4.2.1.2 Tâche SCAS

La tâche SCAS consiste à répondre avec des éléments/documents XML aux topics CAS de manière stricte, c'est à dire en respectant toutes les conditions sur la structure et le contenu énoncées dans les requêtes.
Pour nos expérimentations, nous utilisons les requêtes CAS de la campagne d'évaluation 2003 (30 requêtes et 30 jugements de pertinence associés). Les jugements de pertinence ont été effectués par les participants en utilisant seulement les conditions de contenu (c'est à dire comme s'ils étaient en train de juger des requêtes CO) et les résultats ont ensuite été filtrés pour répondre aux contraintes de structure exprimées dans les requêtes.

4.2.1.3 Tâche VCAS

La tâche VCAS utilise également des requêtes CAS, mais pour lesquelles les participants peuvent répondre de manière vague, c'est à dire avec des éléments/documents qui satisfont globalement les requêtes.
Pour nos expérimentations concernant la tâche VCAS, nous utilisons les requêtes CAS de la campagne 2004 (35 requêtes et 26 jugements de pertinence associés). Comme pour la campagne d'évaluation 2003, les jugements de pertinence sont effectués par les participants en utilisant seulement les conditions de contenu des requêtes, c'est à dire exactement comme pour la tâche CO. Cependant, aucun filtre n'est ensuite appliqué pour vérifier les conditions de structure.

4.2.2 Mesures d'évaluation

Comme nous l'avons vu au chapitre 2, les mesures utilisées pour l'évaluation sont basées sur les mesures traditionnelles de rappel et précision. Afin d'obtenir des courbes de rappel/précision, les deux dimensions de pertinence (exhaustivité et spécificité) sont agrégées en une seule valeur et plusieurs fonctions d'agrégation ont été proposées lors des campagnes 2003 et 2004.
Nous retenons pour notre part celles que nous considérons comme les plus significatives :

- Afin d'évaluer la capacité de notre modèle à répondre au critère de spécificité, nous utilisons la fonction orientée spécificité ***s3_e321*** pour laquelle seule les éléments très spécifiques ont un poids de pertinence non nul, ainsi que la fonction d'agrégation généralisée orientée spécificité [109] ***sog*** (*specificity-oriented generalised*), qui a été proposée afin de mieux refléter le critère d'évaluation défini dans INEX, selon lequel la spécificité joue un rôle plus important que l'exhaustivité.
- Afin d'évaluer la capacité de notre modèle à retrouver des éléments exhaustifs, nous utilisons la fontion orientée exhaustivité ***e3_s321***, pour laquelle seuls les éléments très exhaustifs ont un score de pertinence non nul.
- Enfin, la fonction d'agrégation stricte (***s***) pour laquelle seuls les éléments très spécifiques et très exhaustifs ont un score de pertinence non nul et la moyenne de toutes les fonctions d'agrégation proposées pour la campagne 2004 (***Avg***) sont utilisées pour évaluer où se situe le meilleur compromis entre exhaustivité et spécificité.

4.3 Conditions expérimentales

4.3.1 Indexation

Lors de l'indexation de la collection, l'algorithme de Porter [160] est utilisé pour lemmatiser les termes. Une liste de mots vides est aussi consultée pour supprimer les termes qui n'apportent pas ou peu de sens au contenu des éléments, comme par exemple les pronoms ou les déterminants.
Comme nous l'avons vu plus haut, le choix de l'unité d'indexation minimale est l'une des premières problématiques soulevée lors de l'indexation des documents. Il est couramment répandu dans la littérature que ce choix implique la définition de l'unité d'information minimale qui pourra être retournée à l'utilisateur. Deux points de vues s'affrontent. Le premier prétend qu'indexer tous les noeuds présente peu d'intérêt, puisque dans le cadre d'une recherche à partir de simples mots-clés, des noeuds de type *titre* par exemple ne doivent pas

être renvoyés par le SRI à l'utilisateur car ils ne sont pas porteurs d'information. Dans ce cas-là, afin de ne pas perdre l'information textuelle portée par ces noeuds, cette dernière est propagée jusqu'au premier noeud faisant partie des noeuds sélectionnés pour faire partie de l'index [84]. Un autre point de vue serait au contraire d'indexer tous les noeuds feuilles, car cela a le double avantage d'automatiser complètement le processus d'indexation mais aussi de permettre la réutilisation de l'index pour des requêtes composées de conditions de structure, aussi spécifiques soient-elles.
Afin de confronter ces deux approches, nous avons construit deux index de la collection INEX :

– Dans le premier, certaines balises sont éliminées de l'index et le texte de leurs éventuels noeuds feuilles descendants est affecté au premier noeud faisant partie de la liste des noeuds "indexables". Ceci est en fait équivalent à simplifier la structure de l'arbre du document. Le choix des types de noeuds à supprimer de l'index est fait automatiquement, en utilisant des statistiques sur la collection : les types de noeuds comptant en moyenne moins de 2 termes (une fois les mots vides supprimés) sont écartés de l'index. Cette condition sur le nombre de termes peut paraître faible, mais elle diminue de plus de 25% le nombre de noeuds de l'index par rapport à la seconde solution proposée. Les types de noeuds supprimés sont essentiellement ceux utilisés pour la présentation des documents (balises *italique*, *gras*,...). Par exemple, l'arbre du document *article.xml* (figure 2.2) est simplifié comme indiqué sur la figure 4.1 lorsque l'on décide de ne plus indexer les noeuds de type *sous-titre* (et donc de les éliminer de la liste des réponses possibles) :

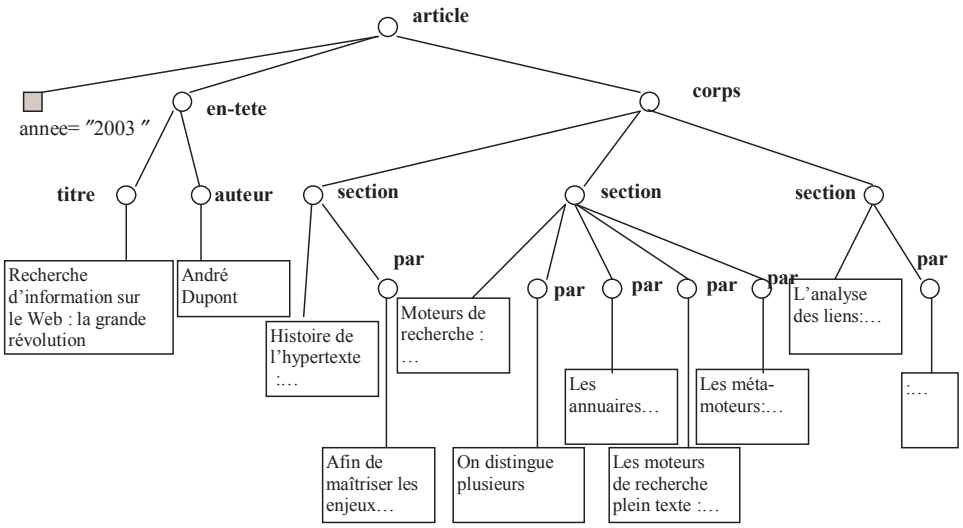


FIG. 4.1 – Exemple de simplification de l'arbre d'un document XML article.xml

– Dans le second index, toute la structure des documents est conservée.

Dans la suite des expérimentations, nous noterons ces index respectivement *IS* (*Index Simplifié*) et *IC* (*Index Complet*). L'intérêt de ces index est discuté dans

la section 4.6. Les expérimentations présentées dans les sections suivantes sont effectuées sur l'index complet IC.

4.3.2 Traitement des requêtes

Afin de pouvoir comparer nos résultats avec les résultats officiels des campagnes d'évaluation 2003 et 2004, seul le champ *Title* des Topics est utilisé pour formuler les requêtes, et pour chaque requête, on utilise les 1500 premiers éléments résultats pour l'évaluation.

Traitement des requêtes orientées contenu Afin de diminuer le temps de réponse de notre système aux requêtes, nous utilisons le principe suivant : pour chaque requête, on sélectionne les noeuds feuilles candidats à la propagation en gardant les 250 noeuds feuilles de plus fort score. On ajoute ensuite à cet ensemble les noeuds feuilles ayant une similarité à la requête non nulle contenus dans les documents associés.

Traitement des requêtes orientées structure et contenu L'index Dictionnaire est utilisé pour trouver les balises équivalentes. Par exemple, d'après les directives d'INEX, les noeuds *sec* (*section*) sont équivalents aux noeuds *ss1*, *ss2* et *ss3*.
La transformation des requêtes du langage NEXI (utilisé dans INEX) au langage XFIRM ne pose pas de problèmes particuliers. On trouvera des exemples de transformations de requêtes dans le tableau 4.1. Lorsqu'une requête INEX

INEX topic	**XFIRM query**
//article [about(.,'clustering + distributed') and about(.//sec,'java')]	// ec : article [clustering + distributed] // sec [java]
//article[about(./sec,'"e- commerce"') // abs[about(., 'trust authentication')]	//article [ ] AND sec["e- commerce"] // ec : abs [trust authentication]
//article[(.//yr='2000' OR .//yr='1999') AND about(., "intelligent transportation system"')// sec [about(.,'automation +vehicle)]	//article ["intelligent transportation system"] // ec : sec [automation + vehicle]

Tab. 4.1 – Transformation de requêtes INEX en requêtes XFIRM

contient une condition sur la date de publication d'un article (comme c'est le cas pour la dernière requête du tableau 4.1), cette condition n'est pas traduite en langage XFIRM, car la propagation sur un terme trop commun (comme une date) est trop longue. Pour résoudre ce problème, les requêtes sont traitées sans

cette condition, et les résultats sont ensuite triés sur la date de publication de l'article.

4.4 Expérimentations sur les requêtes orientées contenu

Les expérimentations présentées dans cette section ont pour but de quantifier l'impact sur l'exhaustivité et la spécificité des paramètres suivants de notre modèle :

- la fonction de pondération des termes de la requête et des noeuds feuilles utilisée pour calculer le score de pertinence des noeuds feuilles (équation 4.1) ;

$$RSV_m(q, nf) = \sum_{i=1}^{T} w_i^q * w_i^{nf} \tag{4.1}$$

- le paramètre α dans la fonction de propagation, qui modélise l'importance de la distance entre les noeuds dans la propagation (équation 4.2) ;

$$p_n = |F_n^p|. \sum_{nf_k \in F_n} \alpha^{dist(n,nf_k)-1} * RSV_m(q, nf_k) \tag{4.2}$$

- la fonction utilisée pour introduire la dimension d'informativité des noeuds, notamment en étudiant l'impact de la longueur des éléments ;
- l'introduction du contexte des éléments dans le calcul de la dimension informativité.

4.4.1 Evaluation de la formule de pondération des termes utilisée pour le calcul du score des noeuds feuilles

Nous nous proposons d'évaluer ici les formules de pondération des termes utilisée pour le calcul du score des noeuds feuilles (équation 4.1). Ces formules sont dérivées de formules utilisées dans le cadre de la RI traditionnelle. Ces dernières sont transformées afin de s'adapter à une nouvelle granularité de l'information, et elles utilisent ou non la taille des noeuds feuilles pour calculer leur similarité à la requête.
Afin de vérifier la nécessité de s'adapter à une nouvelle granularité de l'information, la première fonction que nous testons pour la pondération des termes est la fonction *tf*idf*, couramment utilisée en RI. On a alors :

$$\begin{aligned} w_i^q &= tf_i^q * idf_i \\ w_i^{nf} &= tf_i^{nf} * idf_i \end{aligned} \tag{4.3}$$

où tf_i^q et tf_i^{nf} sont respectivement la fréquence du terme i dans la requête q et le noeud feuille nf et $idf_i = log(|D|/(|d_i|+1))+1$, avec $|D|$ le nombre total de documents dans la collection et $|d_i|$ le nombre de documents contenant i.

Ces formules sont ensuite adaptées pour tenir compte de la nouvelle granularité de l'information que nous traitons (on ne parle plus de documents mais de noeuds feuilles). Nous utilisons la notion d'*ief* (*Inverse Element Frequency*), comme défini dans l'équation 4.4 :

$$ief_i = log(\frac{|F_c|}{|nf_i|+1}) + 1 \tag{4.4}$$

où $|nf_i|$ est le nombre de noeuds feuilles contenant le terme i et $|F_c|$ le nombre total de noeuds feuilles.
Les formules de pondération des termes sont alors les suivantes :

$$\begin{aligned} w_i^q &= tf_i^q * ief_i \\ w_i^{nf} &= tf_i^{nf} * ief_i \end{aligned} \tag{4.5}$$

Notons que si la requête est composée d'une expression $e =$ "$t_1..t_n$" , les formules de pondération deviennent alors :

$$\begin{aligned} w_i^q &= tf_i^q * ief_e \\ w_i^{nf} &= tf_i^{nf} * ief_e \end{aligned} \tag{4.6}$$

avec $ief_e = log(\frac{|F_c|}{|nf_e|+1}) + 1$, où $|nf_e|$ est le nombre de noeuds feuilles contenant l'expression e et $|Fc|$ le nombre total de noeuds feuilles de la collection.

La troisième formule que nous nous proposons d'évaluer est une adaptation de la formule BM25 d'Okapi [167, 194, 197]. Cette formule tient compte de la taille des noeuds feuilles pour l'évaluation de leur pertinence, comme le montre l'équation 4.8 :

$$w_i^q = tf_i^q \tag{4.7}$$

$$w_i^{nf} = log(\frac{|F_c| - |nf_i| + 0.5}{|nf_i| + 0.5}) * \frac{(k_1+1)tf_i^{nf}}{K + tf_i^{nf}}) \tag{4.8}$$

où $|F_c|$ est le nombre total de noeuds feuilles dans la collection, $|nf_i|$ est le nombre de noeuds feuilles contenant le terme i, tf_i est la fréquence du terme i dans le noeud feuille nf, $K = k_1 * ((1-b)+b*l)/\Delta l$, avec $k_1 = 1.2$ et $b = 0.75$, l est le nombre de termes dans nf et Δl est la taille moyenne des noeuds feuilles de la collection.

Les résultats présentés dans les tableaux 4.2 et 4.3 ont été obtenus en utilisant $\alpha = 1$ dans la formule de propagation (équation 4.2). Le but est en effet

d'évaluer l'impact de la formule utilisée pour le calcul du poids des termes d'indexation, et non d'évaluer la fonction de propagation. Pour obtenir le score des noeuds internes, les scores des noeuds feuilles sont donc simplements sommés. *On notera cependant que nous obtenons des résultats similaires avec d'autres valeurs d'*α.

	sog	s3_e321	e3_s321	s	avg
tf-idf	0.0884	0.0820	0.1692	0.1242	0.1143
tf-ief	0.0873	0.0817	0.1720	0.1306	0.1155
BM25	0.0726	0.0686	0.1423	0.1230	0.0995

TAB. 4.2 – Précisions moyennes pour le jeu de requêtes 2003 en faisant varier la fonction utilisée pour le calcul du poids des noeuds feuilles

	sog	s3_e321	e3_s321	s	avg
tf-idf	0.0537	0.0431	0.1704	0.1341	0.0988
tf-ief	0.0464	0.0366	0.1483	0.1070	0.0849
BM25	0.0362	0.0282	0.1488	0.1055	0.0788

TAB. 4.3 – Précisions moyennes pour le jeu de requêtes 2004 en faisant varier la fonction utilisée pour le calcul du poids des noeuds feuilles

On observe une perte d'environ 25% de la précision par rapport aux formules *tf-idf* et *tf-ief* lorsque la formule du BM25 est utilisée. Cette perte de précision peut être observée pour les deux niveaux d'exhaustivité et de spécificité. Ces résultats peuvent être expliqués par le fait que la formule du BM25, en introduisant la taille des noeuds feuilles dans le calcul du poids, privilégie d'avantage les noeuds de petite taille, ce qui ne devrait pas être le cas (ces noeuds ne sont en effet pas porteurs d'information). De plus, les valeurs de paramètres que nous utilisons sont optimales dans le cas de documents [197], mais ne le sont pas forcément dans le cas d'éléments de granularités variées. D'autres expérimentations seraient donc nécessaires pour trouver les valeurs optimales de ces paramètres dans le cadre de notre méthode de propagation de la pertinence.
La formule *tf-idf* donne de meilleurs résultats que la formule *tf-ief* pour les fonctions d'agrégation moyenne et orientées spécificité sur la campagne d'évaluation 2003, et pour toutes les fonctions d'agrégation sur la campagne d'évaluation 2004. Ceci tend à prouver que le document doit être pris en compte d'une manière ou d'une autre dans l'évaluation de la pertinence des noeuds. Dans la suite des expérimentations, nous utilisons cependant *tf-ief* comme fonction de pondération des termes, car la formule semble mieux adaptée à la granularité de l'information que nous traitons, à savoir les noeuds feuilles. Le poids du document sera introduit ultérieurement.

4.4.2 Impact du paramètre distance dans la fonction de propagation

Afin d'évaluer l'impact du paramètre distance dans la fonction de propagation (équation 4.2) sur l'exhaustivité et la spécificité, nous faisons varier la valeur de α de *0.5* (la distance entre les noeuds a beaucoup d'importance) à *1* (la distance n'a pas d'importance). Le calcul du poids des noeuds feuilles est effectué selon l'équation 4.5 présentée dans la section précedente, formule obtenant les meilleurs résultats quelle que soit la valeur de α.

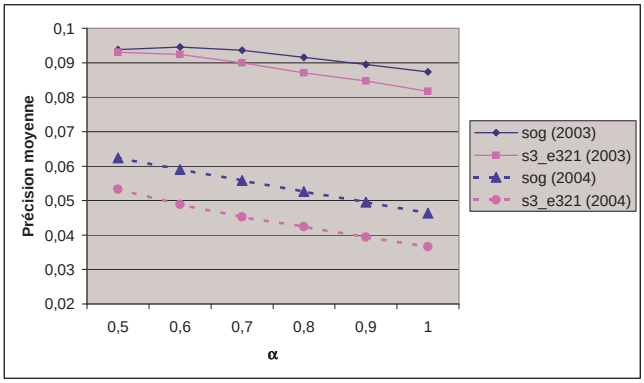


FIG. 4.2 – Evolution de la précision moyenne en fonction d'α, fonctions d'agrégation orientées spécificité

La figure 4.2 montre l'évolution de la précision moyenne en fonction de α en utilisant les fonctions d'agrégation orientées spécificité (*sog* et *s3_e321*) sur les requêtes 2003 et 2004. La première remarque que nous pouvons faire et que pour les deux mesures et pour les deux jeux de requêtes, les performances décroissent quand α augmente. En effet, plus α est petit, plus la distance entre les noeuds joue un rôle important dans la fonction de propagation, et plus le poids des noeuds feuilles est diminué dans la propagation. Par conséquent, les petits noeuds sont préférés aux plus grands, et la spécificité des noeuds résultats est plus élevée.

Contrairement à la spécificité, l'exhaustivité tend à évoluer dans le même sens que α. La figure 4.3 illustre cette tendance, en indiquant les précisions moyennes obtenues avec la fonction d'agrégation *e3_s321* pour les jeux de requêtes 2003 et 2004. Lorsque α prend des valeurs élevées, la fonction de propagation tend à être équivalente à une simple somme des poids de pertinence des noeuds feuilles. Par conséquent, les noeuds les plus hauts dans la structure des documents (c'est à dire les noeuds près du noeud *racine* ou le noeud *racine* lui-même) ont un poids de pertinence plus élevé et sont ainsi mieux classés que les noeuds situés plus profondémment dans la structure (ils ont en effet un plus grand nombre de descendants). Comme les noeuds les plus

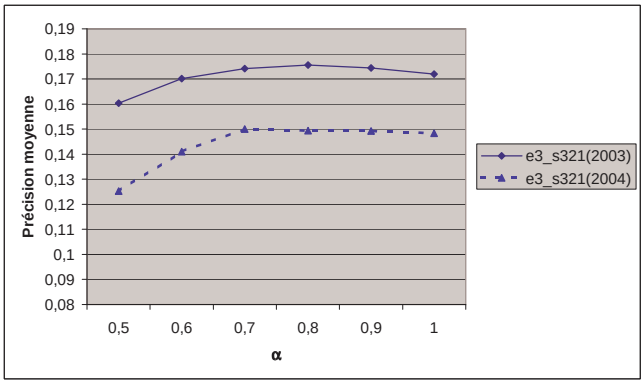


FIG. 4.3 – Evolution de la précision moyenne en fonction d'α, fonction d'agrégation orientée exhaustivité

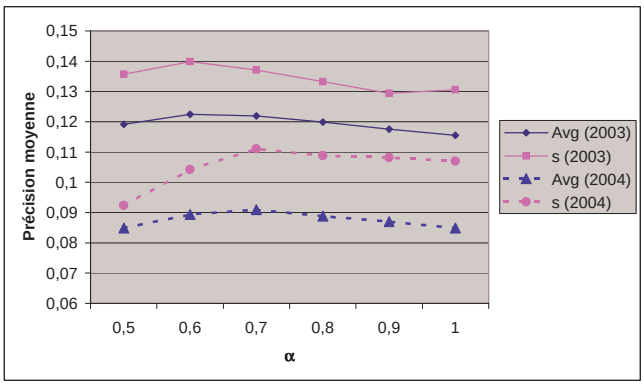


FIG. 4.4 – Evolution générale de la précision moyenne en fonction d'α

hauts dans la hiérarchie sont aussi les plus grands, le critère d'exhaustivité sera plus probablement observé.

