

N^o d'ordre : XXXX

THÈSE

de

L'UNIVERSITÉ PARIS-SUD

présentée en vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ PARIS-SUD

Spécialité : INFORMATIQUE

Par

AMAR-DJALIL MEZAOUR

RECHERCHE CIBLÉE DE DOCUMENTS SUR LE WEB

Soutenue le devant la commission d'examen :

M.	Yves KODRATOFF, Directeur de recherche CNRS	Examineur
M.	Alain LÉGER, Directeur de Pôle Scientifique, France Télécom R&D	Examineur
M.	Bernd AMANN, Professeur d'université, Université Pierre et Marie Curie	Rapporteur
M.	Mohand-Saïd HACID, Professeur d'université, Université Claude Bernad Lyon	Rapporteur
Mme	Marie-Christine ROUSSET, Professeur d'université, Université Paris Sud	Directrice de thèse

Laboratoire de Recherche en Informatique, U.M.R. CNRS 8623,
Université Paris-Sud, 91405 Orsay Cedex, France

Tables des Matières

I	Introduction	1
1	Mais d'abord un peu d'histoire...	1
2	Contexte de la thèse : Le web d'aujourd'hui	4
2.1	Taille et croissance du web	4
2.2	Le rôle du web	5
2.3	La recherche d'informations sur le web	5
2.4	Format des données du web	6
3	Motivation de la thèse : recherche ciblée de documents pour la construction automatique d'entrepôts de données thématiques	6
3.1	Problématique	6
3.2	Idées défendues dans cette thèse	8
3.3	Contributions	9
4	Le projet eDot : Contexte applicatif de la thèse	9
4.1	Contexte du projet	9
4.2	Intérêt du projet	10
4.3	Documents cibles de l'entrepôt « eDot »	10
4.4	Architecture fonctionnelle du projet	11
5	Plan du manuscrit	12
	 Partie I Langages de requêtes pour le web	 15
II	État de l'art	17
1	Langages de requêtes sur le contenu textuel du document	18
1.1	Langages de requêtes mots clés classiques	18
1.2	Langages de requêtes mots clés avancés	20
1.3	Langages de requêtes mots clés utilisant des expertises de domaine	23
2	Langages de requêtes de navigation web	24
2.1	WebSQL	26
2.2	W3QL	28
2.3	SQUEAL	29
3	Langages de gestion et de restructuration de sites web	30
3.1	Langages fondés sur le paradigme objet	31
3.1.1	WebOQL	32
3.1.2	PENELOPE & ULIXES	34
3.1.3	StruQL	36

4	Conclusion	37
III WeQueL : Web Query Language		41
1	Motivations	42
2	Présentation générale du langage WeQueL	43
3	Syntaxe du langage	43
3.1	Syntaxe d'une requête atomique	43
3.1.1	L'attribut d'une requête atomique	44
3.1.2	Les valeurs V_S d'une requête atomique	45
3.1.3	Le constructeur sémantique SC d'une requête atomique	45
3.1.4	Exemples de requêtes atomiques du langage WeQueL	47
3.2	Combinaison de requêtes atomiques	47
3.3	Syntaxe d'une requête WeQueL complexe	50
3.4	Syntaxe opérationnelle du langage WeQueL	52
4	Évaluation et sémantique des requêtes du langage WeQueL	55
4.1	WDM : modèle de représentation des documents du web pour le langage WeQueL	56
4.1.1	Cas particuliers	56
4.2	Évaluation d'une requête atomique	57
4.2.1	Appariement des valeurs d'une requête atomique sur une section d'un document du web	58
4.2.2	Calcul du score d'une requête atomique	59
4.2.3	Exemples d'évaluations de requêtes atomiques	60
4.3	Évaluation d'une combinaison de requêtes atomiques	64
4.4	Évaluation d'une requête WeQueL complexe	66
5	Évaluation expérimentale du langage WeQueL	68
5.1	WeQueL : l'outil de recherche thématique	68
5.1.1	L'échantillon des requêtes testées	68
5.1.2	Corpus des documents évalués	72
5.1.3	Mesures de qualité	74
5.1.4	Résultats expérimentaux	75
5.1.5	Discussion	76
5.2	WeQueL : l'outil de filtrage du projet « eDot »	77
5.2.1	Protocole expérimental	77
5.2.2	Évaluation expérimentale du filtrage	84
6	Conclusion	87
Partie II Exploration du web		89
IV Robots explorateurs du web		91
1	Définitions & Formalisation	91
1.1	Le web	91
1.2	Liens hypertextes	91
1.3	Graphe du web	93
2	Exploration automatique du web	93
2.1	Robot explorateur du web	94
2.2	Architecture d'un robot explorateur du web	94

2.2.1	La pile d'urls en attente de traitement	95
2.2.2	Le module d'application de la stratégie d'exploration	97
2.2.3	Le module de rapatriement de documents	97
2.2.4	Le module de stockage des ressources rapatriées	98
2.3	Le graphe web exploré par un robot	98
2.4	Profondeur d'un document dans le graphe du web	98
3	Déroulement d'une exploration du web	98
4	Conclusion	101
V	État de l'art	103
1	Exploration systématique du web	103
1.1	Exploration systématique en largeur d'abord	103
1.2	Exploration systématique en profondeur d'abord	105
1.3	Exploration systématique aléatoire	106
1.4	Exploration systématique par popularité d'abord	107
1.5	Évaluation des approches systématiques	108
2	Exploration par stratégie de type « le meilleur d'abord »	109
2.1	L'algorithme Fish-Search	109
2.2	L'algorithme Shark-Search	110
3	Exploration sélective du web guidée par des critères thématiques	111
3.1	Approches sélectives avec apprentissage initial	112
3.1.1	Focused Crawling de Jason Rennie & Andrew Katchies McCallum	112
3.1.2	Focused Crawling de Michelangelo Diligenti et al.	115
3.2	Exploration sélective du web sans apprentissage initial	119
3.2.1	InfoSpiders	119
3.2.2	Exploration « intelligente » du web (Intelligent crawling)	121
4	Conclusion	131
VI	Exploration sélective du web guidée par une requête WeQueL	135
1	Présentation générale de notre approche	136
2	L'estimateur <i>WEst</i>	138
2.1	Modèle de probabilité	138
2.2	Les matrices statistiques : support d'implantation de l'estimateur <i>WEst</i>	139
2.2.1	La matrice statistique de parenté	140
2.2.2	La matrice statistique de fratrie	140
2.2.3	Implantation du modèle de probabilité de l'estimateur <i>WEst</i>	140
2.3	Estimation du score d'un candidat	140
2.4	Stratégie d'exploration	141
2.5	Ajustement des estimations de <i>WEst</i>	141
2.5.1	Confirmation d'une prédiction de l'estimateur <i>WEst</i>	142
2.5.2	Infirmité d'une prédiction de l'estimateur <i>WEst</i>	142
2.6	Exemple illustratif de l'approche	143
3	Évaluation expérimentale de l'approche	145
3.1	Protocole expérimental	146
3.1.1	Documents visés	146
3.1.2	Stratégies d'exploration concurrentes	146
3.1.3	Paramètres des explorations expérimentales	148

3.1.4	Estimation du score des candidats pour notre approche	149
3.1.5	Mesure de la qualité de l'exploration	149
3.1.6	Tests de robustesse	150
3.2	Résultats expérimentaux	151
3.2.1	Risque alimentaire	151
3.2.2	Documents en-ligne de cours informatiques	152
3.2.3	Discussion	153
4	Conclusion	157

Partie III	Conclusion et perspectives	159
-------------------	-----------------------------------	------------

VII	Conclusion et perspectives	161
------------	-----------------------------------	------------

Introduction

« *But I still haven't found what I'm looking for* »
U2. The Joshua Tree – 1987

Interprétée par le célèbre groupe irlandais de rock *U2* et parue en 1987 lorsque le web n'existait pas encore, la chanson « *But I still haven't found what I'm looking for* » pourrait, par son titre, parfaitement résumer le sentiment d'un utilisateur découvrant les résultats de sa recherche documentaire via un moteur de recherche classique. Bien que ces moteurs aient considérablement amélioré l'accès aux informations contenues dans les documents du web, certains besoins ne peuvent être entièrement satisfaits faute de pouvoir les exprimer et les évaluer convenablement avec les outils de recherche proposés. De plus, l'absence d'une partie du web dans les index des moteurs de recherche ne permet pas à ces derniers de fournir toutes les réponses pertinentes à une requête donnée. À titre d'exemple, il est impossible de formuler une requête mots clés sous Google [Google site] pour rechercher des documents ayant un format particulier et satisfaisant à la fois plusieurs critères de présence de mots clés dans différentes parties d'un document.

Dans cette thèse, nous nous sommes intéressés à l'étude des techniques de recherche d'informations thématiques sur le web. À cet effet, nous proposons deux solutions complémentaires : un langage de requêtes pour le web offrant des mécanismes évolués de description des documents à rechercher (**WeQueL : Web Query Language**) et une nouvelle technique d'exploration du web guidée par une requête donnée du langage WeQueL.

Avec ces deux outils, nous visons l'amélioration qualitative et quantitative de la recherche de documents thématiques sur le web.

1 Mais d'abord un peu d'histoire...

En 1962, en pleine guerre froide, l'US Air Force demande à ses chercheurs de développer un réseau de communication militaire capable de résister à une attaque nucléaire. Le principe reposait sur une architecture décentralisée capable de garantir le fonctionnement du réseau même si une ou plusieurs machines étaient détruites. En 1969, ce projet a donné naissance à un réseau expérimental

appelé **ARPANET** (*Advanced Research Projects Agency NETwork*). ARPANET fut déployé par le département américain de défense afin de relier quatre instituts universitaires : *Stanford Institute*, *l'université de Californie à Los Angeles*, *l'université de Californie à Santa Barbara* et *l'université d'Utah*. Le réseau ARPANET utilisait alors des protocoles basiques pour assurer les échanges de données entre les quatre serveurs des universités citées.

Aujourd'hui, ARPANET n'existe plus mais est considéré comme le réseau précurseur d'internet. En effet, les développements successifs administrés sur la forme originelle d'ARPANET ont donné naissance à l'internet que tout le monde connaît aujourd'hui. Le tableau I.1 page 2 résume les dates importantes du développement de l'internet.