La figure 4.4 montre l'évolution générale de la précision en fonction de α. Pour les deux jeux de requêtes, $\alpha \in [0.6, 0.7]$ semble être une plage de valeurs optimales pour obtenir le meilleur compromis entre exhaustivité et spécificité. On remarque cependant que les résultats obtenus pour $\alpha = 1$ sont encore relativement bons, ce qui est surprenant, puisque le critère de spécificité n'est pas du tout vérifié. *Des expérimentations ont également été effectuées pour des valeurs de α comprises entre 0.1 et 0.4, mais les précisions moyennes obtenues sont moins bonnes que celles présentées ici, et ce pour toutes les fonctions d'agrégation.*

Enfin, d'une manière générale, les précisions moyennes suivent la même tendance sur les jeux de requêtes 2003 et 2004. Cependant, les résultats sont

meilleurs sur le jeu de requêtes 2003, ce qui est relativement surprenant. Ceci peut être en partie expliqué par les jugements de pertinence utilisés. On trouvera une discussion sur ce sujet dans la section 4.4.6.
Notons pour conclure cette section que le paramètre $|F_n^p|$ de la fonction de propagation (équation 4.2) a lui aussi été évalué : les résultats obtenus en sa présence sont supérieurs aux résultats obtenus en son absence, comme le montre le tableau 4.4.

	$\alpha = 0.5$	$\alpha = 0.6$	$\alpha = 0.7$	$\alpha = 0.8$	$\alpha = 0.9$	$\alpha = 1$
Sans $\|F_n^p\|$	0.0970	0.1109	0.1178	0.1115	0.1108	0.1079
Avec $\|F_n^p\|$	0.1191	0.1225	0.1219	0.1199	0.1175	0.1155

TAB. 4.4 – Impact du paramètre $|F_n^p|$ dans la fonction de propagation, jeu de requêtes 2003, fonction d'agrégation moyenne (Avg)

4.4.3 Evaluation de la dimension d'informativité : Impact de la longueur des éléments

Comme nous l'avons vu au chapitre 3, la façon dont la pertinence des noeuds est calculée avec la fonction de propagation de l'équation 4.2 renverrait un noeud contenant les seuls termes de la requête comme réponse idéale. Un tel noeud ne contient cependant pas assez d'informations pour satisfaire le besoin de l'utilisateur. Dans cette section, nous nous proposons d'évaluer la notion de dimension d'informativité d'un noeud, définie au chapitre 3. Sur chaque noeud, un score de pertinence prenant en compte la dimension d'informativité est calculé.
Il semble intuitif que la notion d'informativité fasse intervenir la longueur du noeud (c'est à dire le nombre de termes qu'il contient), mais tout le problème est de savoir comment et où introduire ce paramètre. Comme nous l'avons vu dans le paragraphe précédent, l'utilisation de la longueur des éléments au niveau des noeuds feuilles ne semble pas être utile. Les expérimentations que nous présentons ici visent à introduire la longueur des éléments une fois la propagation effectuée ou alors encore pendant la propagation.

4.4.3.1 Introduction d'un seuil

Afin d'éliminer les noeuds de petite taille et donc les noeuds non-informatifs, une première solution simple est de mettre un seuil sur le nombre de termes que doit contenir un noeud pour être renvoyé par le système. La formule 4.2 est alors redéfinie comme suit :
Soient un noeud n et $nf_i \in [1..N]$ l'ensemble de ses noeuds feuilles descendants

ayant un score de similarité à la requête non nul. Soit l_i la taille du noeud feuille nf_i (c'est à dire le nombre de termes qu'il contient) et L la somme des tailles des nf_i. Si L est plus petit qu'un certain seuil x, alors le noeud n est considéré comme non informatif.

$$p_n = \begin{cases} \sum_{nf_k \in F_n} \alpha^{dist(n,nf_k)-1} * RSV_m(q, nf_k) \ si \ L > x \\ 0 \ sinon \end{cases} \tag{4.9}$$

$$avec \ L = \sum_{i=1..N} l_i, \ \forall i / \ RSV(q, nf_i) > 0 \tag{4.10}$$

La figure 4.5 montre les résultats obtenus en utilisant deux seuils $x = 25$ ou 50, ces valeurs correspondant de manière intuitive au nombre de mots minimum que doit contenir un noeud pour être porteur d'information. Les expérimentations présentées ci-dessous ont été effectuées avec l'équation 4.5 pour le calcul du poids des noeuds feuilles (*tf-ief*) et $\alpha = 0.6$ pour la propagation, sur le jeu de requêtes de la campagne 2003.

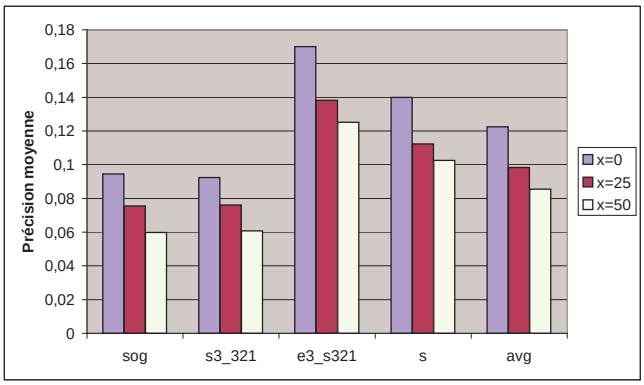


FIG. 4.5 – Evolution de toutes les mesures en utilisant un seuil sur la longueur, jeu de test d'INEX 2003

On observe une perte de performance (pour toutes les fonctions d'agrégation) lorsque le seuil x est utilisé. Des résultats similaires sont obtenus avec des valeurs plus petites de x (5 et 10) et sur le jeu de requêtes de la campagne 2004. Ces résultats peuvent être expliqués de deux façons différentes. Tout d'abord, l'utilisation du seuil x peut empêcher le système de renvoyer certains noeuds contenant des noeuds feuilles de petite taille et ayant un score de similarité à la requête non nul. Par exemple, un noeud *section* contenant de nombreux noeuds feuilles mais seulement un, de petite taille, avec un score de similarité à la requête non nul, sera considéré comme non pertinent. Pourtant, si le noeud feuille en question est un noeud *titre*, la *section* aurait probablement été pertinente et informative pour l'utilisateur.
En outre, ces résultats peuvent aussi être principalement expliqués par le fait que des noeuds de (très) petite taille (comme des noeuds *titre* ou *référence*

par exemple) ont été jugés pertinents par certains participants d'INEX, qui ont considéré que même s'ils n'apportent pas d'information à l'utilisateur, leur similarité à la requête est grande. Pour certaines requêtes, nous avons jusqu'à 85% de perte de précision en utilisant le seuil x.
On notera enfin que des résultats similaires sont obtenus en utilisant des seuils sur la longueur totale des noeuds (c'est à dire sur la somme des longueurs de tous leurs noeuds feuilles).
Pourtant, du point de vue de l'utilisateur, les noeuds de petite taille devraient être moins bien classés par le SRI. C'est ce que nous nous proposons d'évaluer dans la section suivante.

4.4.3.2 Utilisation de la longueur médiane/moyenne

Les évaluations présentées ici cherchent à répondre à la question suivante : la pertinence des éléments est-elle liée à leur longueur ? En d'autres termes, il y a-t-il une taille d'élément pour laquelle ces derniers ont de plus fortes probabilités d'être pertinents ?
Nous nous proposons donc d'utiliser des longueurs *moyenne* et *médiane* des noeuds pertinents dans le calcul de la pertinence d'un noeud. De manière intuitive, on peut penser que plus un élément possède une taille éloignée de la longueur moyenne ou médiane d'un élément pertinent, plus la probabilité qu'il soit informatif est faible. Cette intuition est formalisée de la façon suivante :

$$p_n = \frac{1}{log(|\Delta l - l| + 1) + 1} |F_n^p| . \sum_{nf_k \in F_n} \alpha^{dist(n,nf_k)-1} * RSV_m(q, nf_k) \qquad (4.11)$$

et

$$p_n = \frac{1}{log(|\phi l - l| + 1) + 1} |F_n^p| . \sum_{nf_k \in F_n} \alpha^{dist(n,nf_k)-1} * RSV_m(q, nf_k) \qquad (4.12)$$

où Δl et ϕl sont respectivement la longueur moyenne et médiane d'un élément pertinent. Ces valeurs sont respectivement de 1010 et 226 pour le jeu de requêtes 2003 [104].

Les résultats présentés figures 4.6 et 4.7 sont obtenus en utilisant les équations 4.11 et 4.12 sur les jeux de requêtes 2003 et 2004. Comme pour les expérimentations du paragraphe précédent, nous avons utilisé l'équation 4.5 pour le calcul du poids des noeuds feuilles (*tf-ief*) et $\alpha = 0.6$ dans la fonction de propagation (équation 4.2).
L'introduction de la moyenne de la longueur des éléments pertinents a un effet négatif sur toutes les fonctions d'agrégation, alors que ce n'est pas forcément le cas pour la médiane. L'introduction de la longueur médiane des éléments pertinents a un double effet ; d'un côté, la précision globale et l'exhaustivité

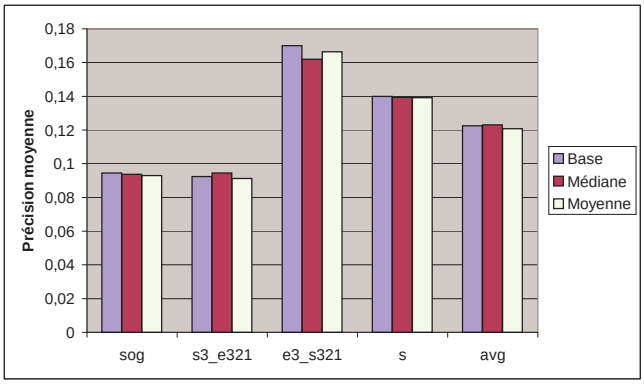


FIG. 4.6 – Evolution de toutes les mesures en utilisant les longueurs moyenne et médiane, jeu de test d'INEX 2003

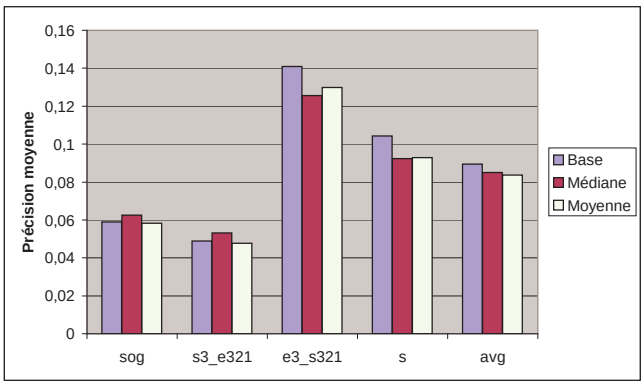


FIG. 4.7 – Evolution de toutes les mesures en utilisant les longueurs moyenne et médiane, jeu de test d'INEX 2004

décroissent, alors que de l'autre, on observe jusqu'à 8% d'augmentation sur la spécificité (principalement sur la fonction d'agrégation *s3_e321* et sur le jeu de requêtes 2003). Il semble ainsi que la dimension de spécificité est liée à la longueur des éléments, et que la longueur des éléments peut être un paramètre utile dans certains cas particuliers.

4.4.3.3 Evaluation de la propagation pondérée par la taille des noeuds feuilles

Comme nous venons de le voir, il est souhaitable pour l'utilisateur que les noeuds de trop petite taille ou trop grande taille soient moins bien classés par le SRI. Ceci n'implique cependant pas qu'ils ne sont d'aucune utilité.
De manière intuitive, on peut penser que le concepteur d'un document utilise les noeuds de petite taille pour faire ressortir des informations importantes.

Ils peuvent ainsi donner des indications précieuses sur la pertinence de leurs noeuds ancêtres. Les expérimentations présentées dans cette section cherchent à vérifier cette affirmation.
Afin de modéliser les importances diverses des noeuds feuilles durant la propagation, nous introduisons le paramètre $\beta(n_k)$:

$$p_n = |F_n^p|. \sum_{nf_k \in F_n} \alpha^{dist(anc,nf_k)-1} * \beta(nf_k) * RSV(q, nf_k) \tag{4.13}$$

D'après nos expérimentations, la valeur optimale de $\beta(n_k)$ est la suivante :

$$\beta(nf_k) = \begin{cases} l_k/\Delta l \ si \ dist(n, nf_k) = 1 \ et \ l_k < \Delta l \\ log(\Delta l/l_k) \ si \ dist(n, nf_k) > 1 \ et \ l_k < \Delta l \\ 1 \ sinon \end{cases} \tag{4.14}$$

avec l_k la taille du noeud feuille nf_k et Δl la taille moyenne des noeuds feuilles de la collection.
Cette valeur peut être traduite de la façon suivante :

- Si un noeud feuille nf_k est de petite taille (c'est à dire de taille inférieure à la moyenne) la pertinence p_{par} de son noeud parent par doit être faible :

$$Si \ l_k < \Delta l \ et \ dist(n, nf_k) = 1 \ alors$$
$$p_{par} = \frac{l_k}{\Delta l} * \sum_{nf_k \in F_n} RSV_m(q, nf_k) \tag{4.15}$$

- Mais son score de similarité à la requête doit augmenter l'informativité de ses noeuds ancêtres anc :

$$p_{anc} = |F_n^p|. \sum_{nf_k \in F_n} \alpha^{dist(anc,nf_k)-1} * log(\frac{\Delta l}{l_k}) * RSV(q, nf_k) \tag{4.16}$$

De manière synthétique, la dimension d'informativité d'un noeud n est incluse de la façon suivante dans le calcul de la pertinence : les résultats obtenus avec ces nouvelles formules (qui ont été ajustées par expérimentations) sont décrits dans le tableau 4.5. Nous avons utilisé l'équation 4.5 pour le calcul du poids des noeuds feuilles (tf-ief) et fixé $\alpha = 0.6$ dans la fonction 4.13.

En ce qui concerne le jeu de test 2003, les résultats obtenus montrent une légère amélioration des performances sur toutes les fonctions d'agrégation[1]. Cette amélioration n'est cependant pas réellement significative et n'est pas confirmée sur le jeu de test 2004.
Pourtant, malgré ces résultats, la modélisation des importances diverses que peuvent prendre les noeuds feuilles dans la propagation nous paraît avoir un

[1]Ces résultats sont légèrements différents de ceux publiés dans [183], la façon de calculer le poids des expressions au niveau des noeuds feuilles ayant été modifiée et le paramètre $|F_n^p|$ ayant été rajouté

		sog	s3_e321	e3_s321	s	avg
2003	*Base (équation 4.2)*	0.0946	0.0924	0.1701	0.1399	0.1225
	Informativité (équation 4.13)	0.0962	0.0937	0.1728	0.1408	0.1241
	Gain	+1.7%	+1.4%	+1.6%	+0.7%	+1.3%
2004	*Base (équation 4.2)*	0.0590	0.0489	0.1410	0.1042	0.0894
	Informativité (équation 4.13)	0.0588	0.0489	0.1408	0.1033	0.0890
	Gain	-0.4%	0%	-0.1%	-0.9%	-0.5%

TAB. 4.5 – Comparaison des précisions moyennes obtenues par calcul de pertinence et calcul de similarité (utilisation des éléments descendants) sur les jeux de requêtes 2003 et 2004

intérêt non négligeable dans le calcul de l'informativité des noeuds, et nous nous proposons de garder ces dernières formules de propagation (équation 4.13) pour le calcul de l'informativité dans notre modèle. Les résultats présentés dans la section 4.4.5 confirment l'intérêt de notre choix.

Les expérimentations que nous avons présentées dans cette section ne permettent pas de tirer des conclusions définitives sur l'impact de la taille des noeuds dans le calcul de leur informativité : celle-ci semble cependant utile, et les résultats que nous obtenons diffèrent légèrement entre les jeux de test des campagnes 2003 et 2004, et soulèvent des problèmes au niveau des jugements de pertinence utilisés. Ceci est discuté dans la section 4.4.7.

4.4.4 Evaluation de la dimension d'informativité : impact du contexte des éléments

Dans le paragraphe précédent, nous avons introduit la notion d'informativité d'un noeud, qui cherche à prendre en compte la taille de l'élément ainsi que l'importance variable de ses noeuds feuilles descendants. Dans cette section, nous nous proposons d'évaluer l'impact de la pertinence du document dans son ensemble sur la pertinence des éléments qu'il contient. Les expérimentations présentées dans la section 2.4.1 laissent en effet entre-apercevoir que le contexte des éléments joue un rôle non négligeable dans l'évaluation de leur pertinence. De manière intuitive, cette idée est facilement explicable : le concepteur d'un document suit une certaine unité dans ses idées, même si le contenu du document est hétérogène. La pertinence des unités d'informations du document est alors liée à la pertinence de cette unité de pensée à la requête. Dans le cadre de notre modèle, on parlera de *pertinence contextuelle*, calculée grâce à la *retro-propagation* de la pertinence du noeud racine (c'est à dire du document) vers les noeuds internes.

4.4.4.1 Pertinence contextuelle

Les expérimentations présentées dans cette section ont pour but d'évaluer l'impact de la rétro-propagation (c'est à dire la propagation du haut vers le bas) du poids de l'élément racine du document vers ses descendants.
Pour ce faire, nous nous proposons de modifier le calcul de la dimension d'informativité d'un noeud n comme présenté dans l'équation 4.17, inspirée des travaux présentés dans [136] :

$$p_n = \rho * |F_n^p|. \sum_{nf_k \in F_n} \alpha^{dist(n,nf_k)-1} * RSV_m(q, nf_k) + (1-\rho) * p_{racine} \qquad (4.17)$$

avec p_{racine} la pertinence du noeud *racine* du document, calculée d'après l'équation 4.2 . $\rho \in [0..1]$ est un paramètre servant de pivot et permettant d'ajuster l'importance de la pertinence du noeud racine lors de la rétro-propagation.
Les résultats que nous présentons ici ont été obtenus en fixant $\alpha = 0.6$ dans l'équation 4.2 et en utilisant l'équation 4.1 (*tf-ief*) pour le calcul du poids des noeuds feuilles.

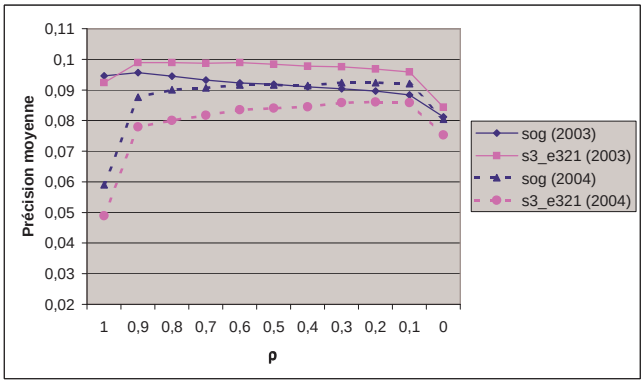


FIG. 4.8 – Evolution de la précision moyenne en fonction de ρ, fonctions d'agrégation orientées spécificité

La figure 4.8 montre l'évolution de la précision moyenne en fonction de ρ et en utilisant les fonctions d'agrégation orientées spécificité (*sog* et *s3_e321*) sur les requêtes 2003 et 2004. La première remarque que nous pouvons faire est que pour les deux mesures et pour les deux jeux de requêtes, la précision moyenne augmente jusqu'à 50% lorsque le score d'informativité tient compte du poids de pertinence du noeud racine ($0 < \rho < 1$) par rapport à la seule prise en compte du poids de pertinence des éléments ($\rho = 1$). On peut donc conclure à partir de ces résultats que l'introduction du pivot ρ dans le calcul de l'informativité et donc du contexte des éléments dans le calcul de leur informativité augmente les performances en ce qui concerne leur spécificité. Lorsque $\rho = 0$,

seule la pertinence de l'élément racine est prise en compte pour le calcul de l'informativité d'un noeud, ce qui entraine logiquement une baisse notable des précisions moyennes.

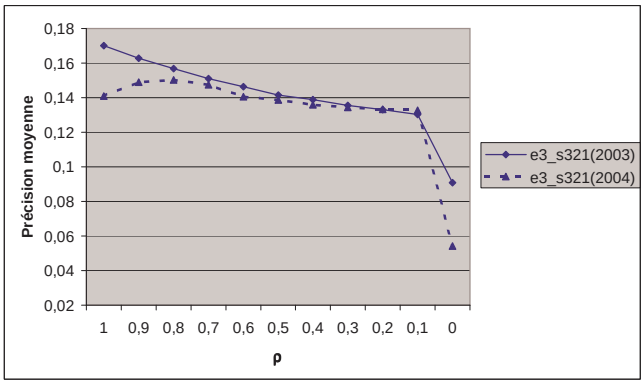


FIG. 4.9 – Evolution de la précision moyenne en fonction de ρ, fonction d'agrégation orientée exhaustivité

Contrairement à la spécificité, on observe pour l'exhaustivité des courbes d'allure différentes pour les jeux de requêtes 2003 et 2004 (figure 4.9). De manière surprenante, l'exhaustivité décroit parallèlement à ρ sur le jeu de requêtes 2003, alors que pour le jeu de requêtes 2004, l'exhaustivité suit un comportement analogue à la spécificité : une amélioration notable des performances est observée pour certaines valeurs de ρ.

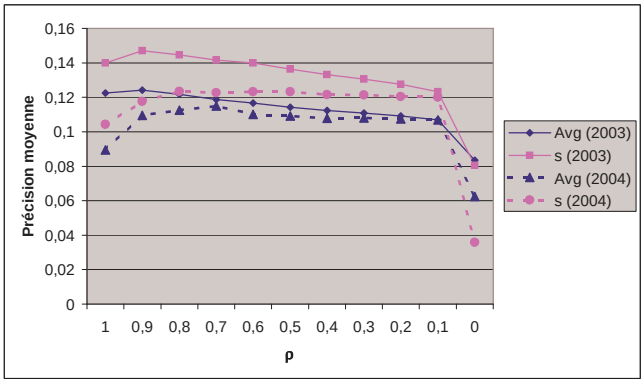


FIG. 4.10 – Evolution globale de la précision moyenne en fonction de ρ

La figure 4.10 montre l'évolution générale de la précision en fonction de ρ. Pour le jeu de requêtes 2003, $\rho = 0.9$ semble être le meilleur compromis entre exhaustivité et spécificité, alors que pour le jeu de requêtes 2004, une valeur de 0.8 semblerait plus appropriée. Quoiqu'il en soit, on observe de manière nette sur les courbes une augmentation de la précision moyenne et donc de la performance lorsque le contexte des éléments est utilisé pour calculer leur informativité (jusqu'à plus de 30% d'augmentation pour le jeu de requêtes 2004).

Ce contexte ne doit cependant pas avoir une place trop importante dans le calcul de ce score, les valeurs optimum de ρ pour les deux jeux de requêtes étant relativement élevées.

4.4.4.2 Tri des éléments en fonction du poids du document

Dans les expérimentations que nous avons présentées jusqu'ici, les unités d'informations étaient triées indépendamment les unes des autres, en fonction de leur score de pertinence ou bien de leur score d'informativité. Nous avons montré ci-dessus que le contexte des éléments était un paramètre important pour le calcul de leur informativité. Nous nous proposons donc d'étendre l'étude de l'impact de ce contexte de la manière suivante :

1. nous calculons un score de pertinence pour tous les documents de la collection,
2. nous calculons un score de pertinence pour tous les éléments de la collection,
3. nous trions les documents par ordre décroissant de pertinence,
4. pour chaque document, nous trions par ordre décroissant de pertinence les éléments qu'il contient.

De cette façon, les éléments sont d'abord triés en fonction de la pertinence du document auquel ils appartiennent puis en fonction de leur propre pertinence. Dans les expérimentations que nous présentons ci-dessous, le poids de pertinence des documents est calculé de deux manières différentes :

- en utilisant la simple formule *tf*idf*,
- par propagation en utilisant $\alpha = 0.6$ dans la fonction de propagation (équation 4.2)

Les résultats présentés dans le tableau 4.6 ont été obtenus en calculant la pertinence des unités d'information avec l'équation 4.2 et $\alpha = 0.6$ et en utilisant la formule 4.1 (*tf-ief*) pour le calcul du poids des noeuds feuilles. Ces résultats (en calculant le poids du document avec *tf-idf* ou par propagation) sont comparés avec les résultats obtenus en triant simplement les éléments selon leur pertinence (équations 4.1 et 4.2 avec $\alpha = 0.6$). Alors que pour le jeu de requêtes 2003 les performances se dégradent lorsque l'on effectue un premier tri sur la pertinence du documents, on observe l'effet inverse sur le jeu de requêtes 2004. Cette observation rejoint les résultats présentés dans la section 4.4.1 et 4.4.4.1 : la modélisation de l'importance du document dans la collection semble être nécessaire pour répondre aux attentes des utilisateurs de la campagne 2004. On observe en effet jusqu'à 40% d'augmentation de la précision moyenne lorsqu'un premier tri est effectué sur le poids des documents calculé par *tf-idf*. Ces observations nous poussent à approfondir notre réflexion sur les jugements de pertinence que nous utilisons (section 4.4.7).

		sog	s3_e32	e3_s32	s	avg
2003	*Base (équation 4.2)*	0.0946	0.0924	0.1701	0.1399	0.1225
	tf-idf sur document	0.0873	0.0945	0.1221	0.1328	0.1066
	Propagation sur document	0.0873	0.0950	0.1253	0.1125	0.1033
2004	*Base (équation 4.2)*	0.0590	0.0489	0.1410	0.1042	0.0894
	tf-idf sur document	0.0958	0.0889	0.1552	0.1462	0.1204
	Propagation sur document	0.0921	0.0859	0.1333	0.1201	0.1073

TAB. 4.6 – Comparaison des précisions moyennes obtenues par tri sur la pertinence des éléments ou tri sur la pertinence des documents puis des éléments

4.4.5 Evaluation de la combinaison *propagation pondérée par la taille des noeuds feuilles / pertinence contextuelle*

Même si les résultats que nous obtenons sur les jeux de test 2003 et 2004 ne sont pas toujours comparables, nous avons montré ci-dessus que les noeuds descendants et ancêtres d'un noeud donné jouaient un rôle prépondérant pour le calcul de sa dimension d'informativité. Nous avons évalué l'intérêt de ces deux propositions prises séparément, et nous nous proposons ici d'évaluer leur combinaison.
L'informativité d'un noeud n est alors calculé selon la formule suivante :

$$\begin{aligned} p_n &= \rho * |F_n^p|. \sum_{nf_k \in F_n} \alpha^{dist(n,nf_k)-1} * \beta(nf_k) * RSV(q, nf_k) \\ &\quad +(1-\rho) * |F^p|. \sum_{nf_k \in F} \alpha^{dist(racine,nf_k)-1} * \beta(nf_k) * RSV(q, nf_k) \end{aligned} \quad (4.18)$$

avec F_n et F respectivement l'ensemble des noeuds feuilles nf_k descendants de n et l'ensemble des noeuds feuilles nf_k du document, $|F_n^p|$ et $|F^p|$ respectivement le nombre de noeuds feuilles ayant un score non nul descendant de n ou du document, $RSV(q, nf_k)$ calculé d'après 4.5 et

$$\beta(nf_k) = \begin{cases} l_k/\Delta l \ si \ dist(n, nf_k) = 1 \ et \ l_k < \Delta l \\ log(\Delta l/l_k) \ si \ dist(n, nf_k) > 1 \ et \ l_k < \Delta l \\ 1 \ sinon \end{cases} \quad (4.19)$$

Cette formule combine en fait une *propagation pondérée* des poids des noeuds feuilles et une *rétropropagation* des poids des documents pour obtenir la pertinence p_n d'un noeud n.

Dans les expérimentations présentées ci-dessous, nous avons fixé $\alpha = 0.6$ et $\rho = 0.9$ pour le jeu de test 2003 et $\rho = 0.8$ pour le jeu de test 2004.

		sog	s3_e321	e3_s321	s	avg
2003	Base	0.0946	0.0924	0.1701	0.1399	0.1225
	Propagation pondérée	0.0952	0.0937	0.1728	0.1408	0.1241
	Rétropropagation	0.0957	0.0989	0.1628	0.1471	0.1242
	Propagation pond.+rétropropagation	0.0990	0.1021	0.1667	0.1515	0.1280
2004	Base	0.0590	0.0489	0.1410	0.1042	0.0894
	Propagation pondérée	0.0588	0.0488	0.1408	0.1033	0.0890
	Rétropropagation	0.0901	0.0800	0.1502	0.1235	0.1125
	Propagation pond.+rétropropagation	0.0905	0.0808	0.1508	0.1236	0.1128

TAB. 4.7 – Apport de la combinaison propagation pondérée et rétropropagation sur les jeux de test INEX 2003 et 2004

Le tableau 4.7 montre que par rapport à une simple propagation, les précisions moyennes augmentent d'environ 4.5% pour le jeu de test 2003 et de plus de 26% pour le jeu de test 2004 (et ce particulièrement grâce à la prise en compte du contexte) quand la propagation pondérée et la rétropropagation sont combinées. La combinaison des deux méthodes permet de plus d'obtenir de meilleurs résultats que ceux obtenus en utilisant les deux méthodes séparément.

4.4.6 Le problème des jugements de pertinence

Les analyses présentées dans [152, 151] montrent que les éléments jugés pertinents par les participants de la campagne d'évaluation 2004 peuvent être divisés en deux grandes catégories :

- les éléments *généraux*, qui correspondent à des utilisateurs préférant des réponses très informatives et éventuellement décomposables (c'est à dire des utilisateurs aimant voir les réponses proposées dans leur contexte)
- les éléments *spécifiques*, qui correspondent à des utilisateurs voulant des réponses très focalisées sur leur besoin

Ces deux catégories d'éléments pertinents correspondent à deux modèles utilisateurs différents : il devrait donc y avoir deux tâches de recherche différentes. Cette hétérogénéité dans les jugements de pertinence vient du fait qu'aucun modèle utilisateur n'est exactement défini dans le cadre d'INEX, ce qui pose de nombreux problèmes aux participants. Pour certains en effet, un noeud *titre* peut être considéré comme pertinent car ressemblant à la requête ou constituant un bon point d'entrée dans le document, alors que pour d'autres (dont nous faisons partie), il n'est en rien informatif car il n'apporte rien par rapport au besoin en information de l'utilisateur.
Cependant, certains jugements de pertinence restent difficilement explicables : on trouvera par exemple des cases de tableaux jugées très exhaustives et très

spécifiques alors que seules, elles n'ont aucun sens. Le tableau qui les contient peut pourtant être pertinent par rapport à la requête.
Une illustration supplémentaire de ce problème est apportée par les statistiques publiées dans [121]. Lors de la campagne d'évaluation 2004, un certain nombre de requêtes ont été jugées par deux participants différents. Les statistiques éditées sur ces jugements montrent que seulement 12% des jugements pertinents sont concordants entre les deux utilisateurs !

Nous avons réévalué notre modèle avec ces nouveaux jugements de pertinence, et malgré le peu de points communs entre ces jugements et les "anciens", les tendances observées sur notre modèle sont les mêmes. L'exhaustivité et la spécificité suivent les mêmes variations en fonction de nos paramètres, et les valeurs optimales de nos paramètres (*calcul du poids des noeuds feuilles*, α, β et ρ) sont identiques, ce qui tend à montrer la robustesse de notre approche.

Lors des discussions ayant eu lieu en décembre 2004 pour le Workshop INEX, il a été décidé, afin de résoudre ces problèmes, de définir plusieurs modèles utilisateurs distincts, et de proposer des mesures propres à chaque modèle utilisateur. Les tâches de recherche seront par exemple de trouver les éléments les plus spécifiques dans des chemins donnés, ou de trouver le plus de contenu pertinent possible.

4.4.7 Le problème des noeuds imbriqués

Dans les expérimentations que nous avons présentées jusqu'ici, tous les ancêtres d'un noeud ayant un score non nul ont aussi un score non nul et sont par conséquent renvoyés dans la liste des résultats. Les listes triées de résultats que nous obtenons contiennent ainsi en moyenne 80% de noeuds imbriqués.
Notre modèle a été paramétré de la sorte afin de permettre une évaluation correcte dans le cadre de la campagne d'évaluation INEX. En effet, lorsque les participants effectuent les jugements de pertinence, des règles d'inférence impliquent que lorsqu'un noeud est jugé pertinent, son noeud parent doit aussi être jugé pertinent [155] : il peut être moins spécifique, mais son exhaustivité est toujours égale ou supérieure. Par conséquent, on obtient une base de rappel très grande, composée d'éléments imbriqués. Un rappel parfait avec les mesures utilisées dans INEX ne peut être atteint que si les systèmes renvoient tous les éléments de la base de rappel, y compris des éléments imbriqués [109].
Cependant, le but de la tâche CO n'est pas de renvoyer tous les éléments pertinents quel que soit leur degré de pertinence mais plutôt de trouver les unités d'informations les plus exhaustives et spécifiques répondant à une requête donnée. Nous avons donc refait nos expérimentations en interdisant le ren-

voi de noeuds imbriqués. Pour ce faire, nous avons procédé comme suit : étant donné deux éléments dans un chemin pertinent, l'élément avec le plus grand score est sélectionné. Une fois que tous les chemins pertinents ont été traités, un filtrage final est appliqué afin d'éliminer les imbrications possibles entre les meilleurs éléments, en ne gardant pour deux noeuds imbriqués que celui ayant le meilleur score.
Les figures 4.11, 4.12 et 4.13 montrent respectivement l'évolution de la précision moyenne en fonction de α pour les fonctions d'agrégation orientées spécificité, orientées exhaustivité et générales. Pour ces expérimentations, nous avons utilisé la formule 4.5 pour le calcul du poids des noeuds feuilles et 4.2 pour la propagation.

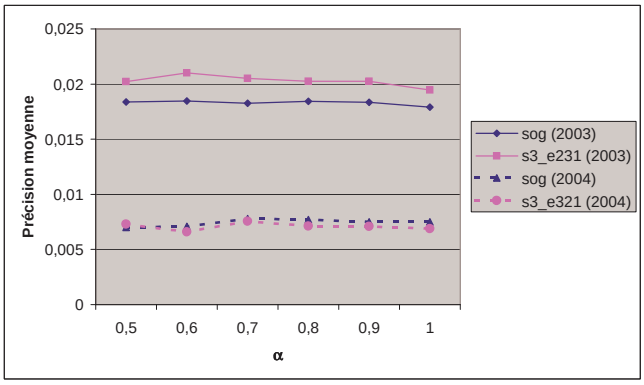


FIG. 4.11 – Evolution de la précision moyenne en fonction de α, fonctions d'agrégation orientées spécificité, aucune imbrication de noeuds autorisée

Les précisions moyennes obtenues sont beaucoup plus faibles que celles obtenues en permettant l'imbrication des noeuds, et ce à cause de la surpopulation de la base de rappel [109]. La spécificité (figure 4.11) semble être indépendante de α, ce qui n'était pas le cas dans les expérimentations présentées dans la section 4.4.2.

Les résultats concernant l'exhaustivité sont cependant comparables (figure 4.12), même si l'augmentation de la précision en fonction de α est plus marquée dans le cas des expérimentations ne permettant pas l'imbrication des noeuds.

Par conséquent, moins les scores de pertinence sont diminués pendant la propagation (α tend vers 1), plus les performances générales augmentent, ce qui n'était pas le cas pour les expérimentations permettant l'imbrication des noeuds. Cependant, l'augmentation des valeurs du paramètre α implique un bon classement des noeuds racines (particulièrement quand $\alpha = 1$), et par conséquent le critère de spécificité n'est toujours pas respecté.
Ces résultats soulèvent un important problème concernant les mesures actuellement utilisées dans INEX, plus particulièrement pour les fonctions d'agrégation orientées spécificité : seuls les éléments très spécifiques sont supposés être per-

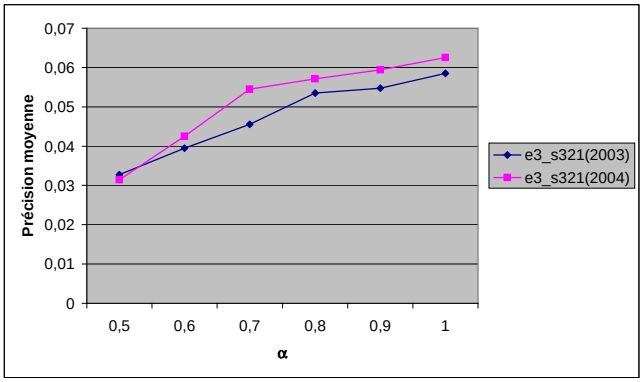


FIG. 4.12 – Evolution de la précision moyenne en fonction de α, fonction d'égrégration orientée exhaustivité, aucune imbrication de noeuds autorisée

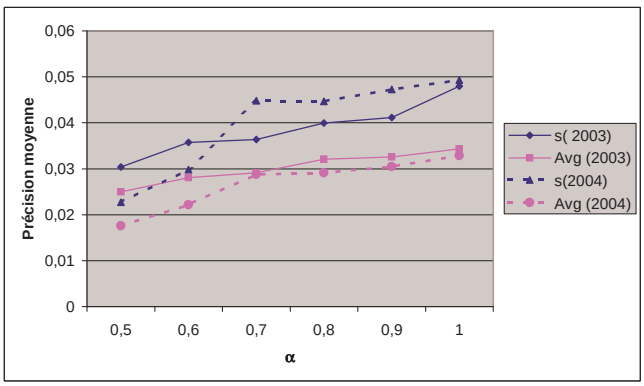


FIG. 4.13 – Evolution générale de la précision moyenne en fonction de α, aucune imbrication de noeuds autorisée

tinents alors que dans nos expérimentations, on obtient les meilleurs résultats lorsque beaucoup de noeuds racines (qui ne sont pas supposés être spécifiques d'une manière générale) sont renvoyés.
Ces résultats montrent le besoin de mesures appropriées pour évaluer des systèmes ne permettant pas l'imbrication des noeuds. La mesure XCG (XML Cumulated Gain) proposée par Gabriella Kazai dans [109] a pour but de résoudre ce problème.
Nous avons donc renouvelé nos évaluations avec cette mesure. Une première série d'expérimentations a utilisé l'équation 4.18 nous permettant d'obtenir des résultats optimaux sur les autres mesures. Nous avons fixé $\alpha = 0.6$ et fait varier le paramètre ρ, en permettant ou non au système de renvoyer des noeuds imbriqués. Les résultats que nous obtenons sont présentés dans le tableau 4.8, pour les deux fonctions d'agrégation *stricte* et *sog* [2].

[2]Ces fonctions sont les seules à notre disposition dans le programme d'évaluation XCG

		$\rho = 0.7$	$\rho = 0.8$	$\rho = 0.9$	$\rho = 1$
Sog	Noeuds imbriqués possibles	0.1949	0.1974	0.2008	0.2068
	Pas de noeuds imbriqués	0.1760	0.1761	0.1783	0.2012
Strict	Noeuds imbriqués possibles	0.2083	0.2101	0.2125	0.2157
	Pas de noeuds imbriqués	0.2058	0.2004	0.2004	0.2141

TAB. 4.8 – Résultats obtenus pour la mesure XCG en faisant varier le paramètre ρ

Une première remarque est que les résultats obtenus avec ou sans noeuds imbriqués sont comparables, avec des performances légèrement meilleures dans le cas où les noeuds imbriqués sont permis. La mesure XCG nous permet donc d'évaluer nos propositions ne permettant pas l'imbrication des noeuds de façon plus satisfaisante que les mesures actuellement utilisées dans INEX. On remarque aussi que le paramètre ρ introduisant le contexte des noeuds dans le calcul de leur pertinence provoque une baisse des performances, que ce soit pour la fonction d'agrégation *stricte* ou la fonction d'agrégation *sog* : les meilleurs résultats sont en effet obtenus pour $\rho = 1$, c'est à dire quand la pertinence contextuelle n'est pas prise en compte.
Suite à ces résultats, nous avons évalué l'impact du paramètre α (modélisant l'importance de la distance entre les noeuds dans la propagation) sur la mesure XCG. Les courbes représentées sur la figure 4.14 montrent l'évolution de la précision pour les fonctions d'agrégation stricte et généralisée. Les résultats représentés sont obtenus en ne permettant pas l'imbrication des noeuds, puisque le but est d'évaluer notre modèle dans ce cas précis. Dans ces expérimentations, on fixe $\rho = 1$.

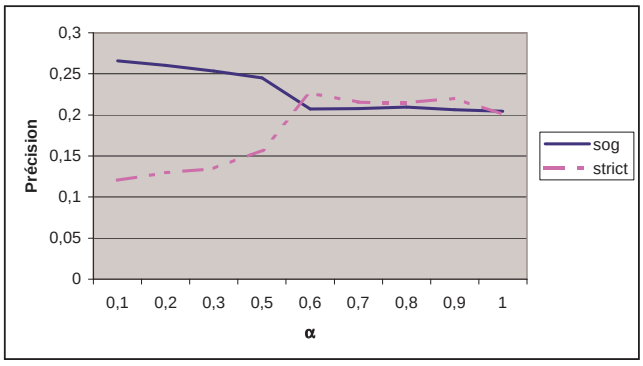


FIG. 4.14 – Evolution de la mesure XCG en fonction de α, pas de noeuds imbriqués

Les résultats obtenus pour la fonction d'agrégation *stricte* sont comparables à ceux obtenus dans les précédentes évaluations de notre modèle : la plage de valeur $\alpha \in [0.6; 0.7]$ nous permet d'obtenir des performances optimales. Les

résultats obtenus par la fonction d'égrégation *sog* sont quant à eux plus surprenants : les meilleures performances sont obtenues pour des valeurs faibles de α (c'est à dire en privilégiant les éléments les plus petits), ce qui n'était pas le cas dans les expérimentations présentées au paragraphe 4.4.2. Des expérimentations complémentaires nous paraissent donc nécessaires pour confirmer cette tendance.
Enfin et à titre de comparaison, les 10 meilleurs approches pour la mesure XCG dans la campagne INEX 2004 sont comprises 0.2228 et 0.2602 pour la fonction d'agrégation *stricte*[3] (ce qui nous permettrait de figurer dans le classement) et entre 0.2953 et 0.3725 pour la fonction d'agrégation *sog*.

4.5 Expérimentations sur les requêtes orientées contenu et structure

Afin d'évaluer notre modèle pour le traitement des requêtes orientées contenu et structure (c'est à dire le calcul du poids des noeuds feuilles et les différentes formules de propagation), nous utilisons les requêtes et jugements de pertinences associés à la tâche SCAS d'INEX 2003. La tâche VCAS dans laquelle les conditions de structure ne doivent pas être forcément respectées nous servira uniquement pour discuter de l'interprétation stricte ou vague des conditions de contenu présentes dans les requêtes.

4.5.1 Impact de la formule de pondération utilisée pour le calcul du poids des noeuds feuilles

Les formules de pondération des termes des noeuds feuilles que nous nous proposons d'évaluer ici sont comparables à celles évaluées dans le cadre des requêtes orientées contenu seulement :

- tf-idf (équation 4.3)
- tf-ief (équation 4.5)
- adaptation de la formule du BM25 à la nouvelle granularité de l'information (équation 4.8)

Les résultats présentés dans le tableau 4.9 ont été obtenus en ne tenant pas compte de la distance dans les fonctions de propagation. Le but est en effet d'évaluer l'impact de la formule utilisée pour le calcul du poids des termes dans les noeuds feuilles, et non d'évaluer les fonctions de propagation. Pour obtenir le score des noeuds résultats des sous-requêtes élémentaires $SRE_{i,j}$ les

[3]Ces évaluations ne sont pas officielles et on été menées par nos soins grâce aux contributions des différents participants.

scores des noeuds feuilles sont simplement sommés, et le score des noeuds cibles est également obtenu en ajoutant à leur score de départ le score des noeuds répondant aux conditions de hiérarchie. Les résultats que nous obtenons pour les différents fonctions d'agrégation sont présentés dans le tableau 4.9.

	sog	s3_e321	e3_s321	s	avg
tf-idf	0.2305	0.2610	0.2517	0.2621	0.2514
tf-ief	0.2323	0.2640	0.2577	0.2666	0.2552
BM25	0.2104	0.2441	0.2193	0.2276	0.2255

TAB. 4.9 – Précisions moyennes pour le jeu de requêtes CAS 2003 en faisant varier la fonction utilisée pour le calcul du poids des noeuds feuilles

Lorsque la formule du BM25 est utilisée, on observe une perte d'environ 10% de la précision par rapports aux formules *tf-idf* et *tf-ief*. Cette perte de précision peut être observée pour les deux niveaux d'exhaustivité et de spécificité. Cette formule ne parait donc pas plus adaptée au traitement des requêtes orientées contenu et structure qu'aux requêtes orientées contenu seulement.
Les résultats obtenus par les formules *tf-ief* et *tf-idf* sont comparables.
Nous nous proposons cependant de conserver dans notre modèle la formule *tf-ief* (équation 4.5), cette formule nous paraissant plus adaptée à la granularité de l'information traitée (les noeuds feuilles) et nous permettant d'obtenir des résulats sensiblement meilleurs.