Date	Événement	# de serveurs reliés à internet
1969	Déploiement d'ARPANET	4
1972	Création du premier logiciel de courrier électronique	31
1973	Premières ébauches du protocole TCP/IP	35
1975	Création du protocole FTP	≈ 100
1979	USENET, premier logiciel de discussion en forum	188
1980	Tim Berners Lee met au point Enquire , premier logiciel de navigation hypertexte	≈ 200
1982	basculement intégral d'ARPANET sous le protocole TCP/IP	235
1984	Mise en place du service DNS	1.024
1989	Tim Berners Lee donne naissance au Web en définissant le langage SGML et le protocole HTTP	≈ 159.000
1990	Tim Berners Lee et Robert Cailliau présentent une version améliorée du web	313.000
1991	Premier système web mis à la disposition des physiciens du CERN à Genève	617.000
	Premier serveur web mis en service au SLAC : Stanford Linear Accelerator Center	
1993	MOSAIC, premier navigateur web sous l'environnement X	2.056.000
1994	WWW, première conférence internationale sur le web	3.864.000
	Yahoo!, premier annuaire web créé par David Filo et Jerry Yang	
1995	Création du consortium W3C dirigé par l'INRIA, le MIT et l'université Keio au Japon	6.642.000
	Apparition des premiers moteurs de recherche : Lycos, Altavista	
1998	Larry Page et Sergey Brin créent Google, un moteur de recherche révolutionnaire fondé sur le PageRank	36.739.000
Depuis	Le web s'est diversifié : sources de données en-ligne, réseaux pair à pair, commerce électronique, web sémantique...	285.139.107

sources : <http://isoc.org>
<http://public.web.cern.ch/>

TAB. I.1 – Dates importantes de l'évolution d'internet et du web

Fondé sur le paradigme de la navigation par liens hypertextes, le web a été développé en 1990

par Tim Berners Lee et Robert Cailliau, tous deux informaticiens au CERN¹ à Genève. Initialement conçu pour aider les physiciens du CERN à partager et à diffuser leurs documentations, le web a conquis depuis tous les publics engendrant une révolution sans précédent de la société d’informations de cette dernière décennie. En devenant un outil indispensable de consultation d’informations et un média incontournable de diffusion de tout type de données, le web a gagné une forte popularité auprès du grand public contribuant ainsi à la démocratisation de l’internet.

À ses débuts, le web ne proposait aucun outil permettant de rechercher des documents correspondant à des besoins donnés. Il fallait impérativement connaître la localisation (l’url) du ou des documents susceptibles de contenir l’information recherchée. Pour rechercher, par exemple, des notes de cours en c++, il fallait connaître les urls des pages d’accueil des personnes enseignant ce langage de programmation et proposant également leurs cours sous forme électronique. Faute de cette connaissance, la recherche était plus compliquée, voir impossible.

Cette situation a motivé le développement d’applications de simplification et d’amélioration de la recherche d’informations sur le web. C’est ainsi qu’en 1995, les premiers moteurs de recherche (Lycos, Yahoo! et Altavista) firent leur apparition sur le marché. Leur objectif consistait à offrir aux utilisateurs un accès transparent au contenu du web en utilisant de simples requêtes mots clés. À partir de cette période, le moteur de recherche est devenu l’intermédiaire indispensable pour rechercher et consulter des documents sur le web.

Le fonctionnement d’un moteur de recherche repose sur le principe d’indexation par mots clés de documents web. Chaque moteur de recherche dispose d’un entrepôt de données, appelé aussi index web, servant à stocker des documents. Cet entrepôt est régulièrement alimenté en documents rapatriés depuis le web par des programmes automatiques appelés *robots exploreurs*. Avant d’être stockés dans l’index, les documents rapatriés sont analysés, nettoyés (débarassés de leurs balises html par exemple) puis indexés par les mots clés présents dans leurs contenus.

Pour une recherche documentaire donnée, l’utilisateur doit soumettre une requête mots clés au moteur de recherche dans laquelle il spécifie les mots clés que doivent contenir les documents souhaités. Cette requête peut être vue comme étant une description sommaire des documents ciblés par la recherche. Partant de cette description, le moteur de recherche évalue la requête soumise sur son index en s’appuyant sur les techniques usuelles de recherche d’informations, comme par exemple le calcul de similarité entre un document et une requête. Un score est alors calculé pour chaque document présent dans l’index du moteur de recherche. Seuls les documents totalisant un score élevé sont présentés en réponse à l’utilisateur.

En 1998, Larry Page et Sergey Brin révolutionnèrent le secteur de la recherche de documents sur le web en créant le moteur de recherche le plus efficace et le plus utilisé du marché : **Google** [Google site]. Selon une étude menée en novembre 2004 par l’entreprise Médiamétrie-eStat², Google laisse très peu de place à la concurrence en étant à l’origine de 74% des visites de sites sur le web. Autrement dit, plus de sept visites sur 10 d’un site quelconque proviennent d’un clic à partir des réponses fournies par Google.

Google doit son efficacité à l’utilisation du **PageRank** [Brin et Page 1998]. Le PageRank est

¹Centre Européen de Recherche Nucléaire : <http://public.web.cern.ch/>

²Entreprise spécialisée dans la mesure du trafic internet : <http://www.estat.com/>

une mesure qui permet de quantifier récursivement la popularité d'un document du web en fonction de la popularité des documents de son entourage. Un document populaire est alors défini comme étant un document pointé par plusieurs autres documents populaires. Précisons que la popularité d'un document dépend uniquement de la popularité des documents de son voisinage et est donc totalement indépendante du contenu de ce document et de toute requête soumise à Google.

L'utilisation du PageRank pour pondérer le score de similarité d'un document donné avec n'importe quelle requête mots clés a pour objectif de privilégier (présenter en haut du classement des réponses) les documents populaires totalisant un taux élevé de similarité avec la requête soumise. L'intuition sous-jacente à cette technique consiste à affirmer qu'à similarité égale, un document populaire a plus de chances de correspondre à la requête soumise qu'un document moins populaire. Signalons, par ailleurs, qu'avec ses huit milliards de documents répertoriés, l'index de Google compte parmi les index les plus exhaustifs du web.

L'amélioration de la qualité des moteurs de recherche a entraîné une forte audience sur les sites web. En effet, selon Médiamétrie-eStat, 40% du trafic web provient des moteurs de recherche³. Cette audience n'a pas échappé aux acteurs économiques qui ont vite compris l'intérêt stratégique d'exister dans le web. Ceci a donné lieu à une multiplication de créations d'entreprises dites « jeune pousses » (*start-up*) proposant chacune des services autour du web (commerce électronique, bourse en-ligne, centrales de réservations de billets de voyages, B2B, P2P...). On parlait alors de la nouvelle économie et de la bulle internet.

Fondée essentiellement sur la spéculation financière, la bulle internet a fini par éclater en 2001 mettant un terme à l'euphorie boursière pour les nouvelles technologies du web. Fort heureusement, cet avatar spéculatif n'a eu aucune incidence négative sur l'intérêt que porte la recherche académique pour le développement du web.

2 Contexte de la thèse : Le web d'aujourd'hui

En quelques chiffres, le web d'aujourd'hui représente : plus de 42 millions de sites différents, une audience estimée à plus de 934 millions d'utilisateurs à travers le monde et un réseau en-ligne de transactions financières chiffrées à plus de 150 milliards de dollars par an⁴. Dans le cadre de nos travaux de recherche, nous avons retenu quatre caractéristiques pour résumer le contexte actuel du web : la taille du web, le rôle du web dans la société d'informations, la recherche d'informations sur le web et le format des documents du web.

2.1 Taille et croissance du web

Depuis sa création, le web a vu sa taille croître régulièrement de manière phénoménale. En 2000, cette croissance a été estimée à plus de 7,3 millions de documents ajoutés par jour⁵. D'autre part, Google [Google site], le moteur de recherche le plus exhaustif du marché, a recensé en début d'année 2005 plus de huit milliards de documents accessibles depuis le web, soit plus du double de ce que ce moteur recensait six mois auparavant. Par ce volume de données et par cette impressionnante croissance, le web couvre actuellement la quasi totalité des domaines de la société contemporaine

³Sur ces 40% de trafic, 74% provient de Google

⁴Source : Journal du net - <http://www.journaldunet.com/>

⁵Source : <http://www.sims.berkeley.edu/research/projects/how-much-info/internet.html>

(actualité, culture, politique, économie, recherche scientifique, loisirs, tourisme, sport...). Avec une telle couverture thématique, il est difficile de penser qu’une information sur un sujet donné ne soit pas abordée par au moins un document parmi les milliards de documents que compte le web ! Il est alors aisé de percevoir l’intérêt stratégique que représente la recherche d’informations sur le web pour certaines communautés (les scientifiques et les industriels par exemple).

2.2 Le rôle du web

Vu à ses débuts comme un simple réseau de consultation d’informations via un navigateur, le web est devenu un support incontournable dans le développement d’applications de traitement d’informations. De par son volume et la diversité de son contenu, le web est aujourd’hui considéré comme une source importante en données hétérogènes (articles, données scientifiques, données financières, données multimédia...).

Les moteurs de recherche figurent parmi les premières applications à avoir utilisé les documents du web pour alimenter leurs entrepôts de données. N’ayant pas de besoins ciblés sur un thème particulier, ces moteurs utilisent et rapatrient le plus possible de documents du web sans aucune restriction sur leurs contenus. On parle dans ce cas d’une approche généraliste (*general purpose search engine*).

Ayant une précision satisfaisante pour les requêtes simples, l’approche généraliste permet de répondre correctement aux recherches et besoins simples (recherche d’articles sportifs, recherche d’agence de voyage, recherche de fichiers musicaux...). Toutefois, la précision de cette approche est beaucoup plus discutable lorsqu’il s’agit de répondre à des besoins ou des recherches pointus : recherche de données scientifiques particulières, mise en place d’une veille technologique sur un domaine précis, construction d’un entrepôt de données thématiques (ex : le projet « eDot », cf. section 4 page 9)...

2.3 La recherche d’informations sur le web

La recherche d’informations sur le web peut s’effectuer de deux manières différentes : par navigation ou par utilisation de moteurs de recherche.

La recherche d’informations par navigation revient à retrouver les documents souhaités par saisie directe de leur url ou par parcours, manuel ou automatique, des liens hypertextes entre sites. Nous avons montré précédemment que ce mode de recherche de documents du web nécessite la connaissance d’un minimum d’urls pertinentes et intéressantes pour la recherche à effectuer. Or, la taille actuelle du web ne permet pas de constituer cette connaissance.