4.5.2 Impact du paramètre distance dans les fonctions de propagation

Afin d'évaluer l'importance du paramètre distance séparant les noeuds dans les différentes fonctions de propagation (équations 3.14, 3.18, 3.25), plusieurs fonctions de propagations ont été évaluées.
- $F_k(RSV_m(q, nf_k), dist(n, nf_k))$ (3.14) prend respectivement les valeurs de :

$$\hookrightarrow \; F_k(RSV_m(q, nf_k), dist(n, nf_k)) = \sum_{nf_k \in F_n} \lambda * RSV(q, nf_k) \tag{4.20}$$

$$\hookrightarrow \; F_k(RSV_m(q, nf_k), dist(n, nf_k)) = \sum_{nf_k \in F_n} \alpha^{dist(n,nf_k)-1} * RSV(q, nf_k) \tag{4.21}$$

- $agreg_{ET}(p_n, p_m, , dist(l, n), dist(l, m))$ (3.18) est fixée respectivement à :

$$\hookrightarrow \; agreg_{ET}(p_n, p_m, dist(l, n), dist(l, m)) = \lambda * (p_n + p_m) \tag{4.22}$$

$$\hookrightarrow \; agreg_{ET}(p_n, p_m, dist(l, n), dist(l, m)) = \frac{p_n}{dist(l, n)} + \frac{p_m}{dist(l, m)} \tag{4.23}$$

$$\hookrightarrow \; agreg_{ET}(p_n, p_m, dist(l,n), dist(l,m)) = \alpha^{dist(l,n)} * p_n + \alpha^{dist(l,m)} * p_m \tag{4.24}$$

La fonction $agreg_{OU}$ est quant à elle une simple fonction somme (voir chapitre 3). - et finalement, $prop\_ag(dist(m,n), p_n, p_m)$ (3.25) prend respectivement les valeurs de :

$$\hookrightarrow \; prop\_ag(dist(m,n), p_n, p_m) = \lambda * p_m + p_n \tag{4.25}$$

$$\hookrightarrow \; prop\_ag(dist(m,n), p_n, p_m) = \frac{p_n + p_m}{dist(n,m)} \tag{4.26}$$

$$\hookrightarrow \; prop\_ag(dist(m,n), p_n, p_m) = \alpha^{dist(m,n)} * p_m + p_n \tag{4.27}$$

où λ et $\alpha \in]0..1]$, et *dist(x,y)* est la distance qui sépare le noeud x du noeud y dans l'arbre du document (c'est à dire le nombre d'arcs nécessaire pour joindre x et y).

Les fonctions 4.20, 4.22,et 4.25 utilisent une simple constante λ pour diminuer les poids de pertinences durant la propagation, comme dans les expérimentations présentées dans [84]. L'importance du paramètre distance est évaluée dans les fonctions 4.21, 4.24, 4.27, grâce au paramètre α.

Dans les expérimentations que nous présentons dans cette section, l'équation 4.5 a été utilisée pour le calcul du poids des noeuds feuilles, et les fonctions de propagation ont été testées par groupes de la façon suivante :

- Equations 4.20, 4.22, 4.25 : Courbe λ
- Equations 4.21, 4.24, 4.27 : Courbe α
- Equations 4.21, 4.23, 4.26 : Courbe *mixte*

Pour chacun de ces groupes d'équations, nous avons, selon les cas, fait varier les valeurs de λ ou de α entre 0.5 et 1[4].

La distance séparant les noeuds dans l'arbre du document semble jouer un rôle important lors de la propagation sur la dimension de spécificité (figure 4.15), puisque ce sont les fonctions utilisant une simple constante pour diminuer les poids durant la propagation qui obtiennent les moins bonnes précisions moyennes. On observe cependant un comportement inverse pour la dimension d'exhaustivité (figure 4.16). Cette observation rejoint celle que nous avons fait pour les requêtes orientées contenu : le fait d'utiliser une constante pour diminuer le poids des noeuds internes revient en fait à faire une somme pondérée des poids des noeuds feuilles. Par conséquent, les noeuds les plus hauts dans

[4] *Des expérimentations, non présentées ici, ont aussi été effectuées en faisant varier λ et α entre 0.1 et 0.4 : les précisions moyenne sont inférieures aux résultats obtenus ci-dessous, et ce pour toutes les fonctions d'agrégation*

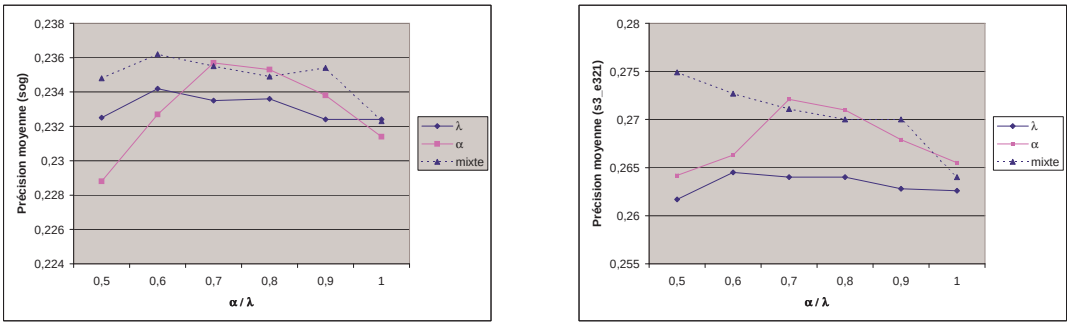


FIG. 4.15 – Evolution de la précision moyenne en fonction de C ou α, fonctions d'agrégation orientées spécificité

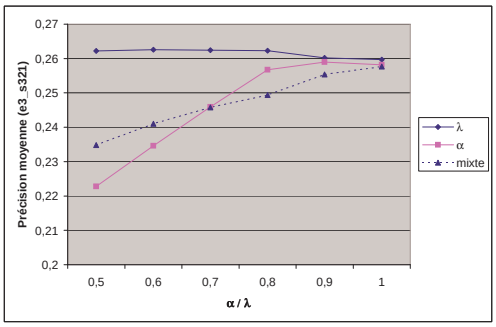


FIG. 4.16 – Evolution de la précision moyenne en fonction de C ou α, fonction d'agrégation orientée exhaustivité

la structure des documents (c'est à dire les noeuds près du noeud *racine* ou le noeud *racine* lui-même) ont un poids de pertinence plus élevé et sont ainsi mieux classés sur les noeuds situés plus profondémment dans la structure. Comme les noeuds les plus hauts dans la hiérarchie sont aussi les plus grands, le critère d'exaustivité sera plus probablement observé. Cette observation est confirmée par le fait que plus α est proche de 1, plus les performances entre les fonctions utilisant une constante et les fonctions utilisant la distance entre les noeuds sont comparables.

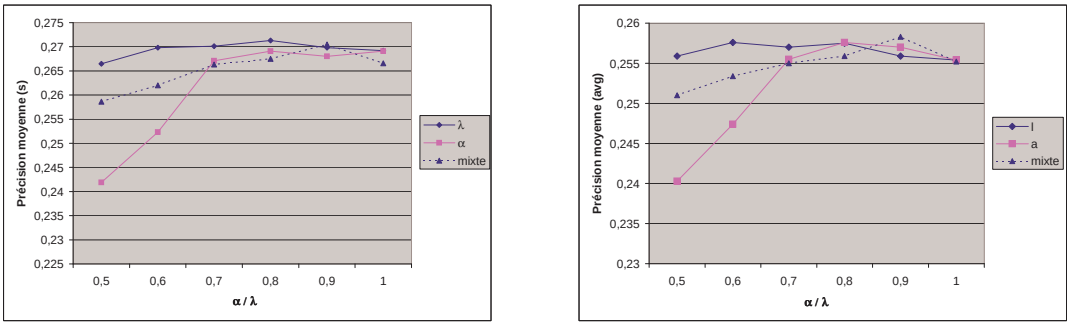


FIG. 4.17 – Evolution générale de la précision moyenne en fonction de C ou α

D'une manière générale (figure 4.17), on obtient des résultats sensiblement

meilleurs lorsque la distance entre les noeuds est utilisée que lorsqu'elle ne l'est pas. La meilleure combinaison est obtenue en utilisant les équations 4.21, 4.23, et 4.26, avec dans l'équation 4.21 une valeur de α optimale plus élevée que dans le cas des requêtes orientées contenu : les meilleurs résultats sont effet obtenus avec $\alpha = 0.9$, ce qui montre que la distance entre les noeuds est un paramètre ayant plus d'importance pour les requêtes orientées contenu que pour les requêtes orientées contenu et structure.

4.5.3 Conditions de structure : contraintes strictes ou contraintes vagues ?

Dans les expérimentations que nous avons présentées, la seule condition de structure traitée de manière stricte est celle concernant l'élément cible. Si les autres conditions de structure sont respectées, le poids des noeuds cible n'en sera que plus grand, et ils seront mieux classés par le système.
Nous avons renouvelé nos expérimentations en considérant que *toutes* les conditions de structure doivent être respectées. Les performances obtenues baissent en moyenne de 20%. Ceci peut en partie être expliqué par la façon dont sont effectués les jugements de pertinence de la tâche SCAS d'INEX : les juges ne prennent en effet pas en compte les conditions de structure, et les résultats sont ensuite filtrés pour répondre à ces dernières.
De manière opposée, nous avons conduit des expérimentations en considérant les conditions de structure de manière vague, c'est à dire comme une indication (et non une obligation) sur le type de résultat à fournir à l'utilisateur. Afin de répondre à ce besoin, qui semble plus proche des attentes réelles des utilisateurs, nous avons testé deux approches :

- la première consiste à augmenter l'index Dictionnaire avec des synonymes de balises plus étendus [182]. Par exemple, pour la tâche SCAS et d'après les instructions d'INEX, une balise *sec* (*section*) est considérée équivalente aux balises *ss1*, *ss2* et *ss3*, qui représentent des *sous-sections*. Pour la tâche VCAS, nous nous proposons par exemple d'étendre cette équivalence avec les balises *p* (paragraphe) et *ip1* (premier paragraphe d'une section). Pour évaluer cette approche, nous avons créé manuellement 4 index Dictionnaire différents (DICT, DICT2, DICT3 et DICT4), avec DICT le dictionnaire utilisé pour la tâche SCAS et DICT2, DICT3, DICT4, des dictionnaires avec des sysnonymes de balises au sens de plus en plus large.
- une deuxième approche consiste à traiter les requêtes CAS comme de simples requêtes CO, c'est à dire en ne gardant que les conditions de contenu, et en effectuant une propagation pondérée et une rétropropagation sur les noeuds ($\alpha = 0.6$ et ρ=0.9).

Ces approches sont évaluées grâce au jeu de test de la campagne d'évaluation INEX 2004. Les résultats que nous obtenons sont présentés dans le tableau

4.10.

	sog	s3_e321	e3_s321	s	avg	% imbrication
DICT	0.0300	0.0295	0.0374	0.0312	0.0346	17.75
DICT2	0.0436	0.0401	0.0552	0.0454	0.0475	38.54
DICT3	0.0459	0.0355	0.1214	0.0701	0.0615	58.27
DICT4	0.0548	0.0436	0.1056	0.0730	0.0693	73.85
CO	0.0557	0.0534	0.0988	0.1007	0.0740	83.54

TAB. 4.10 – Précisions moyennes pour la tâche VCAS 2004

On peut noter que plus l'index Dictionnaire utilisé est étendu, plus le pourcentage d'imbrication des noeuds est élevé, et plus la précision moyenne augmente. Ceci n'est pas vraiment surprenant, car comme les conditions de structure sont traitées de manière incertaine, la base de rappel obtenue d'après les jugements de pertinence est surpeuplée, comme c'est le cas pour les requêtes CO. Tous les noeuds imbriqués doivent donc être renvoyés pour obtenir de bonnes performances avec les mesures actuelles. Ceci est confirmé par les résultats que nous obtenons en ne considérant que les conditions de contenu et en les traitant selon le modèle que nous proposons pour les requêtes CO (propagation pondérée et rétropropagation).
Comme on peut le constater, les performances sont légèrement meilleures lorsque seules les conditions de contenu des requêtes sont traitées. Ces observations confirment celles effectuées par de nombreux participants à la campagne INEX 2004. Cependant, on notera que la précision moyenne pour des taux de rappel peu élevés est meilleure dans le cas où les conditions de structure sont interprétées. Cette observation rejoint les résultats présentés dans [191]. Comme l'utilisateur évalue un système avant tout grâce aux premiers éléments renvoyés, nous considèrerons donc qu'il est préférable de traiter les conditions de structure pour répondre au mieux à la tâche VCAS.

4.6 Quelques considérations sur le choix de l'unité d'indexation minimale

Les expérimentations présentées dans cette section ont pour objectif d'évaluer l'impact ed l'unité d'indexation minimale choisie. Comme nous l'avons vu dans la section 4.3.1, deux index ont été créés : un index contenant tous les noeuds de la collection (IC) et un index dans lequel les très petits noeuds ont été enlevés (IS), de taille égale à environ 80% du premier. Les évaluations effectuées jusqu'ici l'ont été en utilisant l'index IC, et nous nous proposons dans cette section de réévaluer nos modèles sur l'index IS. On trouvera une comparaison des précisions moyennes obtenues dans le tableau 4.11. Pour chacune des tâches

d'INEX, les expérimentations présentées utilisent les paramètres optimaux fixés dans les sections précédentes.

		sog	s3_e321	e3_s321	s	avg
CO 2003	*IC*	0.0990	0.1021	0.1667	0.1515	0.1280
	IS	0.1006	0.1009	0.1628	0.1436	0.1254
	Gain	+1.6%	-1.2%	-2.4%	-5.3%	-2.1%
CO 2004	*IC*	0.0905	0.0808	0.1508	0.1236	0.1128
	IS	0.0912	0.0819	0.1708	0.1275	0.1205
	Gain	+0.7%	+1.3%	+13.2%	+3.1%	+6.8%
SCAS 2003	*IC*	0.2354	0.2702	0.2554	0.2705	0.2583
	IS	0.2131	0.2392	0.2416	0.2864	0.2469
	Gain	-9.5%	-11.5%	-5.5%	+5.8%	-4.5%
VCAS 2003	*IC*	0.0548	0.0436	0.1056	0.0730	0.0693
	IS	0.0520	0.0379	0.0914	0.0535	0.0593
	Gain	-5.1%	-13%	-13.5%	-26.7%	-14.5%

TAB. 4.11 – Comparaison des précisions moyennes obtenues sur deux index

Une première remarque concerne les différences de résultats observés sur la tâche CO en 2003 et 2004. De manière intuitive, on peut penser que l'IC est plus adapté au traitement de la tâche, et ce à cause de la propagation pondérée (utilisant les noeuds de petites tailles) que nous proposons. Cette intuition semble se confirmer sur le jeu de test 2003, pour lequel on observe en moyenne 2% de perte de précision. En revanche, pour le jeu de test 2004, la propagation pondérée n'augmente pas les performances (comme nous l'avons vu dans les sections 4.4.3 et 4.4.4, la prise en compte du contexte des éléments lors de la rétropropagation a un impact beaucoup plus grand), et dans ces conditions, l'IS nous permet d'obtenir de meilleures précisions moyennes.
En ce qui concerne les requêtes CAS, l'IC permet d'obtenir des performances significativement meilleures que l'IS, ce qui n'est pas étonnant puisque lors du traitement des CAS, les systèmes peuvent être amenés à traiter des conditions de structure très fines, et la hiérarchie complète des documents doit pouvoir être restituée.
D'une manière générale, on préfèrera utiliser l'index IC dans notre modèle de propagation de la pertinence. Ce dernier présente le double avantage d'être construit de manière complètement automatique et de permettre de répondre de manière optimale aux requêtes CO et CAS. Un index simplifié de type IS pourra cependant être utilisé lorsque l'on souhaitera diminuer le temps de réponse du sytème à une requête donnée.

4.7 Evaluation comparative avec les résultats des campagnes INEX 2003 et INEX 2004

L'objectif de cette section est de confronter nos résultats avec ceux obtenus par les participants d'INEX 2003 et 2004. En 2003, les soumissions officielles ont été classées grâce aux fonctions d'agrégation stricte et généralisée. En 2004, de nouvelles fonctions d'agrégation ont été utilisées, et les soumissions ont été classées selon chacune des fonctions d'agrégation et sur la moyenne de ces dernières. Afin d'homogénéiser la présentation de nos résultats, nous effectuerons ici une comparaison sur les fonctions d'agrégation stricte *s*, orientées spécificité *sog* et *s3_e321* et orientée exhaustivité *e3_s321*.

4.7.1 Tâche CO

Les tableaux 4.12 et 4.13 présentent les rangs et les précisions moyennes obtenus pour la fonction d'agrégation stricte par les différents participants à la tâche CO d'INEX 2003 (56 participants au total) et 2004 (70 participants au total). Nous avons inclus dans ces tableaux les résultats de notre approche afin de mettre en évidence le rang qu'on aurait obtenu dans ce cas. On trouvera aussi sur les figures 4.18 et 4.19 les courbes de rappel-précision de notre approche comparées aux courbes de rappel-précision des participations officielles à INEX. Nos courbes sont en gras, et le trait plein équivaut aux résultats obtenus sur l'index complet (IC), alors que le trait en pointillés représente les résultats obtenus sur l'index simplifié (IS).
Une première remarque est que notre approche obtient de bons résultats comparés aux soumissions officielles (premier rang pour la campagne 2003 et dans les 5 premiers pour la campagne 2004 pour la fonction d'agrégation stricte). Des résultats similaires, non présentés sous forme de tableau mais visibles grâce aux courbes de rappel-précision, sont obtenus pour les autres fonctions d'agrégation.

Parmi les meilleures approches, on citera l'Université d'Amsterdam [192, 191], qui propose une approche basée sur les modèles de langage, en utilisant un modèle de langage par élément. IBM Haifa Research Lab [135, 136] adapte le modèle vectoriel, en utilisant 6 index différents pour les termes (index article, index section, index paragraph, index abstract,...). Les résultats des recherches sur les différents index sont ensuite fusionnés. Dans [136], les formules de pondération des noeuds intègrent le poids des documents, grâce à un pivot. L'approche présentée dans [46] (Université de Waterloo) utilise quant à elle une fonction de pondération basée sur celle du BM25, en considérant les documents dans leur globalité.

Rang	Précision moyenne	Organisation	Identifiant du run
	0.1515		***XFIRM-Index Complet***
	0.1436		***XFIRM-Index Simplifié***
1	0.1214	U. of Amsterdam	UamsI03-CO-lambda=0.20
2	0.1144	U. of Amsterdam	UamsI03-CO-lambda=0.5
3	0.1102	U. of Amsterdam	UamsI03-CO-lambda=0.9
4	0.1001	Universität Duisburg-Essen	factor0.2
5	0.0952	IBM, Haifa Research lab	CO-TDB-With-No-Clustering
6	0.0929	LIP6	local-okapi-element,list,ef
7	0.0915	Universität Duisburg-Essen	difra sequential
8	0.0780	Carnegie Mellon University	LM context TDK
9	0.0708	Universität Duisburg-Essen	factor0.5
10	0.0688	University of Bayreuth	co second

TAB. 4.12 – Classement de notre système parmi les résultats officiels de la campagne d'évaluation INEX 2003 pour une fonction d'agrégation stricte, tâche CO

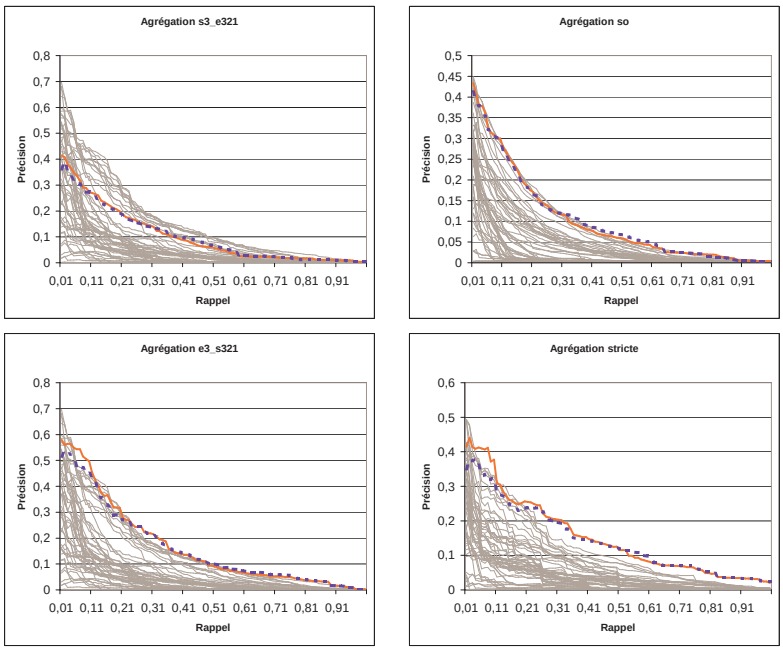


FIG. 4.18 – Courbes de rappel-précision de notre systèmes et des résultats officiels de la campagne d'évaluation INEX 2003, tâche CO

Rang	Précision moyenne	Organisation	Identifiant du run
1	0.1524	University of Waterloo	Waterloo-Baseline
2	0.1466	University of Waterloo	Waterloo-Expanded
3	0.1428	IBM Haifa Research Lab	CO-0.5-LAREFIENMENT
4	0.1327	IBM Haifa Research Lab	CO-0.5
	0.1275		***XFIRM-Index Simplifié***
5	0.1271	LIP6	simple
	0.1236		***XFIRM-Index Complet***
6	0.1225	Queensland Univ. of Tech.	CO_PS_099_049
7	0.1207	Queensland Univ. of Tech.	CO_PS_Stop50K_099_049
8	0.1124	IBM Haifa Research Lab	CO-0.5-Clustering
9	0.1100	University of Amsterdam	UAms-CO-T-FBack
10	0.1013	University of Amsterdam	UAms-CO-T

Tab. 4.13 – Classement de notre système parmi les résultats officiels de la campagne d'évaluation INEX 2004 pour une fonction d'agrégation stricte, tâche CO

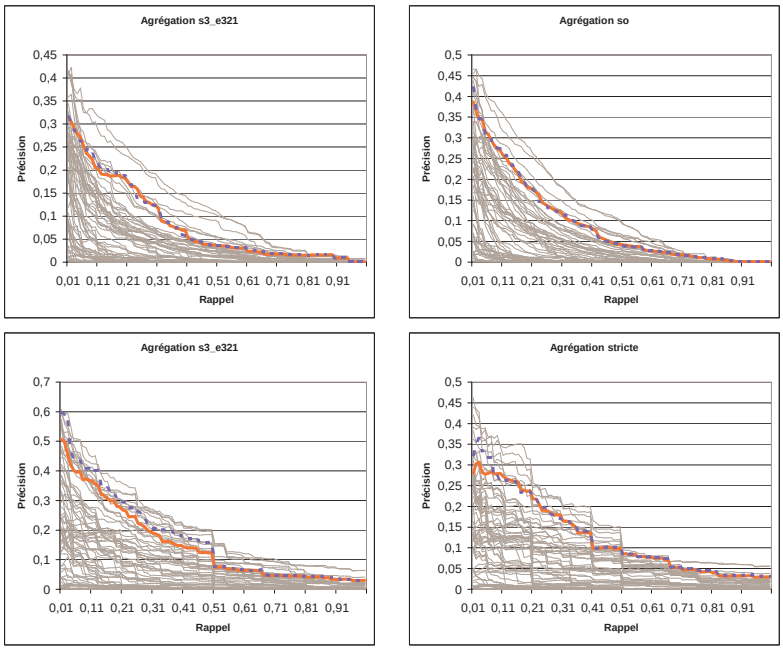


Fig. 4.19 – Courbes de rappel-précision de notre système et des résultats officiels de la campagne d'évaluation INEX 2004, tâche CO

Pour la campagne d'évaluation 2004, les résultats obtenus dans [136] et [46] montrent que la prise en compte du document est nécessaire, ce qui est confirmé par nos résultats. Enfin, les soumissions classées deuxième et troisième [46, 136] effectuent un cycle de réinjection de la pertinence (relevance feedback).

4.7.2 Tâche SCAS

Le tableau 4.14 compare les rangs et précisions moyennes obtenus pour la fonction d'agrégation stricte par les différents participants à la tâche SCAS d'INEX 2003 (37 participants an total) et les résultats de notre approche. On trouvera sur la figure 4.20 les courbes rappel-précision de notre approche comparées aux courbes rappel-précision des soumissions officielles. Comme pour les requêtes CO, nos courbes sont en gras, et le trait plein équivaut aux résultats obtenus sur l'index complet IC alors que le trait en pointillés représente les résultats obtenus sur l'index simplifié IS.
Une première remarque est que notre approche est bien classée par rapport aux soumissions officielles pour la fonction d'agrégation stricte. Des résultats similaires sont obtenus pour les autres fonctions d'agrégation, comme le montre les courbes de la figure 4.20.
Conformément aux directives d'INEX 2004, nos soumissions utilisent seulement le champ *Title* des requêtes, alors qu'en 2003, de telles restrictions n'étaient pas imposées. On notera que lorsque nous utilisons les champs *Title* et *Keywords*, nos performances augmentent d'environ 8%, ce qui nous classerait entre la première et la deuxième place des soumissions officielles.
Ces résultats améliorent enfin considérablement ceux que nous avions obtenus lors de notre participation officielle en 2003. Cette participation était basée sur une approche "fetch and browse" [185] : une première recherche était effectuée grâce au moteur de recherche plein-texte Mercure [24], et les documents résultats étaient ensuite parcourus pour rechercher les parties les plus spécifiques. Nous avions alors été classés 24ème pour la fonction d'agrégation stricte. Cette amélioration n'est pas surprenante, car le modèle XFIRM est capable de traiter toutes les conditions de contenu, alors que les soumissions effectuées avec le moteur de recherche Mercure ne vérifiaient que les conditions sur les éléments cibles.

Parmi les meilleures approches, on citera l'Université d'Amsterdam [192], qui utilise des modèles de langages. L'Université technologique de Queensland [86] utilise une méthode basée sur le filtrage pour trouver les unités d'information les plus spécifiques. Enfin, IBM Haifa Research Lab propose une adaptation du modèle vectoriel [136].

Rang	Précision moyenne	Organisation	Identifiant du run
1	0.3182	U. of Amsterdam	UamsI03-SCAS-MixedScore
2	0.2987	U. of Amsterdam	UamsI03-SCAS-ElementScore
	0.2864		***XFIRM-Index Simplifié***
	0.2705		***XFIRM-Index Complet***
3	0.2601	Queensland Univ. of Tech.	CASQuery_1
4	0.2476	University of Twente and CWI	LMM-Comp.Retrieval-SCAS
5	0.2458	IBM, Haifa Research lab	SCAS-TK-With-Clustering
6	0.2448	Universität Duisburg-Essen	scas03-way1-alias
7	0.2437	RMIT University	RMIT_SCAS_1
8	0.2419	RMIT University	RMIT_SCAS_2
9	0.2405	IBM, Haifa Research lab	SCAS-TDK-With-No-Clus.
10	0.2352	RMIT University	RMIT_SCAS_3

Tab. 4.14 – Classement de notre système parmi les résultats officiels de la campagne d'évaluation INEX 2003 pour une fonction d'agrégation stricte, tâche SCAS

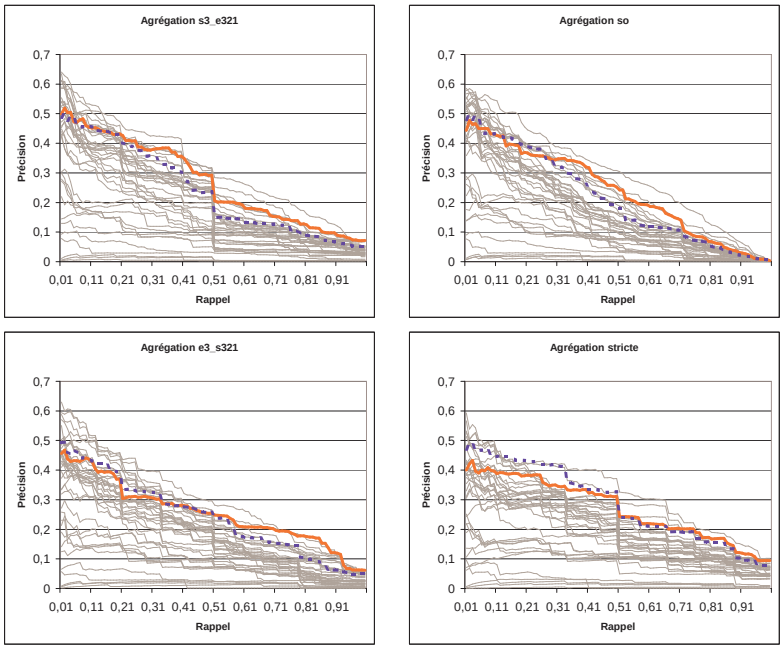


Fig. 4.20 – Courbes de rappel-précision de notre système et des résultats officiels de la campagne d'évaluation INEX 2003, tâche SCAS

4.7.3 Tâche VCAS

Comme pour les tâches CO et SCAS présentées dans les paragraphes précédents, on trouvera dans le tableau 4.15 et sur la figure 4.21 une comparaison de nos résultats avec les soumissions officielles de la tâches VCAS 2004. Sur la figure 4.21, nos soumissions sont en gras, et le trait plein correspond aux résultats pour l'index IC alors que le trait en pointillés correspond aux résultats pour l'index IS.
Une première remarque est que nous aurions été classés parmi les 10 meilleures approches pour toutes les fonctions d'agrégation. Notre soumission utilisant seulement les conditions de contenu et les traitant comme des requêtes CO aurait été classée à la cinquième place en ce qui concerne la fonction d'agrégation stricte.

Les meilleurs résultats sont obtenus par l'Université Technologique de Queensland [85], qui utilise les conditions de structure dans le seul but d'augmenter le score de certains éléments. De nombreuses approches [193, 148] obtiennent de bons résultats en ne traitant que les conditions de contenu des requêtes.

4.8 Expérimentations sur une collection de données hétérogènes

Afin de vérifier la faisabilité de notre approche sur une collection de documents suivant des DTDs différentes, nous avons participé à la tâche hétérogène de la campagne d'évaluation INEX 2004.
Dans cette tâche, de nouvelles collections ont été proposées aux participants. Ces collections sont décrites dans le tableau 4.16.

Les collections ajoutées à la collection originale d'INEX sont principalement composées de références bibliographiques, ce qui nous permet de dire qu'il s'agit plutôt de collections orientées données que de collections orientées documents. Les tailles des différents documents de ces collections sont très hétérogènes : les plus petits documents font quelques Ko alors que le plus gros fait 300 Mo. Différents formats de requêtes ont été définis pour répondre aux challenges liés aux collections hétérogènes [60] :

- requêtes CO (Content Only) : elles sont l'équivalent des requêtes CO de la tâche ad-hoc. Le but est de développer des méthodes indépendantes de toute DTD.
- requêtes BCAS (Basic Content and Structure) : ces requêtes se focalisent sur la combinaison d'une seule condition de contenu associée à

Rang	Précision moyenne	Organisation	Identifiant du run
1	0.1375	Queensland Univ. of Tech.	VCAS_PS_stop50K_099_049
2	0.1365	Queensland Univ. of Tech.	VCAS_PS_099_049
3	0.1260	University of Amsterdam	Uams-CAS-T-Fback
4	0.1058	Queensland Univ. of Tech.	VCAS_PS_stop50K_049025
5	0.1053	IRIT	VTCAS2004TC35xp200sC
6	0.0792	UCLA	VCAS-3
7	0.0787	Cirquid Project	LMM-VCAS-Relax-0.35
8	0.0751	Cirquid Project	LMM-VCAS-Relax-0.35
9	0.0735	University of Amsterdam	Uams-CAS-T-XPath
	0.0730		***XFIRM-Index Complet***
10	0.0710	Carnegie Mellon University	Lemur_CAS_as_CO_NoStrem
	0.0535		***XFIRM-Index Simplifié***

TAB. 4.15 – Classement de notre système parmi les résultats officiels de la campagne d'évaluation INEX 2004 pour une fonction d'agrégation stricte, tâche VCAS

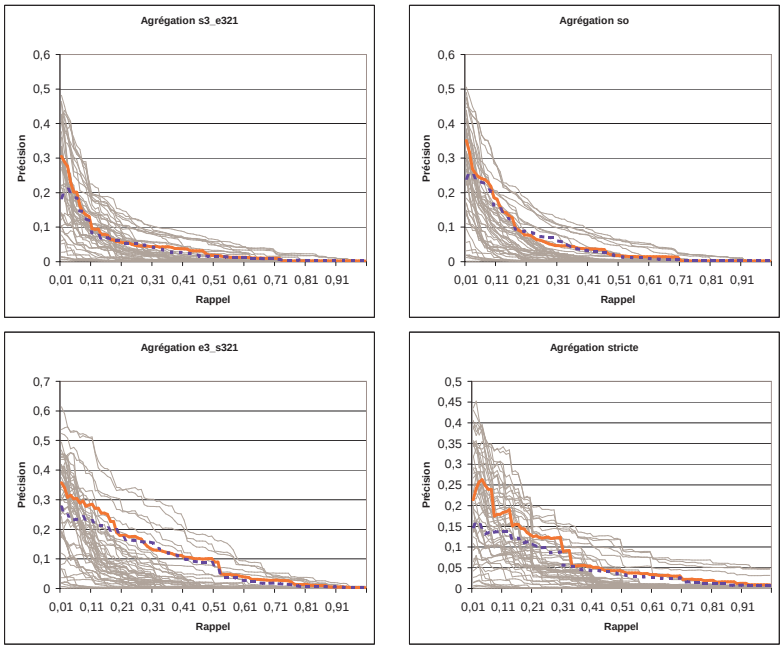


FIG. 4.21 – Courbes de rappel-précision de notre système et des résultats officiels de la campagne d'évaluation INEX 2004, tâche VCAS

Collection	Taille (en Mo)	Nombre de noeuds
IEEE Computer Society	494	8 200 000
Berkeley	33.1	1 194 863
CompuScience	313	7 055 003
bibdb Duisburg	2.08	40 118
DBLP	207	5 114 033
hcibib	30.5	308 554
qmul-dcs-pubdb	1.05	23 436

Tab. 4.16 – Collections de la tâche hétérogène

seule une condition de structure (par exemple : *sec[about(.,search engines)]* ou *section[about(.,search engines)]*). Le but est d'être capable de traiter les conditions de structure avec des noms de balises n'appartenant pas nécessairement à toutes les collections, mais pouvant avoir des synonymes dans certaines.

- requêtes CCAS (Complex CAS) : elles sont l'équivalent des requêtes CAS définie en langage NEXI pour la tâche ad-hoc. Le but est de permettre des transformations et des correspondances partielles de chemin entre les différentes collections, sans perdre le composant RI de la requête.
- requêtes ECCAS (Extended Complex CAS) : ces requêtes supposent que l'utilisateur est capable de donner la probabilité d'existence d'une contrainte structurelle donnée. Par exemple, la requête *//author(0.8)[about(title(0.5),'Information retrieval')]* signifie que l'utilisateur recherche des auteurs de publications sur la RI, avec une probabilité de 80% que la balise concernée soit *author* (c'est à dire qu'il y a 20% de probabilité que l'information recherchée soit dans un élément portant un nom différent). Pour déterminer que la publication parle de RI, l'utilisateur pense que dans 50% des cas, le titre de la publication va contenir les termes '*Information retrieval*'.

La tâche hétérogène était proposée pour la première fois en 2004 et a été principalement explorative. Les participants ont proposé 10 requêtes CO, 1 requête BCAS et 13 requêtes CCAS.
Comme les structures d'index de notre modèle sont prévues pour traiter des collections de données hétérogènes, le processus d'indexation n'a pas posé de réels problèmes. On notera cependant que pour les fichiers de très grande taille (notamment celui de 300 Mo), un découpage du fichier en plusieurs sous-fichiers a été nécessaire pour que le parseur puisse parcourir l'arbre du document.
Pour les requêtes CO, nous avons utilisé un modèle identique à celui proposé ci-dessus. Pour les requêtes BCAS et CCAS, un nouvel index Dictionnaire a été construit manuellement (en comparant les différentes DTDs). Les résultats de ces expérimentations ne sont pas encore connus, un certain nombre de problème restant à régler au niveau des jugements de pertinence.
Suite à ces expérimentations, de nouvelles questions sont soulevées par rapport à notre modèle :

- comment traiter la très grande différence de taille des documents d'une collection ? Alors que pour la collection originale INEX, on pouvait faire correspondre structure physique et structure logique de document, ce n'est maintenant plus le cas : cela n'a aucun sens de renvoyer à l'utilisateur l'élément racine d'un document de 300 Mo...
- la tâche hétérogène d'INEX mélange des collections orientées données et des collections orientées documents. Les méthodes adaptées pour ces dernières ne le sont pas forcément pour les premières. Par conséquent, ne faut-il pas plutôt considérer plusieurs collections homogènes et appliquer sur chacune des méthodes appropriées qu'une seule collection hétérogène pour laquelle une méthode très généraliste devra être appliquée ? Par exemple, la notion de rétro-propagation dans le cadre des requêtes CO n'est pas forcément appropriée pour toutes les collections...
- comment construire l'index Dictionnaire automatiquement ?

4.9 Conclusion et discussions

Dans ce chapitre, nous avons présenté les expérimentations et les résultats obtenus par notre modèle flexible de recherche dans des documents structurés. Nos expérimentations ont été basées sur les campagnes d'évaluation INEX 2003 et 2004.

Une première série d'évaluations a concerné le traitement des requêtes orientées contenu. Nous avons déterminé les fonctions optimales pour le calcul de poids des noeuds feuilles et la propagation de la pertinence, notamment en évaluant l'impact de la distance entre les noeuds durant la propagation. Il résulte de ces expérimentations qu'une adaptation de la formule *tf-idf* à la granularité de l'information que nous traitons (formule *tf-ief*) permet d'obtenir les meilleures performances, et que la distance entre les noeuds joue un rôle prépondérant durant la propagation. Nous avons ensuite évalué l'impact de la taille des éléments pour le calcul de leur informativité et avons conclu que la longueur devait intervenir lors de la propagation, plus particulièrement pour faire ressortir l'information contenue dans les noeuds de petite taille. Nous avons ensuite montré que la pertinence du document dans lequel se trouvent les sous-arbres joue un rôle non négligeable dans le calcul de leur propre informativité.

Une deuxième série d'évaluation a concerné les requêtes contenant des conditions de structure et de contenu. Comme pour les requêtes contenant de simples conditions de contenu, nous avons déterminé que la formule de pondération op-

timale pour le calcul du poids des noeuds feuilles se base sur *tf-ief*. La distance entre les noeuds pour les différentes propagations est aussi un paramètre non négligeable lors des propagations, mais semble moins important que pour les requêtes composées de simples conditions de contenu. Nous avons également évalué la façon (stricte ou vague) de répondre aux conditions de structure, et il résulte de ces expérimentations que l'utilisateur considère les conditions de structure des requêtes comme des indications sur ce qu'il recherche, et non comme des contraintes strictes.
Les résultats obtenus par notre modèle sont nettement supérieurs à ceux que nous avions obtenus lors que notre participation officielle en 2003 en utilisant une approche basée sur une méthode "fetch and browse" [185]. Avec cette amélioration, nous nous positionnons systématiquement parmi les 10 meilleures approches des campagnes 2003 et 2004, quelle que soit la fonction d'agrégation utilisée. Nous sommes même nettement au-dessus des autres approches pour la tâche CO 2003 (25% d'augmentation par rapport au premier pour la fonction d'agrégation stricte).

L'étude de nos résultats a cependant soulevée quelques problèmes au niveau des mesures d'évaluation utilisées, particulièrement en ce qui concerne les soumissions contenant des noeuds imbriqués. Nous avons donc réévalué notre méthode avec la mesure XCG (présentée dans [109]), et les premiers résultats semblent confirmer la robustesse de notre approche.
Enfin, nous avons pu participer à la tâche hétérogène de la campagne d'évaluation 2004, ce qui nous a permis de cerner certaines limites de notre approche, plus particulièrement en ce qui concerne le traitement de documents orientés données.

Conclusion générale

Synthèse

Les travaux présentés dans ce mémoire se situent dans le contexte général de la recherche d'information, et plus particulièrement dans le cadre de la recherche d'information structurée.
Un système de recherche d'information structurée combine la structure et le contenu des documents pour répondre de la manière la plus spécifique et exhaustive possible au besoin en information de l'utilisateur. Le but est alors de renvoyer à l'utilisateur des unités d'information (c'est à dire des sous-arbres ou encore des noeuds de documents XML) focalisées sur son besoin, et non plus des documents entiers. Pour ce faire, des solutions concernant le stockage des documents, leur interrogation ainsi que le tri des unités d'information résultats doivent être proposées. Nous nous sommes intéressés dans ce mémoire à proposer une solution flexible pour répondre à de telles problématiques.

Le modèle que nous proposons repose sur :
- un *modèle générique de représentation des données*, permettant de traiter des documents possédant des structures hétérogènes et de naviguer aisément dans la structure arborescente des documents ;
- un *langage d'interrogation simple*, ne nécessitant pas la connaissance de syntaxes complexes comme dans SQL ou XQuery [66], et permettant à l'utilisateur d'exprimer son besoin selon divers degrés de précision. S'il ne connait pas la structure des documents qu'il interroge ou bien que le type de l'unité d'information qui lui sera renvoyée lui importe peu, il peut formuler son besoin à l'aide de simples mots clés, et laisser le système décider de la granularité de l'information pertinente. Si son besoin est plus précis, il peut exprimer des conditions sur la structure des documents et relier éventuellement ces conditions de manière à former une hiérarchie : on parle alors d'arbre de la requête
- un modèle de recherche reposant sur un *modèle de propagation de la pertinence* des noeuds feuilles des documents vers les noeuds internes.

Pour les requêtes contenant de simples conditions de contenu, le problème

réside principalement dans la granularité de l'information à renvoyer à l'utilisateur : il s'agit de trouver le sous-arbre de taille minimale qui répondra à sa requête. Dans notre méthode, un premier score de pertinence des noeuds feuilles par rapport aux conditions de contenu est calculé. Ces scores sont ensuite propagés dans l'arbre du document afin de calculer les scores de pertinence des noeuds internes. Afin de répondre au critère de spécificité, les scores des noeuds feuilles sont diminués pendant la propagation. Nous répondons au critère d'exhaustivité en proposant la notion d'informativité d'un noeud, reposant sur les concepts de *propagation pondérée par la taille des noeuds feuilles* et de *pertinence contextuelle*. Pour la propagation pondérée, les noeuds de petite taille voient leur importance accrue lors de la propagation, car pour la plupart, ils contiennent de l'information que l'auteur du document désirait mettre en valeur. La notion de pertinence contextuelle repose quant à elle sur l'intuition suivante : le concepteur d'un document, même s'il s'exprime sur des sujets différents, suit une certaine unité de pensée. La pertinence d'un sous-arbre est donc liée à la pertinence du document dans lequel il se trouve. Pour modéliser cette intuition, nous nous proposons de propager le score du document du haut vers le bas dans l'arbre du document (c'est à dire d'effectuer une rétropropagation), afin de calculer un nouveau score de pertinence pour les noeuds internes.