L’utilisation des moteurs de recherche représente une solution intéressante au problème soulevé par le mode de recherche précédent. Le principe consiste à décrire les documents cibles par une requête mots clés. Après évaluation sur les documents de son index, le moteur de recherche renvoie une liste d’urls (réponses) jugées « pertinentes » par rapport à la requête soumise.

Les moteurs de recherche actuels ne permettent que l’expression de requêtes simples par mots clés auxquelles sont données des réponses peu précises. Pour filtrer les mauvaises réponses et retrouver l’information souhaitée, il est nécessaire de procéder à une analyse humaine, souvent pénible et coûteuse en temps au vu du nombre de réponses retournées. L’utilisation de ce mode de recherche

pour satisfaire les besoins pointus mentionnés dans la section précédente (cf. section 2.2) n’est pas fiable. Par exemple, pour construire un entrepôt thématique de données de qualité, l’utilisation d’un moteur de recherche classique comme source de données nécessite une intervention humaine intensive et régulière pour filtrer et valider les documents à garder dans l’entrepôt. L’absence d’outils de qualité pour l’automatisation de la phase de filtrage/validation rend ce type d’entrepôts difficiles à maintenir à jour.

2.4 Format des données du web

Malgré l’émergence de nouveaux standards de représentation d’informations en-ligne (XML, RDF, OWL, ...), le web reste essentiellement dominé par le format HTML. Selon une étude expérimentale menée par Allan Heydon et Marc Najork [Heydon et Najork 1999], ce format représente plus de 69% du volume total du web.

Offrant principalement des fonctionnalités graphiques pour l’affichage de documents dans les navigateurs web, le langage HTML ne fournit aucun mécanisme de description « sémantique » des données contenues dans les documents du web, ce qui les rend inexploitable par les modèles classiques de traitement et d’intégration d’informations. Pour pallier cette insuffisance du langage HTML, des formats plus élaborés ont été proposés par le W3C : XML, RDF, OWL, *etc.* Ces formats constituent ce que l’on appelle le web sémantique.

Le Web sémantique représente un champ de recherche en pleine expansion dont l’objectif principal consiste à définir les futurs standards de représentation de l’information sur le web. À terme, la communauté scientifique cible la transformation du contenu du web en un contenu compréhensible par les machines et sur lequel il est possible de « raisonner ».

Notre travail de thèse ne concerne pas directement le domaine du web sémantique. En effet, nos travaux de recherche ont porté sur l’amélioration de la recherche d’informations sur l’état actuel du web, c’est-à-dire dans un contexte purement HTML. Il est néanmoins possible d’utiliser nos outils dans un contexte orienté web sémantique pour rechercher des documents thématiques au format HTML et enrichir leur contenu par des annotations sémantiques. Dans le cadre du projet « eDot » (cf. section 4 page 9), nous avons exploité cette possibilité pour fournir des documents HTML à un processus d’enrichissement sémantique de documents HTML traitant du risque alimentaire.

3 Motivation de la thèse : recherche ciblée de documents pour la construction automatique d’entrepôts de données thématiques

3.1 Problématique

Partant du contexte décrit plus haut, la recherche et l’exploitation d’informations sur le web se révèlent, encore aujourd’hui, difficiles à réaliser dans certains cas. En effet, pour le cas des recherches documentaires avancées (documents thématiques par exemple), le recours à des experts humains est nécessaire pour contrebalancer l’imprécision des réponses retournées par les outils de recherche actuels (moteurs de recherche en général). Nous expliquons cette imprécision par le principe même du fonctionnement de ces outils, à savoir l’évaluation de langages de requêtes mots clés sur des index de documents alimentés régulièrement depuis le web. Les problèmes liés à ce principe de fonctionnement peuvent être résumés par les trois points suivants :

1- Expressivité limitée des langages de requêtes proposés : les moteurs de recherche proposent généralement des langages de requêtes mots clés simples mais extrêmement limités en expressivité. Les documents souhaités sont vaguement décrits par une liste de mots clés sans possibilité d’exprimer d’autres caractéristiques qui s’avèrent parfois nécessaires pour cibler la recherche sur des documents complexes. Par exemple, pour les besoins du projet « eDot » (cf. section 4 page 9), il n’a pas été possible de formuler une requête mots clés classique pour rechercher des documents exclusivement au format HTML et contenant les mots clés « salade » et « listéria » dans un tableau.

2- Méthode naïve d’évaluation des langages de requêtes mots clés : Outre leur expressivité réduite, les requêtes mots clés souffrent d’une méthode d’évaluation naïve qui consiste à considérer que tout document contenant, dans n’importe quel ordre et à n’importe quel endroit de son contenu, les mots spécifiés dans la requête soumise est potentiellement pertinent. Un calcul mathématique du score de similarité document-requête permet de départager les documents précédents en favorisant ceux qui contiennent les mots de la requête en grand nombre. Or, l’expérience démontre que l’occurrence fréquente des mots clés de la requête dans n’importe quel ordre dans un document n’est pas un critère de pertinence fiable. Pour illustrer notre propos, considérons les deux exemples suivants :

Exemple 1 : La recherche des mots clés « ciel » et « bleu » dans les documents du web conformément à la méthode décrite plus haut (en ignorant l’ordre de l’occurrence des mots « ciel » et « bleu ») peut faire ressortir trois catégories de documents potentiellement pertinents : les documents contenant « ciel bleu » dans cet ordre, les documents contenant « bleu ciel » dans cet ordre et les documents contenant les mots « bleu » et « ciel » dans un ordre quelconque et éloignés l’un de l’autre. En supposant que les fréquences d’occurrences des mots clés « ciel » et « bleu » dans chaque catégorie sont importantes, la première catégorie regroupe a priori les documents traitant du ciel qui est bleu. La deuxième catégorie regroupe a priori les documents traitant de la couleur bleu ciel. Enfin, la dernière catégorie peut à priori regrouper différents thèmes. Sans aucune indication supplémentaire, la méthode d’évaluation de la requête « ciel bleu » est incapable de lever l’ambiguïté qui caractérise cette recherche.

Exemple 2 : Soit la requête mots clés « cours c++ » pour rechercher sur le web des documents de cours en c++. Sur les 10 meilleures réponses retournées par Google, cinq seulement correspondent réellement à des documents de cours c++. Par ailleurs, quelques documents indésirables (descriptif de formations payantes en c++ ou des emplois du temps en-ligne d’un cours c++) se sont bien classés parmi les réponses retournées par Google parcequ’ils contenaient les mots « cours » et « c++ » en forte proportion. Due à leur indépendance du contenu des documents et de la requête soumise, les mesures de popularité, PageRank [Brin et Page 1998] par exemple, sont impuissantes face aux effets de bord des méthodes d’évaluation des requêtes mots clés.

3- Importance du coût d’alimentation et de mise à jour des index sur lesquels les requêtes sont évaluées : Nous avons vu précédemment que les moteurs de recherche évaluent leurs requêtes mots clés sur les documents stockés dans leur index web. Cet index doit être alimenté par le plus possible de documents différents du web pour que le moteur de recherche puisse répondre à toute requête éventuelle de l’utilisateur (sur : la politique, le sport, l’actualité, le tourisme, la météo...). Pour cela, les robots explorateurs utilisés par les moteurs de recherche parcourent généralement le web de manière systématique sans aucune restriction particulière sur la nature des documents à rapatrier. Le principe consiste à suivre tous les liens hypertextes contenus dans un

document déjà visité pour atteindre et rapatrier de nouveaux documents. Tout nouveau document visité est analysé et ses liens hypertextes sont extraits et mis en attente de traitement. Au delà d'un certain nombre de documents visités, la répétition récursive de l'opération précédente sur chaque nouveau document visité provoque une croissance exponentielle du nombre de liens en attente de traitement. Avec plus de huit milliards de documents dans le web, une exploration systématique engendre très rapidement un nombre impressionnant de liens en attente de traitement qu'il n'est pas possible de gérer sur un système possédant une configuration matérielle modeste. L'approche systématique sur le web d'aujourd'hui est devenue un défi technologique d'envergure nécessitant d'importants moyens techniques que seuls les industriels peuvent financer.

Par ailleurs, le volume et la croissance du web actuel entraînent un allongement pénalisant du temps nécessaire à l'accomplissement d'un cycle complet d'exploration. Cette situation complique sérieusement la mise à jour des index des moteurs de recherche et se traduit par un décalage entre la version en-ligne des documents et leur version stockée dans l'index. Ce décalage peut, dans certains cas, être source d'imprécision dans l'évaluation des requêtes.

3.2 Idées défendues dans cette thèse

Au vu du constat dressé précédemment, l'amélioration de la recherche sur le web passe nécessairement par la spécialisation des outils de recherche. Cette thèse défend le principe qui stipule que le web peut être fouillé de manière plus efficace, que la fouille globale actuelle, en le découpant en différentes communautés thématiques. En effet, le volume impressionnant et l'hétérogénéité du contenu du web sont généralement sources de confusion et d'ambiguïté lors de l'évaluation des requêtes. Il est plus facile de gérer et de fouiller une collection restreinte de documents traitant de la même thématique. De ce fait, il s'agit de viser la construction automatique d'entrepôts thématiques et le développement d'outils de recherche spécialisés pour l'augmentation de la précision de la recherche sur le web. Se posent alors deux problèmes : comment caractériser les documents relatifs à une thématique donnée ? et comment les retrouver sans devoir parcourir l'intégralité du web ? La thèse défendue dans ce manuscrit répond à ces deux questions de la façon suivante.

D'abord, nous montrons qu'une façon d'augmenter la précision et l'expressivité de la caractérisation des documents souhaités consiste à combiner différents critères mots-clés ciblés sur différentes parties de la structure d'un document (exemple : son titre, son corps, ses liens sortants...) ou du contexte d'un document (les liens entrants, l'url, son type...). Nous montrons également qu'avec la définition de mécanismes déclaratifs pour la spécification et le contrôle du mode d'évaluation des mots clés sur les documents du web, il est possible de combiner efficacement des critères de recherche pour qu'ils correspondent aux mieux aux besoins de la recherche souhaitée. Il est alors possible de formuler des requêtes mots clés évoluées dont l'emploi peut s'avérer utile pour décrire les documents d'une thématique complexe.

Ensuite, nous plaçons en faveur des approches sélectives pour explorer efficacement le web à la recherche ciblée de documents thématiques. Cette approche est étayée par une étude expérimentale qui montre que l'exploration du web guidée par une fonction d'estimation de l'intérêt de rapatriement d'un document, augmente de manière significative la proportion de documents pertinents rapatriés.