Les requêtes composées de conditions de structure et de contenu peuvent quant à elles donner une indication sur le type de l'information à renvoyer à l'utilisateur (on parle de noeuds cibles). La principale problématique réside alors sur la façon dont les conditions de structure sont interprétées (c'est à dire de manière stricte ou de manière vague). Dans notre modèle, des propagations successives dans la structure arborescente des documents nous permettent d'augmenter le score de pertinence des noeuds cibles, et donc de mieux classer des noeuds répondant à toutes les conditions de structure. Des éléments possédant une structure différente de celle la requête peuvent ainsi être renvoyés à l'utilisateur. Par exemple, un document possédant la structure /a/b/c sera pertinent pour une requête /a/d/c, mais aussi pour une requête /a/c/b. Lorsque l'utilisateur ne spécifie pas le type de l'élément qu'il désire voir renvoyer (pas d'élément cible), nous cherchons dans le cas des requêtes booléennes (type P2), les noeuds plus proches ancêtres communs répondant aux conditions de structure, et dans le cas des requêtes hiérarchiques (type P3), les noeuds répondant à la première condition de structure des requêtes (noeuds situés le plus haut dans la hiérarchie des documents).

Notre modèle apporte ainsi de la *flexibilité* dans la recherche à plusieurs niveaux : le modèle de représentation des documents (et donc la structure d'index) est générique, et permet de traiter des collections de documents hétérogènes, le langage d'interrogation permet à l'utilisateur d'exprimer son besoin selon

plusieurs degrés de précision, en indiquant ou non des conditions sur le type d'éléments qu'il recherche, et les conditions de contenu ainsi que les éventuelles conditions de structure des requêtes peuvent être traitées de manière vague.

Pour valider ces propositions, un prototype a été implémenté et nous avons effectué une série d'expérimentations sur des collections issues de la campagne d'évaluation INEX. La démarche d'évaluation que nous avons suivie respecte le canevas défini dans INEX. Ce choix est effectué pour pouvoir comparer et situer nos travaux par rapport à ceux présentés dans le cadre d'INEX 2003 et INEX 2004.
Une première série d'évaluations a été effectuée afin de fixer les paramètres de notre modèle pour le traitement des requêtes orientées contenu. Nous avons ainsi déterminé les fonctions optimales pour le calcul des poids des termes des noeuds feuilles et la propagation de la pertinence, notamment en évaluant l'impact de la distance entre les noeuds durant la propagation. Il résulte de ces expérimentations qu'une adaptation de la formule *tf-idf* à la granularité de l'information que nous traitons (formule *tf-ief*) permet d'obtenir les meilleures performances, et que la distance entre les noeuds joue un rôle prépondérant durant la propagation. Nous avons ensuite évalué l'impact de la longueur des éléments pour le calcul de leur informativité et avons conclu que la longueur devait intervenir lors de la propagation, plus particulièrement pour faire ressortir l'information contenue dans les noeuds de petite taille. Nous avons ensuite montré que la pertinence du document dans lequel se trouvent les sous-arbres joue un rôle non négligeable dans le calcul de leur propre informativité.

Une deuxième série d'évaluations a porté sur les requêtes contenant des conditions de structure et de contenu. Comme pour les requêtes contenant de simples conditions de contenu, nous avons déterminé que la formule de pondération optimale pour le calcul du poids des termes des noeuds feuilles se base sur *tf-ief*. La distance entre les noeuds pour les différentes propagations est aussi un paramètre non négligeable lors des propagations, mais semble moins important que pour les requêtes composées de simples conditions de contenu. Nous avons également évalué la façon (stricte ou vague) de répondre aux conditions de structure, et il résulte de ces expérimentations que l'utilisateur considère les conditions de structure des requêtes comme des indications sur ce qu'il recherche, et non comme des contraintes strictes.

Notre modèle a été comparé aux soumissions officielles des campagnes 2003 et 2004, et présente des performances intéressants comparées à celles des meilleurs participants.

Nos expérimentations ont cependant soulevé le problème de l'imbrication des noeuds dans les résultats : les mesures utilisées à ce jour dans INEX impliquent en effet, si l'on souhaite obtenir des performances correctes, que les résultats contiennent des noeuds inclus les uns dans les autres. Nous disposons depuis peu de la mesure d'évaluation XCG censée répondre à ce problème, et nos premières évaluations semblent confirmer la robustesse de notre approche. Un autre problème concerne les jugements de pertinence que nous utilisons : ces derniers, selon les participants, répondent à des tâches de recherche différentes, et les modèles utilisés pour répondre de manière optimale aux requêtes devraient donc être différents. La définition de modèles utilisateurs simples a heureusement été proposée lors du dernier workshop INEX, et ces derniers devraient être utilisés pour les campagnes à venir.

Perspectives

Les perspectives envisageables à nos travaux portent sur plusieurs points.

Un premier point concerne la *pondération des termes d'indexation*. Dans notre modèle, cette pondération est effectuée en prenant en compte l'importance du terme au sein de la collection (importance globale) et au sein du noeud auquel il appartient (importance locale). Le concept même de document semi-structuré permet d'ajouter à ces deux dimensions celle du document. Dans notre modèle, la prise en compte de l'importance des termes au sein du document est effectuée lors du calcul de la pertinence contextuelle des éléments. Une première piste de recherche concerne donc l'intégration de cette importance "semi-globale" au niveau de la pondération des termes. Une seconde piste de recherche concerne l'intégration de la longueur des noeuds au niveau de la pondération des termes. Les premières expérimentations que nous avons effectuées montrent que l'introduction de cette longueur dans les formules de pondération ne permet pas d'améliorer les performances. D'autres expérimentations nous paraissent nécessaires pour confirmer ces résultats.

Un second point concerne le *traitement des conditions de structure*. Notre modèle permet d'y répondre de manière vague, et n'oblige pas l'utilisateur à spécifier le type d'élément qu'il désire voir retourné. Lorsque ce dernier n'indique effectivement pas d'élément cible, se pose alors le problème de la granularité de l'information à lui renvoyer, et ce plus particulièrement dans le cas des requêtes booléennes de type P2 et des requêtes hiérarchiques de type P3. Nous avons solutionné ce problème en cherchant le plus proche ancêtre commun

des noeuds résultats des composants des requêtes de type P2, et en renvoyant les noeuds les plus haut dans la hiérarchie des requêtes de type P3. D'autres solutions pourraient être adoptées, prenant en compte notamment les degrés divers de pertinence des noeuds résultats des sous-requêtes élémentaires : plus un noeud est pertinent par rapport aux autres noeuds résultats des autres sous-requêtes élémentaires, plus il fait pencher la balance pour être renvoyé à l'utilisateur.

Une troisième perspective concerne la *gestion automatisée de corpus de documents hétérogènes*. L'hétérogénéité des documents peut porter sur plusieurs points : leur *structure*, mais aussi leur *taille* ou leur *contenu*.
Considérons d'abord l'hétérogénéité structurelle. Une collection possède des structures hétérogènes lorsque les documents qui la composent suivent des DTDs différentes. Dans le cadre de notre modèle, nous répondons aux problèmes liés à cette hétérogénéité en construisant (manuellement) un dictionnaire des balises possédant une sémantique proche. Des méthodes automatiques doivent cependant être trouvées pour permettre l'interrogation générique des corpus. Plusieurs pistes de recherches sont possibles. La première consiste à élaborer automatiquement une structure générique des documents, qui permettra à l'utilisateur de ne gérer qu'une seule DTD lorsqu'il interroge le corpus. Tous les documents devront cependant être transformés selon cette structure générique, au risque de perdre quelque peu de la sémantique portée par leur structure. Une seconde piste de recherche serait d'établir des méthodes de traduction des différents documents dans chacune des DTDs de la collection. Chaque document serait donc présenté sous plusieurs versions, et quelle que soit la DTD utilisée par l'utilisateur au moment de sa requête, la recherche pourra être effectuée dans tout le corpus. Cettte méthode est cependant coûteuse, puisqu'elle augmente considérablement la taille de la collection.
Dans le cas de collections formées de documents possédant des tailles et des contenus différents, nos formules de propagation de la pertinence ne s'appliquent pas de manière optimale. En effet, la propagation de la pertinence dans l'arbre des documents ne peut pas s'effectuer de la même manière quand un document fait quelques Ko et qu'il possède une unité sémantique (il traite d'un même thème, aussi généraliste soit-il) que lorsqu'il fait 300 Mo et qu'il est conçu comme un catalogue de données. Des méthodes de correspondance d'arbres doivent être développées, et une reflexion doit être menée sur le traitement parallèle des documents orientés données et des documents orientés contenu.

Une quatrième perspective concerne l'intégration de la notion de *réinjection de la pertinence* (*relevance feedback*) à notre modèle. Dans le cadre de la RI structurée, la notion de réinjection de la pertinence intègre à la fois les notions de structure et de contenu. Toute la question est alors de savoir comment

intégrer l'information structurelle dans la formulation de la nouvelle requête. Comment interpréter les unités d'information jugées pertinentes et non pertinentes par l'utilisateur ? Doit-on considérer la structure des réponses comme une contrainte forte qu'il vaut mieux respecter, ou au contraire comme une indication des noeuds les plus probablement pertinents ?

Enfin, nous souhaiterions développer une *interface pour l'interrogation et pour la présentation des résultats* à l'utilisateur. L'interface d'interrogation devrait guider l'utilisateur dans la formulation de la requête, si possible de façon dynamique, en lui présentant par exemple les éléments de structure sur lesquels il peut interroger le système. La présentation des résultats soulève un grand nombre de questions : les résultats doivent-ils être présentés dans leur contexte (c'est à dire au sein du document) ou bien doivent-il, puisqu'ils sont censés être informatifs, apparaître indépendamment ? Doit-on regrouper les résultats par document ou bien présenter une simple liste triée de résultats ? Ce dernier point nous amène aussi à réfléchir au regroupement des unités d'informations [154] : la réponse à une besoin utilisateur peut être amenée par plusieurs éléments indépendants, chacun apportant une information supplémentaire à l'utilisateur. Pour répondre au mieux au besoin de l'utilisateur, ces éléments pourrraient être regroupés, et les résultats seraient alors présentés à l'utilisateur sous forme d'une liste de groupes d'éléments.

Annexe A

La galaxie XML

De nombreuses technologies sont venues se greffer autour d'XML, la plupart étant en cours de standardisation par le W3C, comme le montre la figure A.1. Nous nous proposons ici d'en détailler quelques unes en complément de celles présentées dans le chapitre 2 (XPath, SAX et DOM).

A.1 Les espaces de noms

Les espaces de nom (*namespaces*) permettent de disposer, dans un document XML, de balises provenant de différents catalogues : par exemple des balises HTML, MathML, etc. Il se peut que deux catalogues fournissent des balises de même nom, mais de significations différentes. Les espaces de nom résolvent ce problème : ils nomment de manière unique un objet (élément ou attribut) en associant un domaine à un ensemble de noms. En pratique, on préfixe l'objet de l'espace de nom correspondant.
Les espaces de nom sont identifiés par des URIs (*Uniform Resource Identifiers*), mais l'on précise pour chacun d'eux un " label " qui servira de préfixe aux balises concernées. Par exemple, la balise *b* dans le tableau A.1 propose des caractéristiques différentes selon qu'elle soit employée dans un contexte HTML (préfixe H) ou MathML (préfixe M).

A.2 XML Schema

XML Schema a pour but de remplacer les DTD (*Document Type Definition*) existantes. Comme nous l'avons vu dans le chapitre 2, la DTD d'un document

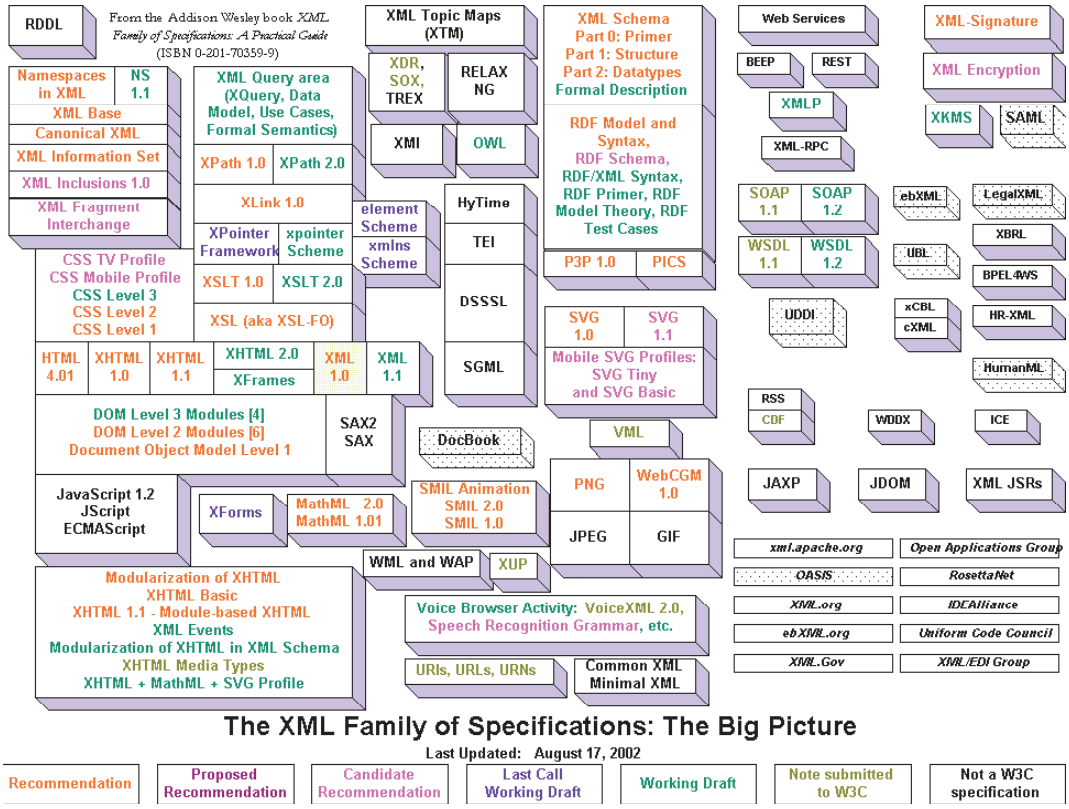


FIG. A.1 – La galaxie XML (d'après [172])

```
<exemple xmlns :H= " http ://www.w3c.org/REC-html40 "
        xmlns :M= " http ://www.w3c.org/REC-MathML ">
<H :b>
        <M :b>2</M :b>
</H :b>
</exemple>
```

TAB. A.1 – Exemple de définition d'un espace de noms XML

XML contient des informations de structure et de typage des données du document XML. XML Schema présente de nombreuses améliorations par rapport aux DTD, notamment une plus grande flexibilité et un typage plus important des données.
XML Schema est une recommandation du W3C depuis 2001 [64]. Elle est divisée en plusieurs sous-recommandations : *XML Schema Part 0* décrit l'utilisation d'XML Schema, *XML Schema Part 1* les structures et *XML Schema Part 2* les types de données.

A.3 XSL (*eXtensible Stylesheet Language*)

XSL est un langage de feuilles de styles. Il est composé de deux parties principales :

- XSLT (*XSL Transformation*) : langage de transformation de documents XML vers d'autres formats (PDF, HTML, ...) : le langage permet en fait d'effectuer des changements de balisage.
- XSL/FO : langage qui permet de formatter l'affichage et/ou l'impression d'un document XML (boîtes, positionnement, ordonnancement et propriétés d'affichage). Il s'agit d'une extension de CSS, associé aux documents HTML.

XSLT 1.0 et XSL/FO 1.0 sont des recommandations du W3C [45, 6].

A.4 XPointer

XPointer permet de spécifier des pointeurs dans des documents XML. Le but est de pouvoir désigner de manière précise et générique des parties d'une ressource XML et de représenter n'importe quelle sélection. Extension du standard XPath, XPointer réutilise en grande partie les mêmes concepts, règles d'évaluation et syntaxes. Il permet aussi de faire des sélections par motifs. Grâce à XPointer, il est possible de créer un lien vers n'importe quel endroit du document, sans avoir besoin d'ancre comme pour HTML, et donc sans avoir besoin de modifier la page cible.
Après 3 ans de travail, XPointer est devenu une recommandation officielle [92]. Elle se compose de 3 recommandations : *XPointer Framework* (la base), *XPointer element scheme* (adressage des éléments), et *Xpointer xmlns scheme* (interprétation des expaces de nommage dans les pointeurs), et la partie *Xpointer xpointer Scheme()* est encore à l'état de Working Draft.

- *XPointer Framework* décrit les types de média internet auxquels les recommandations XPointer proposées s'appliquent, ainsi que la syntaxe du langage XPointer.

```
<element xmlns :xlink=”http ://www.w3.org/1999/xlink/namespace/” xlink :type=”extended”>
        <xlink :locator href=”Source” role=”role1”/>
        <xlink :locator href=”Target” role=”role2”/>
        <xlink :arc from=”role1” to=”role2” show=”embed” actuate=”auto”/>
        <xlink :title>The link title<xlink :title/>
        <xlink :title xml :lang=”fr”>Description du lien<xlink :title/>
        ...
</element>
```

Tab. A.2 – Exemple de lien étendu XLink

- *XPointer element scheme* décrit comment, conjointement au XPointer Framework, il convient d'utiliser XPointer pour adresser des éléments XML dans une application.
- *Xpointer xmlns scheme* décrit le nom de domaine XML utilisé pour les pointeurs XML, y compris dans les préfixes et les noms qualifiés.
- Enfin, *Xpointer xpointer Scheme()* décrit en détail la syntaxe du langage XPointer.

A.5 XLink

XLink permet de généraliser les concepts hypertextes de HTML à XML. XLink 1.0 est une recommandation du W3C depuis 2001 [58].
Les liens HTML possèdent certains inconvénients, comme :

- un lien ne peut pointer que vers un document unique,
- aucun historique autre que celui proposé par les navigateurs (*forward* et *back*) n'est accessible,
- les liens sont mono-directionnels, il n'y a aucune reconnaissance du document source d'où le lien provient.

XLink sert avant tout pour les interactions entre documents XML. Il permet d'effectuer des liens simples ou étendus (multisources, multicibles, externes) et des annotations (ressources contenant d'autres liens). De plus, n'importe quel élément peut devenir un lien, et grâce au XPointer, on peut indexer des positions arbitraires d'un document XML.
On trouvera un exemple de lien étendu XLink dans le tableau A.2.

A.6 RDF (*Resource Description Framework*)

RDF (*Resource Description Framework*)est un cadre de description et d'échange des métadonnées : quelque soit le format utilisé, RDF permet de rendre plus efficace le traitement automatisé des informations du Web, en fédérant les vocabulaires et syntaxes de description des métadonnées existantes dans un cadre commun. RDF est piloté par le W3C et est largement influencé par le Dublin Core. RDF est une recommandation du W3C depuis 1999 [123].
RDF permet de rendre plus "intelligente" l'information nécessaire aux moteurs de recherche et, plus généralement, nécessaire à tout outil informatique analysant de façon automatisée des pages Web. RDF se propose de définir un cadre de définition de métadonnées, sans se prononcer plus en avant sur la nature des métadonnées elles-mêmes. RDF est donc un métalangage spécialisé dans les métadonnées.
De ce métalangage, il sera possible de définir des langages de description de données : ce sera l'objectif de RDF Schema.
Un autre objectif de RDF est de fédérer les vocabulaires et syntaxes de description de méta-données existantes dans un cadre commun. Cela ne veut pas dire qu'il s'agit de définir LE modèle de métadonnées, mais plutôt de permettre à chaque modèle de s'insérer harmonieusement dans les méta-données décrivant une ressource particulière. Dans ce cadre, RDF Schema permettra de mieux contrôler les méta-données au regard de leur modèle.
RDF est conçu pour être indépendant et interchangeable. Il est utile pour la recherche d'information (pour donner aux outils de recherche de plus grandes possibilités), pour le catalogage (puisqu'il décrit le contenu d'un document et les rapports qu'il a avec les divers contenus d'un site Web), et pour le partage et l'échange de connaissances, via des agents logiciels intelligents.
La force de RDF est de ne pas se prononcer sur le sujet et de laisser aux personnes définissant leurs métadonnées le choix du(des) vocabulaire(s) utilisé(s). Ainsi, il serait, par exemple, possible de définir plusieurs propriétés "créateur" : une qui soit compatible avec le Dublin Core, spécification de métadonnées extrêmement généraliste et, une qui soit compatible avec un modèle privé, recensant tous les créateurs dans une base de données ad hoc. Un fragment RDF s'écrirait alors comme présenté dans le tableau A.3 :

A.7 Les vocabulaires métier

Autour d'XML, il existe aussi un certain nombre de vocabulaires métier (pour lesquels la DTD est fixée) proposés par des groupes de travail spécialisés. Parmi eux on peut citer :

```
<rdf :RDF
xmlns :rdf="http ://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns :dc="http ://purl.org/metadata/dublin_core#"
xmlns :mutu="http ://www.mutu-xml.org/modèles/meta">
<rdf :description about="http ://www.mutu-xml.org">
<dc :creator>projet MUTU-XML</dc :creator>
<mutu :créateur nom-en-base="MUTU-XML"/>
</rdf :description>
</rdf :RDF>
```

TAB. A.3 – Exemple d'écriture d'un fragment RDF

- **MathML** (*Mathematical Markup Language*) : langage de notation mathématique sur le web ;
- **PGML** (*Precision Graphics Markup Language*), qui décrit des structures de données graphiques complexes avec les primitives du langage Postscript. Il permet la conversion de documents aux formats *ps* et *pdf* en XML ;
- **SVG** (*Scalable Vector Graphic*) pour créer des graphiques en 2D,
- **SMIL** (*Synchronized Multimedia Integration Language*), pour la création multimédia. Il spécifie comment et quand des éléments multimédia peuvent apparaître dans une page web. Par exemple on peut dire que sur la page le texte apparaît suivi d'une série d'images qui sont accompagnées d'une musique. Il est là pour ajouter un aspect temporel aux pages Web. Il permet de contrôler la position dans l'espace et dans le temps des objets ;
- **CDF** (*Channel Definition Format*), utilisé par Microsoft pour décrire le contenu Active Channel. Une chaîne délivre des informations directement à l'utilisateur en utilisant la technologie push d'un serveur (envoi de contenus web à des utilisateurs sans que ceux-ci aient besoin d'accéder spécifiquement au site). Les chaînes fournissent des informations récentes aux utilisateurs qui peuvent sélectionner le contenu Web qu'ils souhaitent recevoir ;
- **VML** (*Vector Markup Language*) : langage de balisage d'information graphique vectorielle ;
- **WML** (*Wireless Markup Language*) : langage de balisage pour l'internet mobile ;
- **AML** (*Astronomical Markup Language*) : langage décrivant les différents types de données utilisées en astronomie ;
- **CML** (*Chemical Markup Language*), pour la publication Internet des formules chimiques, de molécules, des équations,.. ;
- **MusicML** pour éditer des partitions musicales ;

Toutes ces technologies gravitant autour d'XML peuvent être utiles dans un contexte de recherche d'information. Les espaces de noms et les XML Schema peuvent être utilisés pour préciser ou extraire la sémantique des différentes balises, XLink et XPointer permettent d'utiliser les liens entre éléments dans la recherche d'éléments pertinents (on pourra par exemple adapter et améliorer des techniques comme celles du PageRank [29] ou de HITS [113] utilisées dans la recherche d'information dans des documents HTML) et enfin RDF permet d'extraire les balises ayant un sémantique importante dans les documents, et par là même de retrouver plus aisément de l'information pertinente à des requêtes données.