3.3 Contributions

La clé de la recherche ciblée de documents sur le web réside dans la capacité à caractériser clairement les documents d'intérêt pour pouvoir, par la suite, les distinguer sans ambiguïté des autres documents. Dans ce sens, nous proposons *WeQueL*, un langage de requêtes déclaratif qui permet de décrire les documents souhaités par combinaison de critères de recherche mots clés. La particularité de notre langage réside dans la possibilité de cibler la recherche de mots clés sur des parties spécifiques de la structure du document. L'autre particularité du langage *WeQueL* réside dans la méthode d'évaluation de ses requêtes. En effet, par un système de poids et par un ensemble de fonctions prédéfinies, il est possible d'attribuer un mode d'évaluation différent pour chaque critère de recherche. Un score de pertinence est calculé pour chaque document à la suite de son évaluation sur une requête *WeQueL*. Ce score permet, par exemple, d'établir un classement des documents évalués pour présenter en priorité les documents totalisant les scores les plus élevés, *i.e.* les plus pertinents.

À notre connaissance, aucun autre langage de requêtes pour le web ne permet de décrire aussi finement les documents souhaités en combinant à la fois plusieurs critères différents et en offrant la possibilité de choisir la méthode d'évaluation de la requête construite.

En s'appuyant sur ce langage, nous proposons, par ailleurs, une approche sélective d'exploration du web guidée par une requête *WeQueL* (appelée requête d'exploration). L'objectif de notre approche est de rechercher le plus possible de documents correspondant à la requête d'exploration sans devoir parcourir tout les documents qui se présentent lors du parcours du web. Une exploration sélective guidée par une requête *WeQueL* repose sur une fonction qui permet d'anticiper le score d'évaluation d'un document avant de rapatrier son contenu. Cette anticipation est forte utile lorsqu'il s'agit de choisir le prochain candidat à rapatrier parmi les documents en attente de rapatriement. La stratégie consiste tout naturellement à privilégier le rapatriement du « meilleur », c'est-à-dire, le document estimé avec le score le plus élevé. Cette fonction d'estimation est mise en œuvre par un système d'apprentissage bayésien en-ligne s'appuyant sur un ensemble de mesures statistiques recueillies au fur et à mesure de l'exploration.

4 Le projet eDot : Contexte applicatif de la thèse

Le projet « eDot » [eDot site] (Entrepôt de Données Ouverts sur la Toile) consiste à développer des technologies permettant la construction, la plus automatique possible, d'entrepôts de données thématiques alimentés en documents depuis le web. La thématique choisie est : le risque bactériologique de contamination d'aliments.

Le projet « eDot » est le fruit de la collaboration de trois partenaires universitaires avec un partenaire industriel : l'équipe IASI du LRI (Université Paris-Sud et CNRS), le projet GEMO (INRIA-Futurs), l'équipe BIA (INRA-INAPG) et l'entreprise « jeune pousse » Xyleme.

4.1 Contexte du projet

Dans le cadre du projet Sym'Previus [SymPrevius site], l'équipe BIA a construit une base de données intitulée « Sym'Previus » et un système d'interrogation de cette base adapté au contexte du risque alimentaire. Cette base regroupe les informations caractéristiques du comportement de

certaines bactéries dans les aliments courants et sous certaines conditions d'expérimentation (température, pH...).

La construction de la base Sym'Previus a nécessité une intervention humaine pour la saisie manuelle de données provenant :

- de résultats expérimentaux fournis par des industriels de l'agroalimentaire ;
- de publications scientifiques (au format PDF) choisies pour leur représentativité du domaine du risque microbiologique dans les aliments.

Pour certaines expériences, les données recueillies dans la base Sym'Previus sont incomplètes ou parfois imprécises. L'intérêt du projet « eDot » est de proposer un moyen supplémentaire pour limiter les effets liés à l'incomplétude de la base Sym'Previus. Par des procédés d'acquisition automatique de données depuis le web, les outils développés dans le cadre du projet « eDot » contribuent à la complétion « automatique » des données de la base Sym'Previus, ce qui permet par exemple de s'affranchir du coût de la saisie manuelle des informations.

4.2 Intérêt du projet

Outre l'automatisation du processus d'acquisition pour la complétion de la base de données Sym'Previus, le projet « eDot » présente, par l'intermédiaire de ses différents outils, un intérêt capital pour la microbiologie prévisionnelle. La microbiologie prévisionnelle est un procédé d'aide à la maîtrise du risque de contamination alimentaire. Le principe consiste à prédire le comportement d'un micro-organisme (croissance, destruction, survie) dans les aliments courants sous l'effet de différents facteurs : le pH, les acides organiques, l'activité de l'eau (A_w) ou encore la température.

Pour réaliser ses prédictions, la microbiologie prévisionnelle utilise des modèles mathématiques reposant sur des données expérimentales. Ces données modélisent et quantifient les observations du comportement de certains micro-organismes dans différents aliments. Les objectifs visés par cette modélisation sont de :

- prévoir les chances de présence et de survie d'un micro-organisme à un moment donné du cycle de vie d'un produit alimentaire (fabrication, conditionnement, distribution) ;
- quantifier l'effet des facteurs d'une chaîne de fabrication sur le comportement des micro-organismes (modification de la température de conservation...).

4.3 Documents cibles de l'entrepôt « eDot »

Compte tenu des objectifs du projet « eDot », les documents cibles doivent fournir les données manquantes dans la base de données Sym'Previus afin de permettre la mise en place des procédés mathématiques de la microbiologie prévisionnelle. Les experts microbiologistes ont identifié la nature et le contenu des documents à cibler lors de la phase d'acquisition de données depuis le web. Les documents cibles sont des documents au format PDF ou HTML ayant un ou plusieurs tableaux dans leurs contenus. De plus, ces tableaux doivent contenir un maximum de mots clés spécifiques au domaine du risque microbiologique dans les aliments. À cet effet, les experts microbiologistes ont mis une ontologie du domaine du risque alimentaire à la disposition des différents modules du projet. Cette ontologie décrit la hiérarchie entre les différents concepts pertinents pour la prévention de la contamination alimentaire.

4.4 Architecture fonctionnelle du projet

Suite aux spécifications formelles des besoins des différents processus de traitement de l'entrepôt (acquisition, nettoyage & structuration, stockage et interrogation des données), le projet « eDot » a été découpé en plusieurs modules fonctionnels représentés dans la figure I.1 page 11. Il a également été convenu d'implanter ces modules sous forme de services web, distribués sur des plateformes distantes (INRIA, LRI, INA-PG).

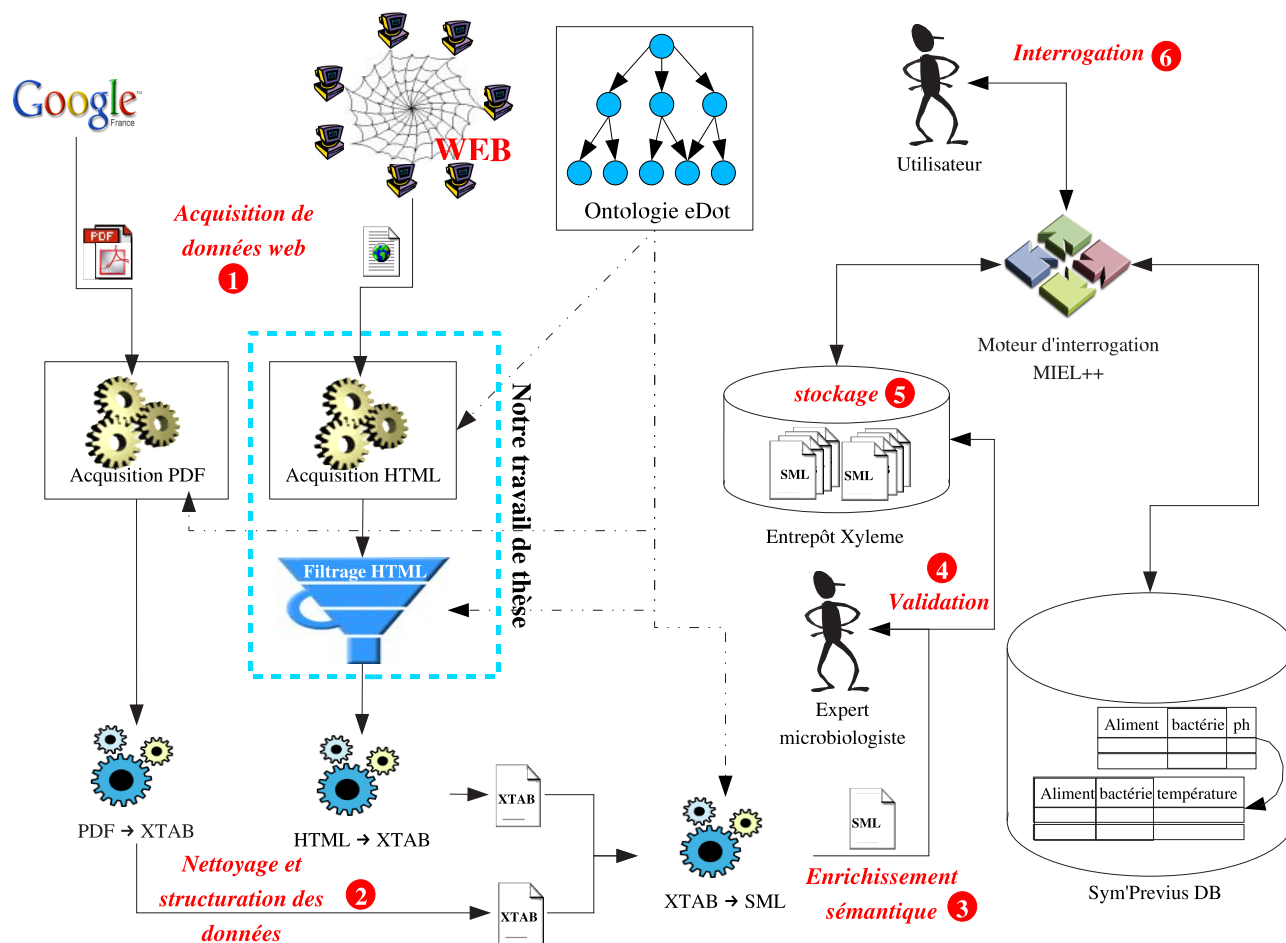


FIG. I.1 – Architecture fonctionnelle du projet « eDot »

En tête de la chaîne de traitements du projet « eDot », le processus d'acquisition de données du web, représenté dans la figure I.1 par l'étape ❶, regroupe deux modules différents : un module d'exploration du web à la recherche de documents cibles au format HTML et un module de récupération de documents cibles au format PDF.

Notre travail de thèse a consisté à développer les techniques nécessaires à l'acquisition automatique de documents cibles au format HTML. Le module correspondant regroupe deux applications :

1. Un robot explorateur dédié à l'exploration du web pour la recherche de documents cibles au format HTML. Le fonctionnement de ce robot est décrit en détail dans le chapitre VI page 135 de ce manuscrit.

2. Un module de filtrage de documents dont le but est de rapatrier et filtrer un ensemble d'urls de documents HTML. La description de ce module est détaillée dans la section 5.2 du chapitre III à la page 41 de ce manuscrit.

Le module de récupération de documents cibles au format PDF est développé par l'équipe BIA à l'INA-PG. Son rôle consiste à récupérer depuis le web le plus possible de documents cibles au format PDF. Pour cela, ce module utilise les réponses fournies par Google à un ensemble de requêtes mots clés pour rapatrier des documents au format PDF. Chaque requête soumise contient :

- `filetype :pdf`, un mot réservé de la syntaxe de Google pour restreindre le format des réponses au format PDF ;
- un nom de germe provenant de l'ontologie Sym'Previus ;
- et un nom d'aliment provenant également de l'ontologie Sym'Previus.

Au terme du rapatriement des documents cibles (formats HTML et PDF), le module de nettoyage et de structuration des données est déclenché pour transformer le contenu des documents cibles en un contenu exploitable par les modules qui suivent dans la chaîne de traitements. L'objectif de cette transformation est de localiser les tableaux contenus dans les documents cibles afin de décrire leur structure. Ce travail revient à identifier les lignes et les colonnes du tableau, les cellules dans chaque ligne, le titre du tableau... Cette transformation est schématisée par l'étape ❷ de la figure I.1.

À la fin de ce processus, chaque document cible est transformé en un document XTAB où les tableaux initiaux du document cible sont annotés par des balises XML. Précisons que, pour les documents PDF, un traitement préliminaire spécifique est utilisé pour convertir le contenu binaire du document PDF en un contenu textuel. La transformation en XTAB s'effectue sur la base de cette conversion.

Dans l'étape suivante (l'étape ❸ de la figure I.1), le module XTAB2SML [Saïs et al. 2005, Saïs 2004] permet d'enrichir les données formatées en XTAB. Cet enrichissement consiste à ajouter des balises sémantiques devant chaque cellule du tableau XTAB pour décrire la nature ou la fonction de la donnée contenue (produit, germe, caractéristique...). Pour cette tâche d'annotation, le module XTAB2SML utilise les termes de l'ontologie Sym'Previus. En sortie, un document SML (*Semantic Markup Language*) est créé pour chaque document XTAB fourni en entrée.

Avant d'être stockés dans l'entrepôt (étape ❺ de la figure I.1), les documents SML sont validés par un expert microbiologiste ((étape ❹ de la figure I.1). Le but de cette opération est d'éliminer les données bruitées ou peu vraisemblables provenant de sources non fiables.

5 Plan du manuscrit

Ce manuscrit est organisé en deux parties : les langages de requêtes pour le web et l'exploration du web.

Partie 1 : Langages de requêtes pour le web

Chapitre II : État de l'art

Ce chapitre présente un état de l'art sur les langages d'interrogation de documents du web. Trois catégories de langages de requête pour le web sont distinguées : les langages sur le contenu

textuel (type langages mots clés des moteurs de recherche), les langages de navigation web (reposant généralement sur le modèle relationnel/SQL) et les langages de restructuration de sites web (inspirés du paradigme objet). Ce chapitre se conclut par le positionnement du langage WeQueL, proposé dans cette thèse, par rapport aux catégories précédentes.

Chapitre III : WeQueL

Ce chapitre présente le langage WeQueL visant l'augmentation de l'expressivité des langages de requêtes mots clés pour l'amélioration de la précision de la recherche documentaire sur le web. La syntaxe et la sémantique des requêtes sont décrites. Puis, une étude expérimentale sur la précision et l'efficacité des requêtes du langage WeQueL est présentée pour deux tâches : la recherche de documents thématiques (trois thématiques ont été ciblées : cours informatiques en-ligne, document de tourisme et photos de formule 1) et le filtrage de documents HTML dans le cadre du projet « eDot ».

Partie 2 : Exploration du web

Chapitre IV : Robots explorateurs du web

Ce chapitre est dédié aux détails techniques de l'exploration du web en général. Ce chapitre introduit également les notions et les notations abordées dans les deux chapitres suivants.

Chapitre V : État de l'art

Ce chapitre passe en revue les différentes techniques d'exploration du web abordées dans la littérature scientifique. Ces techniques sont classées en deux catégories : les techniques systématiques d'exploration du web (sans objectif thématique particulier) et les techniques d'exploration sélectives du web guidées par des critères thématiques.

Chapitre VI : Exploration sélective du web guidée par une requête WeQueL

Ce chapitre présente l'approche sélective d'exploration du web étudiée dans cette thèse. Cette approche est fondée sur le principe d'une exploration ciblée guidée par une requête du langage WeQueL. Ce chapitre est conclu par les résultats issus de deux explorations expérimentales ciblant deux thématiques : une exploration à la recherche des documents cibles de l'entrepôt d'eDot et une exploration à la recherche de documents de cours informatiques.

Première partie

Langages de requêtes pour le web

État de l'art

Dans ce chapitre, nous dressons un état de l'art des différents langages et techniques d'interrogation d'une collection de documents web. Nous avons classé les différents langages abordés dans la littérature scientifique suivant leurs fonctionnalités et leur pouvoir expressif associé. Ainsi, nous distinguons trois catégories : les langages de requêtes web sur le contenu du document, les langages de requêtes de navigation web et les langages de gestion et de restructuration de sites web.

Les langages de requêtes web sur le contenu du document reprennent, pour leur majorité, les techniques usuelles de recherches documentaires employées dans les systèmes de recherche d'informations (*Information Retrieval Systems*). Une requête d'un langage de cette catégorie se compose des critères et des conditions à satisfaire pour qu'un document soit considéré comme une réponse à la requête posée. Les critères et les conditions en question sont généralement formulés sous forme de mots clés pour cibler différents aspects d'un document tel que son contenu, son type, son environnement, *etc.* Cette syntaxe mots clés rend l'usage de ce type de langages simple et à la portée d'utilisateurs novices.

En plus de la recherche classique de documents par contenu, les langages de requêtes de navigation web permettent d'exprimer des besoins de recherche plus évolués qui prennent en compte les aspects navigationnels du web. En effet, et contrairement aux requêtes des langages de la catégorie précédente, les requêtes des langages de navigation web permettent d'exprimer des contraintes sur la structure et la topologie du chemin web menant vers les documents souhaités. Pour ce faire, ces requêtes s'appuient sur des modèles plus riches et sur des formalismes avancés qui restreignent l'utilisation des langages de cette catégorie à un public essentiellement initié (administrateurs de bases de données, concepteurs d'applications web, *etc.*).

Les langages de gestion et de restructuration de sites web sont, pour leur part, essentiellement utilisés dans un contexte de médiation et d'intégration de sources de données web tels que le web lui-même, les bases de données en ligne, les index de documents, *etc.* L'objectif visé consiste à fournir aux utilisateurs de ce type de langages les outils nécessaires à la définition, la structuration et la présentation de nouvelles vues sur les documents satisfaisant les critères de la requête soumise. Ces outils sont spécifiques aux langages de gestion et de restructuration de sites web car ils ne possèdent pas d'équivalent dans les deux catégories introduites plus haut. Par ailleurs, les langages de gestion

et de restructuration de sites web s'adressent à des utilisateurs confirmés car, comme pour les langages de navigation web, cette catégorie emploie des modèles de données complexes.

Dans ce qui suit, nous présentons en détail les particularités des trois catégories de langages web citées plus haut.

1 Langages de requêtes sur le contenu textuel du document

La philosophie de cette catégorie de langages consiste à rechercher dans une collection de documents web, les documents dont le contenu textuel satisfait une ou plusieurs conditions de la requête soumise. Ces conditions sont, dans la majorité des cas, exprimées sous forme de mots clés à rechercher dans le contenu textuel des documents évalués. Pour cette raison, les langages de requêtes de cette catégorie utilisent le vecteur de mots du contenu d'un document comme modèle de représentation de documents web. Cette représentation vectorielle est très simple à mettre en œuvre. Elle présente, par ailleurs, l'avantage d'être entièrement exploitable par les techniques usuelles de recherche d'informations tels que le calcul de similarité entre vecteurs de mots, le classement des réponses (*Ranking*), les techniques de classification automatique, *etc.*

Dans ce qui suit, nous avons divisé les langages de la littérature scientifique de cette catégorie en trois sous groupes : les langages mots clés classiques, les langages mots clés avancés et les langages mots clés utilisant des expertises de domaine.

1.1 Langages de requêtes mots clés classiques

Les langages mots clés classiques sont les outils les plus populaires et les plus utilisés pour rechercher des documents sur le web. Les premiers moteurs de recherche apparus, tels que Yahoo! [Yahoo! site] et Lycos [Lycos site], proposent à leurs utilisateurs d'interroger leur index web par des requêtes mots clés simples. Ceci s'effectue généralement via une interface web assez intuitive sous forme d'un formulaire multi-rubriques. Aucun prérequis n'est donc nécessaire pour pouvoir utiliser des requêtes mots clés.

Une requête mots clés classique se compose d'un ensemble de mots clés dont l'évaluation retourne en sortie une liste d'urls correspondant aux documents réponses à la requête soumise. Chaque mot clé de la requête est choisi par l'utilisateur pour sa « *supposée pertinence* » par rapport à la recherche documentaire visée. Le bien fondé des mots clés choisis dépend alors de la qualité des connaissances de l'utilisateur sur le domaine des documents qu'il souhaite rechercher. Par exemple, la requête mots clés « *Michael Schumacher* » est plus appropriée que la requête « *formule 1* » pour rechercher les documents traitant de *Michael Schumacher*. En supposant que les requêtes mots clés soient bien formulées (contenant les bons mots clés) pour une recherche donnée, la qualité des résultats ne dépend alors que de la manière d'évaluer cette requête.