Bibliographie

[1] S. Abiteboul. Querying semi-structured data. In *International Conference on Database Theory (ICDT), Delphi, Greece*, pages 1–18, 1997.

[2] S. Abiteboul, I. Manolescu, B. Nguyen, and N. Prada. A test plateform for the INEX heterogeneous track. In *Pre-proceedings of INEX 2004, Dagstuhl, Allemagne*, pages 177–182, 2004.

[3] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J.-L. Wiener. The Lorel query language for semi-structured data. *International Journal on Digital Libraries*, 1(1) :pages 68–88, 1997.

[4] M. Abolhassani and N. Fuhr. Applying the divergence from randomness approach for content-only search in XML documents. In *Proceedings of ECIR 2004, Sunderland*, pages 409–419, 2004.

[5] G. Adamson and J. Boreham. The use of an association measure based on character structure to identify semantically related pairs of words and document titles. *Information Storage and Retrieval*, 10 :pages 253–60, 1974.

[6] S. Adler. eXtensible Stylesheet Language (XSL), version 1.0. Technical report, World Wide Web Consortium (W3C),W3C Recommendation, october 2001.

[7] J. Allan, J. Callan, M. Sanderson, J. Xu, and S. Wegmann. INQERY at TREC-7. In *Proceedings of TREC-7*, pages 201–216, 1998.

[8] S. Amer-Yahia, C. Botev, and J. Shanmugasundaram. Texquery : A full-text search extension to Xquery. In *Proceedings of WWW 2004*, 2004.

[9] A.Moffat, R. Sacks-Davis, R. Wilkinson, and J. Zobel. Retrieval of partial documents. In *Proceedings of TREC-2*, 1993.

[10] J. Anderson and J. Pérez-Carballo. The nature of indexing : How humans and machines analyze messages and texts for retrieval : Part II : Machine indexing, and the allocation of human versus machine effort. *Information Processing and Management*, 37 :pages 255–277, 2001.

[11] J. André. Balises, structures et TEI. *Cahiers GUTenberg*, (24), juin 1996.

[12] V. N. Anh and A. Moffat. Compression and an ir approach to XML retrieval. In *Proceedings of INEX 2002 Workshop, Dagstuhl, Germany*, 2002.

[13] ApacheXindice. The apache XML project. http ://xml.apache.org/ xindice/ index.html.

[14] R. Attar and A. Fraenkel. Local feedback in full-text retrieval systems. *Journal of the ACM*, 24(3) :pages 397–417, 1977.

[15] S. Azagury, M. Factor, Y. Maarek, and B. Mandler. A novel navigation paradigm for XML repositories. *Journal of the American Society for Information Science and Technology (JASIST)*, 53(6) :pages 515–525, 2002.

[16] R. Baea-Yates and B. Riberto-Neto. *Modern Information Retrieval*. New-York : ACP Press, Addison-Wesley, 1999.

[17] J.-P. Balpe, A. Lelu, and I. Saleh. *Hypertextes et hypermédias : Réalisations, outils et méthodes*. Paris : Hermès, 1995.

[18] N. J. Belkin and W. Croft. Information filtering and information retrieval : two sides of the same coin ? *Communications of the ACM*, 35(12), December 1992.

[19] N. J. Belkin, R. Oddy, and H. Brooks. Ask for information retrieval : Part I background and theory. *Journal of Documentation*, 38(2) :pages 61–71, 1982.

[20] P. Bohannon, J. Freire, P. Roy, and J. Simeon. From XML schema to relations : A cost-based approach to XML storage. In *Proceedings of the 18th International Conference on Data Engineering (ICDE), San Jose, CA, USA*. Morgan Kaufmann, 2002.

[21] G. Bordogna and G. Pasi. Flexible querying of WEB documents. In *Proceddings of SAC 2002, Madrid, Spain*, pages 675–680, 2002.

[22] M. Boughanem. *Systèmes de recherche d'informations : d'un modèle classique à un modèle connexioniste*. PhD thesis, Thèse de l'Université Paul Sabatier de Toulouse, 1992.

[23] M. Boughanem, C. Chrisment, and C. Soule-Dupuy. Query modification based on relevance backpropagation in adhoc environment. *Information Processing and Management*, 35 :pages 121–139, 1999.

[24] M. Boughanem, T. Dkaki, J. Mothe, and C. Soule-Dupuy. Mercure at TREC-7. In *Proceedings of TREC-7*, 1998.

[25] M. Boughanem, W. Kraaij, and J.-Y. Nie. Modèles de langue pour la recherche d'information. In *Les systèmes de recherche d'informations*, pages 163–182. Hermes-Lavoisier, 2004.

[26] N. Bradley. *The XML Companion*. Addison-Wesley Professional Publisher, 2001.

[27] D. Braga, A. Campi, E. Damiani, P. Lanzi, and G. Pasi. FXpath : Flexible querying of XML documents. In *Proceedings of Eurofuse 2002*, 2002.

[28] S. Briet. *Qu'est ce que la documentation ?* Paris : EDIT, 1951.

[29] S. Brin, L. Page, R. Motwani, and T. Winograd. The pagerank citation ranking : Bringing order to the web. Technical report, Stanford Digital Library Technologies Project, 1998.

[30] A. Brini and M. Boughanem. Relevance feedback : Introduction of partial assessments for query expansion. In *EUSFLAT 2003. , Zittau, Germany.*, pages 67–72, 10-12 septembre 2003.

[31] M. K. Buckland. What is a document ? *Journal of the American Society of Information Science*, 48(9) :pages 804–809, september 1997.

[32] M. K. Buckland. What is a digital document ? *Document Numérique*, 2(2) :pages 221–230, 1998.

[33] P. Buneman, S. Davidson, G. Hillebrand, and D. Suciu. A query language and optimization techniques for unstructured data. In *Proceedings of ACM-SIGMOD International Conference on Management of Data, Montréal*, pages 505–516, 1996.

[34] J. Callan. Passage-level evidence in document retrieval. In *Proceedings of SIGR 1994, Dublin, Ireland*, pages 302–309, 1994.

[35] D. Carmel, Y. Maarek, M. Mandelbrot, and A. Soffer. Searching xml documents via xml fragments. In *Proceedings of SIGIR 2003*, pages 151–158, 2003.

[36] S. Carriere and R. Kazman. Webquery : Searching and visualizing the web through connectivity. *Computer Networkds and ISDN Systems*, 29, 1997.

[37] S. Ceri, S. Comai, E. Damiani, P. Fraternali, S. Paraboschi, and L. Tanca. XML-GL : A graphical language for querying and restructuring WWW data. In *Proceedings Of the 8th Int. WWW Conference, WWW8, Toronto, Canada*, May 1999.

[38] S. Chakrabarti. Integrating the document object model with hyperlinks for enhanced topic distillation and information extraction. In *Proceedings of the 10th World Wide Web Conference (WWW'01). - Hong-Kong, China*, May 2001.

[39] S. Chakrabarti, M. V. den Berg, and B. E. Dom. Focused crawling : a new approach for topic-specific resource discovery. In *Proceedings of the 8th international WWW conference, Toronto, Canada*, 1999.

[40] D. Chamberlin, J. Robie, and D. Florescu. Quilt : An XML query language for heterogeneous data sources. In *Proceedings of the 3rd Internation Workshop on World Wide Web and databases, Dallas, USA*, pages 1–25, 2000.

[41] Y. Chiaramella, P. Mulhem, and F. Fourel. A model for multimedia information retrieval. Technical report, Technical report, FERMI ESPRIT BRA 8134, University of Glasgow, 1996.

[42] T. T. Chinenyanga and N. Kushmerick. Expressive retrieval from XML documents. In *Proceedings of ACM SIGIR 2001, New-Orlean, USA*, pages 163–171, 2001.

[43] T. T. Chinenyanga and N. Kushmerick. An expressive and efficient language for XML information retrieval. *Journal of the American Society for Information Science and Technology (JASIST)*, 53(6) :pages 538–543, 2002.

[44] C. Chrisment. Caractéristiques d'XML. Cours DEA 2IL, 2005.

[45] J. Clark and S. Derose. XML Path Language (XPath) , version 1.0. Technical report, World Wide Web Consortium (W3C), W3C Recommendation, Novembre 1999.

[46] C. L. Clarke and P. L. Tilker. Multitext experiments for inex 2004. In *Pre-proceedings of INEX 2004, Dagstuhl, Allemagne*, 2004.

[47] C. W. Cleverdon, J. Mills, and M. Keen. Factors determining the performance of indexing systems. ASLIB Cranfield Research Project, Cranfield (UK), 1966.

[48] S. Cohen, Y. Kanza, Y. A. Kogan, Y. Sagiv, W. Nutt, and A. Serebrenik. EquiX - a search and query language for XML. *Journal of the American Society for Information Science and Technology*, 53(6) :pages 454–466, 2002.

[49] D. Colazzo, C. Sartiani, A. Albano, P. Manghi, G. Ghelli, L. Lini, and M. Paoli. A typed text retrieval query language for XML documents. *JASIST*, 53(6) :pages 647–488, 2002.

[50] C. Comparot-Poussier and C. Chrisment. Hyperbase pour la gestion électronique de documents techniques. *Ingénierie des Systèmes d'Information*, 2(5) :pages 533–570, 1994.

[51] W. Cooper. Expected search length : a single measure of retrieval effectiveness based on the weak ordering action of retrieval systems. *American Documentation*, 19 :pages 30–41, 1968.

[52] W. Croft, R. Cook, and D. Wilder. Providing government information on the internet : Experiences with THOMAS. U. of Mass. Technical report 95-45, 1995.

[53] C. Crouch, S. Apte, and H. Bapat. An IR approach to XML retrieval based on the extended vector model. In *Proceedings of INEX 2002 Workshop, Dagstuhl, Germany*, pages 98–99, 2002.

[54] C. Crouch, D. Crouch, Q. Chen, and S. Holz. Improving the retrieval effectiveness of very short queries. *Information Processing and Management*, 38 :pages 1–36, 2002.

[55] C. J. Crouch and B. Yang. Experiments in automatic statistical thesaurus construction. In *Proceedings of the ACM-SIGIR Conference on Research and Development in Information Retrieval , Copenhage, Denmark*, pages 77–88, 1992.

[56] J. Daniels. Cognitive models in information retrieval- an evaluation review. *Journal of Documentation*, 42(4) :pages 272–304, December 1986.

[57] L. Denoyer, G. Wisniewski, and P. Gallinari. Document structure matching for heterogeneous corpora. In *Proceedings of XML and IR workshop, SIGIR 2004, Sheffield, England*, 2004.

[58] S. Derose, E. Maler, and D. Orchard. XML Linking Language (XLink), version 1.0. Technical report, World Wide Web Consortium (W3C),W3C Recommendation, juin 2001.

[59] A. Deutsch, M. F. Fernandez, and D. Suciu. Storing semistructured data with STORED. In A. Delis, C. Faloutsos, and S. Ghandeharizadeh, editors, *Proceedings ACM SIGMOD International Conference on Management of Data, Philadelphia, Pennsylvania, USA*, pages 431–442, June 1999.

[60] V. Dignum and R. van Zwol. Guidelines for topic development in heterogeneous collections. Guidelines of INEX 2004, 2004.

[61] M. Dubinko, S. Schnitzenbaumer, M. Wedel, and D. Ragget. Xforms requirements. Technical report, World Wide Web Consortium (W3C), W3C Working draft, 2000.

[62] D. Egnor and E. Lord. XYZFind : Searching in context with XML. In *Proceedings of ACM SIGIR 2000 Workshop on XML and IR, Athens*, pages 69–78, 2000.

[63] E-XMLMedia XMLizer. http ://www.e-xmlmedia.fr/site-francais/produits-xmlizer.htm.

[64] D. C. Fallside. XML Schema. Technical report, World Wide Web Consortium (W3C),W3C Recommendation, 2001.

[65] M. Fernandez, D. Florescu, A. Levy, and D. Suciau. A query language for a web site management system. *SIGMOD Record*, 26(3) :pages 4–11, September 1997.

[66] M. Fernandez, A. Malhotra, J. Marsh, M. Nagy, and N. Walsh. XQuery 1.0 and XPath 2.0 data model. Technical report, World Wide Web Consortium (W3C), W3C Working Draft, may 2003.

[67] D. Florescu, D. Kossman, and I. Manolescu. Integrating keywords search into XML query processing. In *Proceedings of BDA'2000, Blois/France*, pages 265–280, Octobre 2000.

[68] D. Florescu and D. Kossmann. Storing and querying XML data using an RDMBS. *IEEE Data Engineering Bulletin*, 22(3) :pages 27–34, 1999.

[69] C. Fox. *Lexical analysis and stoplists*, pages 102–130. Frakes W B, Baeza-Yates R (eds) Prentice Hall, New jersey, 1992.

[70] W. B. Frakes. *Stemming Algorithms*, pages 131–160. Frakes W B, Baeza-Yates R (eds) Prentice Hall, New jersey, 1992.

[71] N. Fuhr. Information retrieval, lecture notes. Technical report, Universität Dortmund, Fachbereich Informatik, 2002.

[72] N. Fuhr. Metrics working group report. INEX 2004 Workshop, Dagstuhl, Germany, 2004.

[73] N. Fuhr. Information retrieval. Vorlesung, SommerSemester 2005, 2005.

[74] N. Fuhr, N. Govert, G. Kazai, and M. Lalmas. Proceedings of the first workshop of the initiative for the evaluation of XML retrieval (INEX 2002), 2002.

[75] N. Fuhr and K. Grossjohann. XIRQL : a query language for information retrieval in XML documents. In *In Proceedings of SIGIR 2001, Toronto, Canada*, 2003.

[76] N. Fuhr, M. Lalmas, and S. Malik. INEX 2003 workshop proceedings, 2003.

[77] N. Fuhr, M. Lalmas, S. Malik, and Z. Szlavik. INEX 2004 workshop pre-proceedings, 2004.

[78] N. Fuhr and T. Rölleke. HySpirit - a probabilistic inference engine for hypermedia retrieval in large databases. In *Proceedings of the 6th International Conference on Extending Database Technology (EDBT), Valencia, Spain*, 1998.

[79] M. Fuller, E. Mackie, R. Sacks-Davis, and R. Wilkinson. Structural answers for a large structured document collection. In *Proceedings of ACM SIGIR 1993, Pitthsburgh*, pages 204–213, 1993.

[80] G. Furnas, S. Deerwester, S. Dumais, T. Landauer, R. Harshman, L. Streeter, and K. Lochbaum. Information retrieval using a singular value decomposition model of latent semantic structure. In *Proceedings of ACM SIGIR 88*, pages 465–480, 1988.

[81] G. Furnas and T. Landauer. The vocabulary problem in a human-system communication : an analysis and a solution. *Communication of the ACM*, 1987.

[82] G. Gardarin. *XML : Des bases de données aux services Web*. Dunod - 01 Informatique, Paris 2002, 2002.

[83] G. Gardarin. Introduction à xml. Cours, disponible sur http ://perso.wanadoo.fr/georges.gardarin/, 2005.

[84] N. Gövert, M. Abolhassani, N. Fuhr, and K. Grossjohann. Content-oriented XML retrieval with hyrex. In *Proceedings of the first INEX Workshop, Dagstuhl, Germany*, 2002.

[85] S. Geva. Gpx - gardens point xml information retrieval at inex 2004. In *Pre-Proceedings of INEX 2004, Dagstuhl, Germany*, pages 110–117, 2003.

[86] S. Geva and L. Murray. Xpath inverted file for information retrieval. In *Proceedings of INEX 2003, Dagstuhl, Germany*, 2003.

[87] GoXml DB de XML global.

[88] C. Goldfarb. *The SGML Handbook*. Oxford University Press, Oxford, 1990.

[89] T. Grabs. *Storage and Retrieval of XML Documents within a Cluster of Database Systems*. PhD thesis, Ecole Polytechnique Fédérale de Zürich, 2003.

[90] T. Grabs and H.-J. Scheck. Flexible information retrieval from xml with PowerDB XML. In *Proceedings in the First Annual Workshop for the Evaluation of XML Retrieval (INEX)*, pages 26–32, December 2002.

[91] K. Grossjohann. Query formulation and result visualization for XML retrieval. In *Proceedings of the SIGIR 2000 Workshop on XML and Information Retrieval, Athens, Greece*, 2000.

[92] P. Grosso, E. Maler, J. Marsh, and N. Walsh. XML Pointer Language (XPointer). Technical report, World Wide Web Consortium (W3C),W3C Recommendation, march 2003.

[93] T. Grust. Accelerating XPath location steps. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data, Madison, Wisconsin, USA*. In M. J. Franklin, B. Moon, and A. Ailamaki, editors, ACM Press, 2002.

[94] A. Gutierrez, R. Motz, and D. Viera. Building databases with information extracted from web documents. In *Proceedings XX international conference of the Chilean computer sciences society*, pages 41–49, 2000.

[95] M. Hearst. TextTiling : A quantitative approach to discourse segmentation. *Computational Linguistics*, 23(1) :pages 33–64, mars 1997.

[96] D. Hiemstra. A linguistically motivated probabilistic model of information retrieval. In *Proceedings of the 2nd European Conference on Research and Advanced Technology for Digital Libraries (ECDL)*, pages 569–584, 1998.

[97] G. Huck, I. Macherius, and P. Fankhauser. PDOM : Lightweigt persistency support for the document object model. In Succeeding with Object Databases,John Wiley, 2000.

[98] IPEDO XML database de IPEDO. http ://www.ipedo.com/html/ipedo-xml-database.html.

[99] TextML de IXIA SOFT. http ://www.ixiasoft.com.

[100] H. Jang, Y. Kim, and D. Shin. An effective mechanism for index update in structured documents. In *Proceedings ACML CIKM, Kansas City*, pages 383–390, 1999.

[101] B. Johnson and B. Schneiderman. Tree-maps : a space filling approach to the visualization of hierarchical information structures. Technical report, Technical report CS-TR-2657, University of Maryland, Computer Science Department, april 1991.

[102] K. Järvelin and J. Kekäläinen. Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems*, 20(4) :pages 422–446, 2002.

[103] V. Kakade and P. Raghavan. Encoding XML in vector spaces. In *Proceedings of ECIR 2005, Saint Jacques de COmpostelle, Spain*, 2005.

[104] J. Kamps, M. de Rijke, and B. Sigurbjornsson. Length normalization in XML retrieval. In *Proceedings of SIGIR 2004, Sheffield, England*, pages 80–87, 2004.

[105] C.-C. Kanne and G. Moerkotte. Efficient storage of XML data. In *In Proceedings of the 16th International Conference on Data Engineering, San Diego, California, USA*, page 198, 2000.

[106] M. Kaszkiel and J. Zobel. Passage retrieval revisited. In *Proceedings of SIGIR 1997, Philadelphia, USA*, pages 178–185, 1997.

[107] G. Kazai. Report of the INEX 2003 metrics working group. In *Proceedings of INEX 2003, Dagstuhl, Germany*, December 2003.

[108] G. Kazai, M. Lalmas, and A. de Vries. Reliability tests for the XCG and inex-2002 metrics. In *Pre-Proceedings of INEX 2004*, pages 33–39, december 2004.

[109] G. Kazai, M. Lalmas, and A. P. de Vries. The overlap problem in content-oriented XML retrieval evaluation. In *Proceedings of SIGIR 2004, Sheffield, England*, pages 72–79, July 2004.

[110] G. Kazai, M. Lalmas, N. Fuhr, and N.Gövert. A report on the first year of the INitiative for the Evaluation of XML retrieval (INEX 2002). *JASIST*, 55(6) :pages 551–556, april 2004.

[111] G. Kazai, M. Lalmas, and T. Roelleke. Focused document retrieval,. In *9th International Symposium on string processing and information retrieval, Lisbon, Portugal*, September 2002.

[112] K. Kise, M. Junker, A. Dengel, and K. Matsumoto. Experimental evaluation of passage-based document retrieval. IEEE, 2001.