La méthode d'évaluation d'une requête mots clés classique revient, dans la plupart des cas, à calculer un score de similarité entre chaque document à évaluer et la requête soumise. Ce procédé, très répandu en recherche d'information, est rendu possible grâce à la modélisation vectorielle des documents web et des requêtes mots clés. En effet, de la même manière qu'un document web, une requête mots clés est aussi représentée par un vecteur de mots. La similarité entre une requête mots clé et un document web revient alors à calculer la proximité géométrique entre leurs vecteurs de

mots respectifs. Le cosinus de l'angle entre le vecteur du document et le vecteur de la requête est l'une des mesures de proximité géométrique les plus utilisées dans l'évaluation des requêtes mots clés.

Une forte proximité géométrique (par exemple cosinus proche de 1) entre le vecteur caractéristique d'un document d et le vecteur caractéristique d'une requête mots clés Q traduit une présence massive des termes de la requête Q dans le contenu du document d . Cette présence massive est naïvement interprétée comme une similitude sémantique entre Q et d , c'est-à-dire que d et Q traitent du même sujet.

Par ailleurs, le système d'évaluation d'une requête mots clés par score de similarité permet de sélectionner les réponses à retourner à une requête en fixant un seuil minimal de similarité. Seuls les documents ayant un score de similarité supérieur à ce seuil sont retenus. Il est également possible, à partir de là, d'ordonner les réponses sélectionnées pour les présenter à l'utilisateur. L'ordre communément adopté par les systèmes à base de langages mots clés classique est l'ordre décroissant de similarité. Ainsi, les réponses totalisant les scores élevés, donc théoriquement pertinentes, sont présentées en priorité à l'utilisateur.

La méthode d'évaluation des requêtes mots clés par score de similarité décrite précédemment ne s'applique que sur des vecteurs de mots construits à partir du contenu textuel du document. Cette méthode est par conséquent totalement inexploitable pour rechercher des documents à contenu binaire (images, vidéos, son, postscript, pdf, *etc.*). Cette insuffisance est due à la caractérisation exclusive du document par son contenu. Il est donc logique de ne pas pouvoir calculer un score de similarité d'un document qui ne possède pas de vecteur mots clés caractéristique.

Pour les documents à contenu textuel (HTML par exemple), la précision des réponses obtenues par score de similarité est souvent faible. Ceci se traduit par un grand nombre de réponses parmi lesquelles une infime partie correspond approximativement aux souhaits de l'utilisateur. Cette imprecision s'explique par l'importance accordée par le score de similarité aux occurrences des mots clés de la requête dans le contenu du document évalué. En effet, l'abondance d'un mot w dans un document d ne rend pas forcément d pertinent par rapport à la requête contenant le mot w . Par exemple, les mots « web » et « document » sont les mots les plus fréquemment répétés dans ce chapitre (respectivement 138 et 219 fois) et pourtant ce chapitre ne traite pas des documents web mais traite des langages de requêtes web dans la littérature scientifique (occurrences de « langage » : 125 et « requête » : 188 dans ce chapitre).

D'autres phénomènes accentuent davantage l'imprécision des langages requêtes mots clés en biaisant le calcul de la similarité entre un document et une requête. La polysémie¹ représente, à elle seule, un sérieux problème dans l'évaluation d'une requête par calcul de similarité. En effet, la représentation vectorielle d'un document s'effectue sur la base de l'orthographe des mots du contenu. Les mots polysémiques sont tout simplement confondus. Par conséquent, l'interprétation des scores de similarité est complètement dénaturée pour des documents utilisant une terminologie polysémique.

Un autre phénomène qui altère le modèle vectoriel des langages mots clés est l'emploi excessif de certains mots clés dans des documents considérés stratégiques (sites touristiques, documents commerciaux et publicitaires, sites pornographiques, *etc.*). Cette exagération est volontaire et a pour but d'améliorer le positionnement de ces documents dans le classement des réponses d'un moteur

¹regroupe les mots ayant la même orthographe mais des sens différents. Ex : avocat (le métier) et avocat (le fruit)

de recherche classique.

Par ailleurs, le pouvoir expressif des langages de requêtes mots clés est limité au seul contenu du document car la syntaxe et la construction des requêtes mots clés ne permet pas toujours d'exprimer certains types de besoins. Les besoins navigationnels, par exemple, ne peuvent pas être exprimés par de simples mots clés. De plus, ces besoins nécessitent des modèles et des structures de données qui prennent en considération les chemins hypertextes entre les documents web et la topologie du graphe web. Ces deux derniers aspects sont tout simplement ignorés par les systèmes à base de langages mots clés classiques qui, pour des raisons de simplicité et d'efficacité, considèrent souvent le web comme une collection plate de documents hétérogènes. Ainsi, il est impossible de formuler une requête mots clés pour rechercher, par exemple, tous les articles sportifs sur la formule 1 accessibles depuis la page d'accueil de *Yahoo! Sport* et pointés par le mot *Ferrari*.

1.2 Langages de requêtes mots clés avancés

Les langages de cette catégorie partagent la même philosophie que les langages mots clés classiques vus plus haut. Cependant, deux différences majeures les distinguent.

La première différence réside dans la manière dont les langages de cette catégorie modélisent le web et ses documents. En effet, les langages de requêtes mots clés avancés emploient, contrairement aux langages mots clés classiques, une modélisation vectorielle plus riche que le simple contenu du document. Les liens entrants, *i.e.* les liens hypertextes qui pointent vers un document, sont ainsi exploités pour offrir une description additionnelle au contenu des documents. C'est de cette manière que Google [Google site], par exemple, associe une description textuelle à des documents binaires pour les rendre interrogeables par requêtes mots clés.

L'utilisation des liens entrants dans les langages mots clés avancés a été motivée par le constat unanime de la communauté web sur l'apport significatif des liens entrants dans la description des documents web. En effet, il est communément admis que les mots apparaissant dans l'ancre ou le voisinage d'un lien entrant l d'un document d , contiennent souvent une description concise mais de bien meilleure qualité que la description apportée par le contenu même du document d [McBryan 1994, Brin et Page 1998, Halkidi et al. 2002]. C'est ainsi que Google réussit à faire apparaître dans ses réponses des documents pertinents dont le contenu ne possède pas nécessairement les mots clés de la requête soumise. Par exemple, les pages de Google et Yahoo! apparaissent parmi les réponses à la requête « moteur de recherche » alors que les mots « moteur » et « recherche » n'apparaissent nul part dans les contenus respectifs de ces pages.

En plus des liens entrants, AltaVista [AltaVista site] utilise de manière intensive le contenu des balises <META> dans les documents HTML pour associer une description générique aux documents HTML. La position du mot dans le document (en début ou en fin du document) est également prise en compte dans la représentation d'un document.

Google [Google site] modélise les mots décrivant un document de manière plus fine que la simple occurrence. La modélisation de Google s'intéresse aux aspects esthétiques et contextuels de la présence d'un mot dans un document. Ainsi, un mot est représenté par son occurrence mais aussi par son style d'écriture (en gras ou en italique), sa taille (grande, normale ou petite), sa position dans le document, *etc.* Ces informations sont stockées dans une structure vectorielle appelée *Hit List* [Brin

et Page 1998].

La seconde différence entre les langages mots clés avancés et les langages mots clés classiques, réside dans la manière d'évaluer les requêtes mots clés ou dans la mesure de similarité utilisée pour cette évaluation. Malheureusement, Google, AltaVista et la plupart des moteurs de recherche communiquent très peu sur leurs méthodes d'évaluation de requêtes mots clés. Cette opacité s'explique par l'importance des enjeux économiques et financiers qui entourent la recherche documentaire sur le web. Ces enjeux ont fait que, actuellement, la méthode d'évaluation des requêtes d'un moteur de recherche est une marque de fabrication stratégique et un secret industriel bien protégé.

AltaVista [AltaVista site] utilise, par exemple, des techniques linguistiques pour contrebalancer les effets indésirables de la polysémie et couvrir plus de réponses. Avant d'évaluer une requête, AltaVista enrichit celle-ci par les lemmes de ses mots clés initiaux. Le calcul de la similarité sur l'index de documents d'AltaVista s'effectue sur la base de cette requête enrichie. Les résultats obtenus sont, par la suite, ajustés par des critères de localisation des mots clés dans les documents retenus comme réponses. Les réponses possédant une grande proportion de mots clés de la requête dans le début du contenu ou dans une balise <META> sont présentées en tête du classement des réponses. Ce classement privilégie également les documents contenant la plupart des mots clés de la requête dans une localisation proche, *i.e.* les mots de la requête ne sont pas dispersés dans le contenu du document.

Google [Google site] utilise sa propre mesure de pertinence pour juger de la proximité d'un document par rapport à une requête soumise. Cette mesure, appelée *Information Retrieval score (IR score)* [Brin et Page 1998], est calculée en deux étapes. Dans la première étape, Google identifie les documents ayant dans leur description textuelle (contenu+liens entrants) au moins l'un des mots de la requête soumise. Chaque document identifié est, par la suite, représenté par les vecteurs *Hit List* correspondant aux mots de la requête qu'il contient. Ces vecteurs sont utilisés dans la seconde étape pour calculer le *IR score* de chaque document par rapport à la requête soumise. Ce calcul exploite les informations esthétiques et contextuelles contenues dans les *Hit List* pour pondérer l'importance de l'occurrence des mots. Google [Google site] combine, par ailleurs, le score précédent avec une mesure de popularité pour lutter contre la fraude dans le classement des réponses à une requête donnée. Cette mesure, appelée «*PageRank*», est en grande partie à l'origine du succès et de l'efficacité du moteur de recherche Google.

Le *PageRank* [Brin et Page 1998] est une mesure de popularité indépendante du contenu du document et de la requête posée. Cette mesure utilise à la place la topologie du web et la topologie des liens entrants. Pour un document donné d , la mesure du *PageRank* estime la probabilité P de visiter le document d par une exploration aléatoire. Intuitivement, plus le document d est pointé par d'autres documents populaires, plus d a de chances d'être visité et d'être lu. Par conséquent, une probabilité P élevée signifie que le document d possède potentiellement une forte audience qui fait de lui un document de référence dans son domaine. Il est donc inutile d'exagérer l'occurrence de certains mots pour gonfler artificiellement le *IR score* de certains documents.

D'autres mesures du même genre que le *PageRank* peuvent facilement être combinées avec le calcul de similarité utilisé par les langages mots clés. Le *Hub and Authority* [Kleinberg 1999], par exemple, est une mesure de popularité qui permet d'améliorer le classement des réponses à une requête donnée. Le principe consiste à identifier parmi les réponses retournées celles qui sont

beaucoup référencées (*Authority*) par des documents populaires et celles qui référencent beaucoup de documents populaires (*Hub*) (voir figure II.1 page 22).

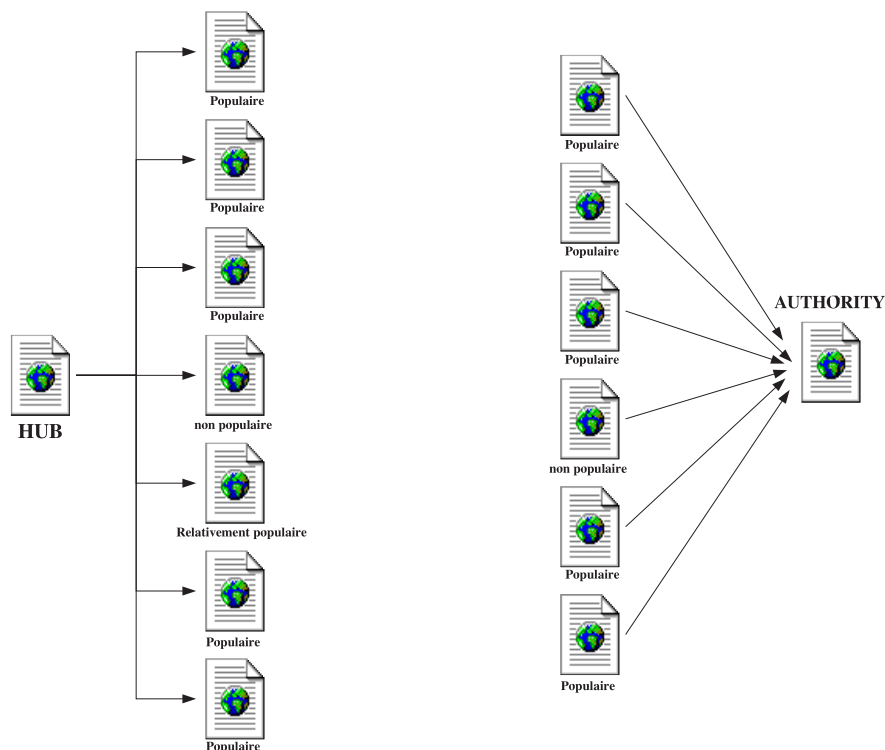


FIG. II.1 – Schématisation de la notion de Hub et Authority

Certains langages mots clés avancés proposent un niveau de granularité de recherche plus poussé que celui proposé par les langages mots clés classiques et les langages cités plus haut. En effet, les langages vus jusqu'à présent ne permettent pas de cibler la recherche des mots clés sur des parties précises d'un document. Il n'est donc pas possible de contrôler la recherche des mots clés de la requête soumise pour la focaliser sur une sous section particulière du document.

Google propose une option appelée *advanced Google* [Google_advanced site] qui permet à un utilisateur de spécifier la partie du document qui doit être ciblée par les mots clés de la requête ainsi que de spécifier d'autres critères tels que le type du document, la date de sa création, *etc.* Cette spécification s'effectue de manière déclarative via l'emploi de mots réservés qui déterminent les critères de granularité de la recherche. Ces mots réservés sont :

- «*allinurl*» pour cibler la recherche sur les mots de l'url ;
- «*allinanchor*» pour cibler la recherche sur l'ancre des liens entrants et/ou sortants ;
- «*allintitle*» pour cibler la recherche sur le titre d'un document ;
- «*allintext*» pour cibler la recherche sur le corps du document (contenu de la balise <BODY>) ;
- «*filetype*» permet de spécifier le type MIME [Freed et Borenstein 1993, Freed et Borenstein 1996a, Freed et Borenstein 1996b] des documents souhaités.

Dans l'option avancée de Google, la condition d'occurrence des mots de la requête dans le document n'est plus suffisante pour déclarer qu'un document est une réponse potentielle à la requête soumise. Il faut pour cela que ce document contiennent les mots de la requête dans la partie spécifiée

par l'utilisateur. Cette contrainte de localisation permet d'affiner la recherche mots clés pour mieux exprimer certains besoins non formulables avec des requêtes mots clés standards. Il est, par exemple, possible avec l'option avancée de Google de rechercher des documents de cours java au format *pdf* pointées par les mots *cours* et *java*. La requête à formuler est : «*allinanchor : cours java filetype :pdf*». Les résultats obtenus sont plus précis que les résultats obtenus par une requête générale telle que «*cours java pdf* ».

L'option avancée de Google offre à ses utilisateurs un moyen plus expressif que Google classique pour poser des requêtes mots clés. Les requêtes ainsi exprimées se montrent pour certains besoins plus efficaces et plus précises que des requêtes non ciblées. Cependant, la limitation majeure de cette option réside dans son manque de souplesse et de flexibilité. En effet, Il est d'une part impossible de spécifier des critères de granularité autres que ceux offerts par l'option avancée de Google (titre, url, text et ancre des liens). D'autre part, les possibilités de combiner deux ou plusieurs critères de granularité différents sont extrêmement réduites.

1.3 Langages de requêtes mots clés utilisant des expertises de domaine

Les langages de cette catégorie s'appuient sur des expertises de domaine pour améliorer la recherche par mots clés. Le principe consiste à utiliser une ontologie de domaine et/ou un thésaurus de référence pour intégrer de nouvelles informations sémantiques dans l'évaluation d'une requête mots clés. Ces informations, inférées de l'expertise utilisée, sont utilisées pour des fins d'enrichissement de la requête initiale. Un enrichissement consiste, par exemple, à incorporer dans la requête posée Q les synonymes des mots clés de Q ou les termes qui leurs sont équivalents dans l'expertise utilisée. Ceci permet de se libérer de la contrainte de conformité orthographique entre les mots d'un document et ceux de la requête posée. Ainsi, les réponses logiquement rejetées par les langages précédents sont couvertes par les langages à base d'ontologies. Par conséquent, leur rappel est plus important que les langages précédents.

THESUS [Halkidi et al. 2002] illustre parfaitement la catégorie des langages à base d'expertises de domaine. THESUS est l'acronyme de **THE**matic **SUB**set **S** of the web. Son objectif consiste à regrouper les documents web en sous-collections thématiques. Le système THESUS a été développé par Benjamin Nguyen dans le cadre de ses travaux de recherche pour sa thèse de doctorat [Nguyen 2003] et en collaboration avec l'équipe BD-NET du département informatique de l'université d'économie et d'affaires commerciales d'Athènes en Grèce. L'implantation du système THESUS, en tant que web service, est disponible à l'adresse suivante : <http://www.db-net.aueb.gr/thesus/wdCluster.jsp>.

THESUS est un système de classification et d'interrogation de documents web guidé par une ontologie du domaine. Il est construit au dessus d'un entrepôt thématique alimenté en documents à partir du web. La particularité de THESUS réside dans sa manière de représenter les documents web. En effet, contrairement à tous les langages vus précédemment, THESUS n'utilise que les liens entrants pour décrire un document web. Le contenu du document est totalement ignoré. Les auteurs ont justifié ce choix par deux raisons. D'une part, les auteurs considèrent que les liens entrants apportent souvent une description de qualité des documents vers lesquels ces liens pointent. D'autre part, ils estiment que le contenu des documents introduit un bruit suffisamment important pour nuire aux tâches de classification et de recherche de documents.

À l'opposé des langages des deux catégories précédentes, les requêtes de THESUS opèrent sur

un entrepôt structuré de documents thématiques et non sur n'importe quel document web. Pour cela, THESUS définit une requête d'initialisation de l'entrepôt à partir de l'ontologie utilisée. La soumission de cette requête à Google permet de présélectionner du web les documents conformes à l'ontologie en cours d'utilisation dans le système. THESUS recueille les réponses obtenues de Google et les rapatrie dans son entrepôt. Une phase de classification guidée par la hiérarchie de l'ontologie est ensuite enclenchée dans le but d'organiser et de structurer les documents rapatriés. THESUS utilise pour cela une adaptation de l'algorithme *DBSCAN* [Ester et al. 1996] et la métrique de *Wu & Palmer* [Wu et Palmer 1994]. Les documents sont alors regroupés dans des groupes (*clusters*) étiquetés par les mots décrivant les nœuds qui leur correspondent dans l'ontologie. Chaque document de l'entrepôt est identifié par sa description de mots provenant des liens entrants ainsi que par l'étiquette du groupe (*cluster*) auquel il appartient.

La phase d'initialisation du système THESUS utilise Google de manière intensive pour recueillir par exemple les liens entrants. La limitation d'utilisation de Google à 1000 requêtes par jour rend cette phase difficilement portable à l'échelle.

Une fois l'initialisation de l'entrepôt de THESUS finie, le système d'interrogation par mots clés devient opérationnel. Pour une requête mots clés Q donnée, le moteur de THESUS procède en premier lieu à la réduction de l'espace de recherche et cela afin d'éviter d'évaluer la requête Q sur la totalité de l'entrepôt. Pour ce faire, THESUS met en correspondance les mots clés de Q avec les termes de l'ontologie grâce au thésaurus *WordNet* [Felbaum 1998, Voorhees 1998, Miller et al. 1990]. Les termes de l'ontologie trouvés par cette mise en correspondance sont par la suite appariés sur les étiquettes des groupes de documents (*Clusters*) de l'entrepôt. Cet appariement permet à THESUS d'identifier les groupes de documents potentiellement intéressants pour l'évaluation de la requête Q . THESUS considère que ces documents sont «*sémantiquement*» proches des mots de la requête posée et donc mieux appropriés pour satisfaire Q . Un calcul classique de similarité est ensuite effectué sur les documents précédents afin de les ordonner. Seuls les documents totalisant un score supérieur à un seuil fixé sont retenus.

Dans leurs expérimentations sur le domaine de la technologie, les auteurs ont montré que le moteur de requête de THESUS est en moyenne deux fois plus précis que Google pour les mêmes requêtes.

De manière globale, THESUS se montre efficace et suffisant pour effectuer des recherches thématiques simples. Cette efficacité n'est malheureusement pas garantie pour des recherches thématiques complexes sollicitant le contenu des documents. En effet, substituer le contenu d'un document par ses liens entrants rend THESUS incapable de retrouver des documents ayant une structure donnée ou contenant des mots clés dans une partie précise du contenu. Par exemple, l'approche THESUS est inexploitable dans le cadre du projet eDot [eDot site] pour rechercher des documents HTML contenant des mots clés précis sur le risque alimentaire dans des tableaux.