[113] J. M. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46(5) :pages 604–632, September 1999.

[114] T. Kohonen. *Self-organization and associative memory*. Springer Verlag, 1989.

[115] T. Kohonen, S. Kaski, K. Lagus, J. Salojarvi, J. Honkela, V. Paareto, and A. Saarela. Self organization of massive text document collection. *IEEE Transactions on Neural Networkds, Special Issue on Neural Networks for Data Mining and Knowledge Discovery*, pages pages 574–585, 2000.

[116] R. Korhage. *Information storage and retrieval*. John Wiley and Sons, Inc., 1997.

[117] E. Kotsakis. Structured information retrieval in XML documents. In *Proceedings of the ACM Symposium on applied computing*, 2002.

[118] A. Kristensen. Formsheets and the XML forms language. In *Proceedings of WWW9, Amsterdam*, pages 1189–1201, 1999.

[119] K. Kwok, L. Grunfeld, and M. Chan. TREC-8 adhoc, query and filtering track experiments using PIRCS. In *Proceedings of TREC-8*, 2000.

[120] M. Lalmas. Dempster-shafer's theory of evidence applied to structured documents : modeling uncertainty. In *Proceedings of SIGIR'97, Philadelphia, USA*, pages 110–118, 1997.

[121] M. Lalmas, N. Fuhr, S. Malik, Z. Szlavik, and V. huyen Trang. Some statistics about INEX 2004. INEX 2004 Workshop, Slides available on http ://inex.is.informatik.uni-duisburg.de :2004/workshop.html, '2004.

[122] R. R. Larson. Cheshire II at INEX : Using a hybrid logistic regression and boolean model for XML retrieval. In *Proceedings of INEX 2002 Workshop , Dagstuhl, Allemagne*, pages 2–7, 2002.

[123] O. Lassila and R. R. Swick. Resource Description Framework (RDF) model and syntax specification. Technical report, World Wide Web Consortium (W3C),W3C Recommendation, Februar 1999.

[124] Y. Lee, S. Yoo, and K. Yoon. Index structures for structured documents. In *In Proc. ACM Workshop on XML and IR, Bethesda*, pages 91–99, 1996.

[125] A. Levy, M. Fernandez, D. Suciu, D. Florescu, and A. Deutsch. XML-QL : A query language for XML. Technical report, World Wide Web Consortium technical report, Number NOTE- xml-ql-19980819, 1998.

[126] Q. Li and B. Moon. Indexing and querying XML data for regular path expressions. In *Proceedings of the 27th VLDB Conference, Roma, Italy*, 2001.

[127] J. A. List, V. Mihajlovic, A. Vries, G. Ramirez, and D. Hiemstra. The TIJAH XML-IR system at Inex 2003. In *Proceedings of INEX 2003, Dagstuhl, Germany*, 2003.

[128] Y. Loiseau, H. Prade, and M. Boughanem. Qualitative pattern matching with linguistic terms . *AI Communication* , 17(1) :pages 25–34, 2004.

[129] H. Luhn. A statistical approach to mechanized encoding and searching of literary information. *IBM*, 1(4) :pages 309–317, 1957.

[130] R. W. Luk, H. Leong, T. S. Dillon, A. T. Shan, W. B. Croft, and J. Allan. A survey in indexing and searching XML documents. *Journal of the American Society for Information Science and Technology*, 53(3) :pages 415–435, 2002.

[131] S. Malik, T. Tombros, and B. Larsen. Hyrex for INEX iTrack. In *Preproceedings of INEX 2004, Dagstuhl, Germany*, pages 264–269, 2004.

[132] J. Maniez and E. de Grolier. A decade of research in classification, 1991.

[133] M. Maron and J. Kuhns. On relevance, probabilistic indexing and information retrieval. *Journal of the Association for Computing Machinery*, 7 :pages 216–244, 1960.

[134] M. Marx, J. Kamps, and M. de Rijke. The university of amsterdam at INEX 2002. In *INEX 2002 Workshop Proceedings, Dagstuhl, Germany*, pages 23–28, 2002.

[135] Y. Mass and M. Mandelbrod. Retrieving the most relevant XML components. In *Proceedings of INEX 2003, Dagstuhl, Germany*, 2003.

[136] Y. Mass and M. Mandelbrod. Component ranking and automatic query refinement for XML retrieval. In *Proceedings of INEX 2004*, pages 134–140, 2004.

[137] Y. Mass, M. Mandelbrod, E. Amitay, D. Carmel, Y. Maarek, and A. Soffer. JuruXML- an XML retrieval system at INEX'02. In *Proceedings of INEX 2002, Dagstuhl, Germany*, pages 73–80, 2002.

[138] M. Melucci. Passage retrieval : A probabilistic technique. *Information Processing and Management*, 34(1) :pages 43–68, 1998.

[139] A. Michard. *XML - Langage et Application*. Paris : Eyrolles, 1999.

[140] L. Mignet, D. Barbosa, and P. Veltri. The XML web : A first study. In *Proceedings of WWW2003, Budapest, Hungary*, 2003.

[141] E. Mittendorf and P. Schäuble. Document passage retrieval based on hidden markov models. In *Proceedings of the 17th ACM SIGIR Conference, Dublin, Ireland*, pages 318–327, 1994.

[142] S. Mizzaro. Relevance, the whole (hi) story. *Journal of the American society for information science*, 48(9) :pages 810–832, 1997.

[143] J. Mothe. Recherche et exploration d'information, découverte de connaissance pour l'accès à l'information. HDR, Université Paul Sabatier de Toulouse, 2000.

[144] N.Gövert, G. Kazai, N. Fuhr, and M. Lalmas. Evaluating the effectiveness of content-oriented XML retrieval. Technischer Bericht, University of Dortmund, Computer Science 6, 2003.

[145] Y. Ogasa, T. Morita, and K. Kobayashi. A fuzzy document retrieval system using the keyword connection matrix and learning method. *Fuzzy sets and systems*, 39 :pages 163–179, 1991.

[146] Y. Ogawa, M. Hiroko, N. Masumi, and H. Sakiko. Structuring and expanding queries in the probabilistic model. In *Proceedings of TREC-8*, 1999.

[147] P. Ogilvie and J. Callan. Using language models for flat text queries in XML retrieval. In *Proceedings of INEX 2003 Workshop, Dagstuhl, Germany*, pages 12–18, December 2003.

[148] P. Ogilvie and J. Callan. Hierarchical language model for xml component retrieval. In *Proceedings of INEX 2004 Workshop, Dagstuhl, Germany*, 2004.

[149] C. P. Paice. Soft evaluation of boolean search queries in information retrieval systems. *Information Technology : Research and Development*, 3(1) :pages 33–42, 1984.

[150] J. Pearl. *Probabilistic reasoning in Intelligent Systems : Networks of Plausible Inference*. Morgan Kaufmann Publishers, Inc., 1988.

[151] J. Pehcevsji, J. A. Thom, and A.-M. Vercoustre. Hybrid xml retrieval revisited. In *Pre-Proceedings of INEX 2004, Dagstuhl, Germany*, pages 90–97, 2004.

[152] J. Pehcevski, J. A. Thom, and A.-M. Vercoustre. Hybrid xml retrieval : combining information retrieval and native xml database. *Journal of Information Retrieval, special issue on INEX (accepted for publication)*, 2004.

[153] J. Picard and J. Savoy. Searching and classifying the web using hyperlinks : a logical approach. In *23th European Colloquium on Information Retrieval Research (ECIR)*, 2001.

[154] B. Piwowarski. *Techniques d'apprentissage pour le traitement d'information structurées : application à la recherche d'information.* PhD thesis, Paris : Université Paris 6, 2003.

[155] B. Piwowarski. Working group report : the assessment tool. In *Proceedings of INEX 2003, Dagstuhl, Germany*, pages 181–183, December 2003.

[156] B. Piwowarski, G.-E. Faure, and P. Gallinari. Bayesian networks and INEX. In *Proceedings in the First Annual Workshop for the Evaluation of XML Retrieval (INEX)*, December 2002.

[157] B. Piwowarski and P. Gallinari. Expected Ratio of Relevant Units : a measure for structured information retrieval. In *Proceedings of INEX 2003, Dagstuhl, Germany*, pages 158–166, December 2003.

[158] B. Piwowarski and M. Lalmas. Interface pour l'évaluation de systèmes de recherche sur des documents XML. In *Actes de CORIA 2004, Toulouse, France*, pages 109–121, 2004.

[159] J. Ponte and W. Croft. A language modeling approach to information retrieval. In *Proceedings of the 21st ACM conference on research and development in information retrieval (SIGIR 98)*, 1998.

[160] M. F. Porter. An algorithm for suffix stripping. Program 14, 1980.

[161] Y. Qiu and H. Frei. Concept based query expansion. In *Proceedings of the 16th ACM SIGIR Conference on Research and Development in Information Retrieval, Pittsburgh, PAA, USA*, pages 160–169, 1993.

[162] V. V. Raghavan, S. J. Gwang, and P. Bollmann. A critical investigation of recall and precision as measures of retrieval system performance. *ACM Transactions on Information Systems*, 7(3) :pages 205–229, july 1989.

[163] B. A. Ribeiro-Neto and R. Muntz. A belief network model for IR. In *Proceedings Of the 19th annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Zurich, Suisse*, pages 253–260, 1996.

[164] S. Robertson. The probability ranking principle in IR. *Journal of Documentation*, 33(4) :pages 294–304, 1977.

[165] S. Robertson. On term selection for query expansion. *Revue de Documentation*, 46 :pages 359–364, 1990.

[166] S. Robertson, S. Walker, S. Jones, and M. H.-B. andM. Gatford. Okapi at TREC 3. In *Proceedings of the 3rd Text REtrieval Conference (TREC-3)*, pages 109–126, 1994.

[167] S. E. Robertson and S. Walker. Some simple effective approximations to the 2-Poisson model for probabilistic weighted retrieval. In *Proceedings of SIGIR 1994*, pages 232–241, 1994.

[168] J. Robie, J. Lapp, and D. Schach. Xml query language (XQL). In *Proceedings of W3C QL'98 (Query Languages 98), Massachussets*, 1998.

[169] J. Rocchio. *Relevance feedback in information retrieval*. Prentice Hall Inc., Englewood Cliffs, NJ, 1971.

[170] T. Roelleke, M. Lalmas, G. Kazai, J. Ruthven, and S. Quicker. The accessibility dimension for structured document retrieval. In *Proceedings of ECIR 2002*, 2002.

[171] D. Rumelhart, G. Hinton, and R. Williams. Learning internal representations by error propagation. In *In Parallel distributed proceedings, vol.2, D. Rumelhart, J. Mc Clelland eds. MIT Press*, 1986.

[172] K. B. Sall. XML family of specifications. http ://mywebpages.comcat.net/kensall/big-picture. From the Addison Wesley book : XML family of specification, a practical guide, 2002.

[173] G. Salton. *The SMART retrieval system : Experiments in automatic document processing*. Prentice Hall, 1970.

[174] G. Salton. A comparison between manual and automatic indexing methods. *Journal of American Documentation*, 20(1) :pages 61–71, 1971.

[175] G. Salton, J. Allan, and C. Buckley. Approaches to passage retrieval in full text information systems. In *Proc. of SIGIR'93, Pittsburgh, PA*, 1993.

[176] G. Salton, E. Fox, and H. Wu. Extended boolean information retrieval. *Communications of the ACM*, 31(2) :1002–1036, November 1983.

[177] G. Salton and M. McGill. *Introduction to modern information retrieval*. McGraw-Hill Int. Book Co, 1984.

[178] G. Salton, A. Singhal, C. Buckely, and M. Mitra. Automatic text decomposition using text segments and text themes. In *HyperText'96, Washington DC, USA*, pages 53–65, 1996.

[179] K. Sauvagnat and M. Boughanem. Etat de l'art : Recherche d'information dans des documents XML. Technical report, Rapport Interne IRIT, IRIT/ 2004-1-R, janvier 2004.

[180] K. Sauvagnat and M. Boughanem. The impact of leaf nodes relevance values evaluation in a propagation method for XML retrieval. In R. Baeza-Yates, Y. Marek, T. Roelleke, and A. P. de Vries, editors, *Proceedings of the 3rd XML and Information Retrieval Workshop, SIGIR 2004, Sheffield, England*, pages 13–22, July 2004.

[181] K. Sauvagnat and M. Boughanem. Le langage de requête XFIRM pour les documents XML : De la recherche par simples mots-clés à l'utilisation

de la structure des documents. In *Proceedings of Inforsid 2004, Biarritz, France*, may 2004.

[182] K. Sauvagnat and M. Boughanem. Using a relevance propagation method for adhoc and heterogeneous tracks at inex 2004. In *Pre-proceedings of INEX 2004, Dagstuhl, Allemagne*, 2004.

[183] K. Sauvagnat and M. Boughanem. A la recherche de noeuds informatifs dans des corpus de documents XML - ou pourquoi on a toujours besoin de plus petit que soi... In *Actes de CORIA 05, Grenoble, France*, 2005.

[184] K. Sauvagnat, M. Boughanem, and C. Chrisment. Searching XML documents using relevance propagation. In A. Apostolico and M. Melucci, editors, *SPIRE 04 , Padoue, Italie*, pages 242–254. Springer, 6-8 October 2004.

[185] K. Sauvagnat, G. Hubert, J. Mothe, and M. Boughanem. IRIT at INEX 03. In *Proceedings of INEX 2003 Workshop, Dagstuhl, Germany*, December 2003.

[186] T. Schileder and H. Meuss. Querying and ranking XML documents. *Journal of the American Society for Information Science and Technology*, 53(6) :pages 489–503, 2002.

[187] F. Sèdes. Bases documentaires - hyperbases proposition d'un modèle générique et contribution à la spécification d'un langage pour l'intégration et la manipulation d'informations semi-structurées. HDR, Décembre 1998.

[188] G. Shafer. *A mathematical theory of evidence*. Princeton, NJ : Princeton University Press, 1976.

[189] W. Shaw, R. Burgin, and P. Howell. Performance standards and evaluations in IR test collections : Cluster-based retrieval models. *Information Processing and Management*, 33(1) :pages 1–14, 1997.

[190] D. Shin, H. Jang, and H. Jin. BUS : an effective indexing and retrieval scheme in structured documents. In *Proceedings of digital libraries, Pittsburgh*, pages 235–243, 1998.

[191] B. Sigurbjörnsson, M. de Rijke, and J. Kamps. The university of Amsterdam at INEX 2004. In *Pre-Proceedings of INEX 2004 workshop, Dagstuhl, Germany*, december 2004.

[192] B. Sigurbjörnsson, J. Kamps, and M. de Rijke. An element-based approach to XML retrieval. In *Proceedings of INEX 2003 workshop, Dagstuhl, Germany*, december 2003.

[193] B. Sigurbjörnsson, B. Larsen, M. Lalmas, and S. Maalik. INEX04 guidelines for topic development. In *Pre-proceedings of INEX 2005, Dagstuhl, Allemagne*, pages 212–218, 2004.

[194] A. Singhal, C. Buckley, and M. Mitra. Pivoted document length normalization. In *SIGIR '96 : Proceedings of the 19th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 21–29. ACM Press, 1996.

[195] A. Singhal, G. Salton, M. Mitra, and C. Buckley. Document length normalization. *Information Processing and Management*, 32(5) :pages 619–633, 1996.

[196] Tamino de SOFTWARE A.G. http ://www.softwareag.com/tamino/.

[197] K. Sparck-Jones, S. Walker, and S. Robertson. A probabilistic model for information retrieval/development and comparative experiments, part 1 and 2. *Information Processing and Management*, 36(6) :pages 779–840, 2000.

[198] Z. Szalik and T. Roelleke. Building and experimenting with a heterogeneous collection. In *Pre-proceedings of INEX 2004, Dagstuhl, Allemagne*, pages 24–32, 2004.

[199] X. Tannier, J.-J. Girardot, and M. Matthieu. Utilisation de la langue naturelle pour l'interrogation de documents structurés. In *Actes de CORIA 05, Grenoble, France*, 2005.

[200] H. Tebri. *Formalisation et spécification d'un système de filtrage incrémental d'information*. PhD thesis, Toulouse : Université Paul Sabatier, 2004.

[201] A. Theobald and G. Weikum. The index-based XXL search engine for querying XML data with relevance ranking. In *EDBT 2002, 8th International Conference on Extending Database Technology, Prague, Czech Republic*, pages 477–495, 2002.

[202] M. Tmar. *Modèle auto-adaptatif de filtrage d'information : apprentissage incrémental du profil et de la fonction de décision*. PhD thesis, Toulouse : Université Paul Sabatier, 2002.

[203] Trec web page. http ://trec.nist.gov.

[204] A. Trotman. Searching structured documents. *Information Processing and Management*, 40 :pages 619–632, 2004.

[205] A. Trotman. Choosing document structure weights. *Information Processing and Management*, 41(2) :pages 243–264, March 2005.

[206] A. Trotman and B. Sigurbjörnsson. Narrowed extended XPath I (NEXI). In *INEX 2003 proceedings, Dagstuhl, Allemagne*, pages 219–237, December 2004.

[207] A. Trotman and B. Sigurbjörnsson. NEXI, now and next. In *INEX 2003 proceedings, Dagstuhl, Allemagne*, pages 10–15, December 2004.

[208] H. Turtle. *Inference Networks for Document Retrieval*. PhD thesis, University of Massachussetts, Amhesrt, 1991.

[209] H. Turtle and W. Croft. Inference networks for document retrieval. In *Proceedings of ACM SIGIR 90*, pages 1–24, 1990.

[210] How much information ? 2003. http ://www.sims.berkeley.edu/research/projects/how-much-info-2003/execsum.htm, 2003. UC Berkeley's School of Information Management and Systems.

[211] C. van Rijsbergen. *Information retrieval*. Butterworths, 1979.

[212] J.-N. Vittaut, B. Piwowarski, and P. Gallinari. An algebra for structured queries in bayesian networks. In *INEX 2004 Pre-proceedings, Dagstuhl, Allemagne*, pages 58–65, 2004.

[213] C. Vogt. *Adaptive combination of evidence for information retrieval*. PhD thesis, University of California, San Diego, 1999.

[214] W3C. DOM Level 1 (Document Object Model). Technical report, World Wide Web Consortium (W3C), W3C standard, october 1998.

[215] W3C. EXtensible Markup Language (XML) 1.0. Technical report, World Wide Web Consortium (W3C), Technical report, february 1998.

[216] W3C. XQuery and XPath full-text use cases. Technical report, World Wide Web Consortium (W3C), W3C working draft, fevrier 2003.

[217] S. Walker, S. Robertson, M. Boughanem, G. Jones, and K. S. Jones. Okapi at TREC-6 automatic and ad hoc, VLC, routing, filtering and QSDR. In *Proceedings of TREC-6*, pages 125–136, 1997.

[218] F. Weigel, H. Meuss, F. Bry, and K. U. Schulz. Content-aware dataguides : Interleaving IR and DB indexing techniques for efficient retrieval of textual XML data. In *Proceedings of ECIR 2004, Sunderland, UK*, pages 378–393, 2004.

[219] F. Weigel, K. U. Shulz, and H. Meuss. Ranked retrieval of structured doucments with the STerm vector space model. In *Pre-Proceedings of INEX 2004, Dagstuhl, Allemagne*, pages 126–133, 2004.

[220] H. White and K. McCain. Bibliometrics. *Annual review of Information Science and Technology*, 24 :pages 119–165, 1989.

[221] R. Wilkinson. Effective retrieval of structured documents. In *Proceedings of SIGIR 1994, Dublin, Ireland*, pages 311–317, 1994.

[222] R. Wilkinson and P. Hingston. Using the cosine mesure in a neural network for document retrieval. In *Proceedings Of the ACM SIGIR Conference on Research and Development in Information Retrieval, Chicago, USA*, pages 202–210, Oct. 1991.

[223] J. Wolff, H. Flörke, and A. Cremers. Searching and browsing collections of structural information. In *Proceedings of IEEE advances in digital libraries, Washington, 2000*, pages 141–150, 2000.

[224] S. Wong, W. Ziarko, and P. Wong. Generalized vector space model in information retrieval. In *Proceedings of the 8th ACM SIGIR Conference on Research and Development in information retrieval, New-York, USA*, pages 18–25, 1985.

[225] XPeranto de IBM. http ://www.almaden.ibm.com/software/dm/ Xperanto/ index.shtml.

[226] J. Xu, R. Weischedel, and C. Nguyen. Evaluating a probabilistic model for cross-lingual information retrieval. In *Proceedings of the ACM-SIGIR 2001*, pages 105–110, 2001.

[227] Xyleme zone server de xyleme. http ://www.xyleme.com.

[228] R. Yager. On ordered weighted averaging aggregation operators in multicriteria decision making. *IEEE Transactions on Systems, Man and Cybernetics*, 18 :pages 183–190, 1988.

[229] S. Yoo. An XML retrieval model based on structural proximities. In *INEX 2002 Workshop Proceedings, Dagstuhl, Allemagne*, pages 60–64, 2002.

[230] M. Yoshikawa, T. Amagasa, T. Shimura, and S. Uemura. XRel : A path-based approach to storage and retrieval of XML documents using relational databases. *ACM Transactions on Internet Technology*, 1(1) :pages 110–141, 2001.

[231] C. Yu and G. Salton. Precision-weighting- an effective automaic indexing method. *Journal of the ACM*, 23 :pages 76–88, 1976.

[232] L. Zadeh. Fuzzy sets. *Information and control*, 8 :pages 338–353, 1965.

[233] H. Zargayouna. Contexte et sémantique pour une indexation de documents semi-structurés. In *Actes de CORIA 04, Toulouse, France*, pages 161–178, 2004.

[234] G. Zipf. *Human Behaviour and the Principle of Least Effort*. Addison-Wesley, 1949.

[235] J. Zobel, A. Moffat, R. Wilkinson, and R. Sacks-Davis. Efficient retrieval of partial documents. *Information Processing and Management*, 31(3) :pages 361–377, 1995.