2 Langages de requêtes de navigation web

Les langages de requêtes de navigation web permettent d'interroger les documents web à partir de leur contenu. Ces langages permettent également à l'utilisateur de formuler des requêtes de navigation qui ne peuvent être exprimées dans les langages précédents. Nous appelons par «*requêtes de navigation*» ou «*requêtes de connexion*» [Florescu et al. 1998], les requêtes contenant des

contraintes sur les chemins hypertextes du graphe du web menant aux documents visés par la requête. Par exemple, une requête de navigation permet d'exprimer et de rechercher des images de formule 1 accessibles depuis la page d'accueil de *Yahoo! Sport* par des chemins n'excédant pas trois liens et ayant au moins un lien avec le mot «*F1*» dans son ancre.

La prise en charge des requêtes de navigation par les langages de cette catégorie est rendue possible grâce à l'utilisation des expressions régulières de chemins (*regular path expressions*). Ce formalisme, emprunté au domaine de la théorie des graphes, offre à ses utilisateurs de puissants mécanismes déclaratifs de formulation de contraintes sur les chemins hypertextes du web.

Une expression régulière de chemins se construit à partir d'un alphabet de représentation des objets du web (document, lien hypertexte, url, ancre, *etc.*). Les éléments de cet alphabet constituent les expressions atomiques de ce formalisme. La combinaison de ces atomes, grâce aux opérateurs classiques d'expressions régulières ou à des opérateurs prédéfinis, permet de former des expressions de chemins plus complexes. Les opérateurs classiques sont : l'opérateur de concaténation, l'opérateur d'alternance ($|$), le quantifieur de répétition ($*$).

Le modèle de données employé dans les langages de requêtes de navigation web est le modèle relationnel. Le web et ses composants (documents+liens) sont modélisés par des tables relationnelles conformes à un schéma défini par avance et propre à chaque langage de cette catégorie. Ce schéma détermine la granularité des requêtes d'un langage de navigation web lors d'une recherche documentaire. En résumé, le web est vu par les langages de cette catégorie comme étant une base de données standard interrogeable par requêtes.

L'exploration du web permet par la suite de constituer un corpus de documents et de liens hypertextes pour matérialiser les tables de la base de données du web. La recherche documentaire consiste donc à interroger cette base via des requêtes SQL classiques ou via des requêtes propres au langage web utilisé.

L'utilisation du langage SQL et du modèle relationnel rend les langages de navigation web compatibles avec la plupart des moteurs de requêtes SQL du marché. Ceci permet, par exemple, d'intégrer le web à moindre coût dans des applications de bases de données déjà en place. Le web est alors vu comme une source de données supplémentaire exploitable à travers les requêtes des langages de cette catégorie.

Le modèle relationnel combiné aux requêtes de navigation constitue un avantage indéniable des langages de cette catégorie sur les langages mots clés. Cet avantage est rendu possible grâce aux mécanismes de parcours et d'exploration du web mis en place par les langages de navigation web lors de l'évaluation d'une requête de navigation. Ces mécanismes permettent de représenter une partie de la structure de graphe du web en reconstituant progressivement les chemins de liens hypertextes entre documents parcourus. La notion de navigation entre documents est ainsi explicitement capturée et représentée.

En contre partie, les langages de navigation web perdent les avantages offerts par le modèle vectoriel de mots clés. En effet, il n'est plus possible d'appliquer sur les tables relationnelles les techniques usuelles de recherche d'information pour évaluer une requête sur un document. Cette contrainte prive, par exemple, les langages de navigation du calcul d'un score d'évaluation d'une

requête sur un document. Par conséquent, peu de moyens objectifs s'offrent aux moteurs de requêtes de ces langages pour ordonner leurs réponses à une requête donnée.

Précisons, par ailleurs, que l'utilisation des langages de navigation web nécessite une certaine aisance dans la conception et la manipulation du modèle relationnel, une connaissance suffisante sur les bases de données et une bonne maîtrise des langages d'interrogation d'une base de données (SQL par exemple). Ces prérequis rendent l'emploi des requêtes de cette catégorie réservé à un public confirmé : administrateurs de bases de données, concepteurs d'applications web, *etc.*

2.1 WebSQL

WebSQL [Mendelzon et al. 1996, Mihaila 1996] est un langage web déclaratif d'interrogation du web et de sa structure de graphe. Ses requêtes peuvent à la fois porter sur le contenu des documents ou sur des contraintes de navigation dans la structure hypertexte du web. WebSQL reprend les techniques usuelles de bases de données pour les appliquer au web [Florescu et al. 1998]. La syntaxe des requêtes WebSQL est très proche de la syntaxe des requêtes du langage SQL : `[SELECT...FROM...WHERE...]`.

En 1997, WebSQL [Arocena et al. 1997, Mendelzon et al. 1997] a été enrichi avec de nouvelles fonctions d'analyse de la topologie du web. La nouvelle version permet de rechercher les liens hypertextes qui n'existent plus, appelés aussi *liens cassés*. Cette version permet également de récolter les liens entrants d'un document donné.

WebSQL propose deux tables relationnelles pour représenter les documents du web et leurs liens hypertextes : **Document** et **Anchor**. Le schéma de ces deux tables est donné ci-dessous :

```
Document(url:string,title:string,text:string,type:string,length:integer,modif:date)
Anchor (base:string,href:string,label:string)
```

où :

{	Document	est la relation décrivant un document par son url, son titre, son contenu, par son type MIME, par sa taille en octets et par la date de sa dernière modification.
	Anchor	est la relation décrivant un lien hypertexte l par l'url du document contenant l , l'url du document vers lequel l pointe et les mots apparaissant dans l'ancre de l .

WebSQL n'effectue aucune exploration web au préalable. Les tables **Document** et **Anchor** ne sont pas matérialisées. Elles restent donc vides et virtuelles. Leur emploi sert uniquement à des fins de compatibilité avec le moteur SQL utilisé pour interpréter et évaluer les requêtes posées.

Le moteur de requêtes WebSQL distingue trois types de requêtes : les requêtes sur le contenu (type mots clés), les requêtes sans navigation et les requêtes de navigation. À chaque type de requêtes, le moteur de requêtes WebSQL associe une méthode d'évaluation spécifique.

Les requêtes sur le contenu :

Les requêtes sur le contenu permettent de rechercher des documents suivant des critères mots clés spécifiés via l'opérateur **MENTIONS** dans la section **FROM** ou **WHERE** de la requête. Ces mots clés sont soumis à Yahoo! [Yahoo! site] et Lycos [Lycos site]. Ces deux moteurs renvoient une liste d'urls

correspondant à des réponses potentielles. Les urls de cette liste sont rapatriées par un module spécifique de WebSQL pour constituer une base préliminaire de documents web. Cette base est par la suite évaluée sur les autres critères mots clés spécifiés via l'opérateur **CONTAINS**. Contrairement aux mots clés introduits via l'opérateur **MENTIONS**, les mots clés de l'opérateur **CONTAINS** sont évalués par le moteur SQL du langage WebSQL.

Exemple : Soit la requête ci-dessous qui recherche les urls référencées par des liens contenant dans leur ancre le mot « *applet* ». Ces liens doivent être contenus dans des documents « *java* ».

WebSQL récolte et charge les urls fournies par Yahoo ! et Lycos à la suite de la soumission de la requête mots clés « *java* ». À partir de la table **Anchor**, WebSQL recherche les urls référencées par des liens contenant le mot « *applet* » dans leur ancre.

```
SELECT  l.href
FROM    Document d  SUCH THAT  d MENTIONS «java»
        Anchor l    SUCH THAT  (l.base = d.url)
WHERE   l.label      CONTAINS   «applet»
```

Les requêtes sans navigation :

Une requête sans navigation est une requête dans laquelle les urls des documents à évaluer sont explicitement mentionnées dans la clause **FROM**. L'évaluation des documents mentionnés porte seulement sur le contenu.

Pour une telle requête, WebSQL n'a plus besoin d'interroger Yahoo ! et Lycos pour constituer sa base préliminaire de documents. Les urls mentionnées sont directement rapatriées et les documents correspondants sont ensuite évalués par le moteur SQL.

Exemple : Soit la requête ci-dessous qui recherche l'url et le titre de tous les documents pointés par la page <http://www.lri.fr> avec le mot « *Database* ». Le document HTML correspondant à l'url <http://www.lri.fr> est rapatrié en local.

```
SELECT  d.url, d.title
FROM    Document d  SUCH THAT
        Document y  SUCH THAT  (y.url = «http://www.lri.fr»)
        Anchor l    SUCH THAT  (l.base = y.url)
WHERE   l.label      CONTAINS   «Database»
```

Les requêtes de navigation

L'identification d'une expression régulière de chemins dans une requête *Q* permet à WebSQL de considérer *Q* comme étant une requête de navigation. Les expressions régulières de chemin sont détaillées plus haut (voir section 2 page 24).

WebSQL évalue une requête de navigation en deux étapes. La première étape consiste à regrouper les documents à stocker dans la base et sur lesquels s'effectuera l'évaluation. Les deux cas, vus précédemment pour les deux autres types de requêtes, peuvent se présenter : collecte de documents à la suite de la soumission des mots clés de la requête à Yahoo ! et Lycos ou collecte des documents

correspondant aux urls mentionnées dans la requête.

La seconde étape consiste à interpréter l'expression régulière spécifiée dans la requête. Cette interprétation débute par une exploration du web en partant des documents collectés dans la base. De cette exploration, WebSQL construit, progressivement et de manière itérative, les chemins hypertextes accessibles depuis les documents de sa base. À chaque itération, WebSQL évalue les contraintes exprimées par l'expression régulière sur les chemins construits.

L'alphabet utilisé par WebSQL pour former ses expressions régulières de chemins est constitué de quatre termes : le chemin vide $=$, la référence interne $\mapsto$, le lien entre deux documents du même site web $\rightarrow$ et le lien entre deux documents sur deux sites différents $\Rightarrow$. La définition formelle de cet alphabet est détaillée dans le tableau II.1 page 28.

Catégorie de liens	Symbole	Définition	Formalisation
Références internes	$\mapsto$	regroupent les liens sortant d'un document vers une sous partie de ce même document	<code>l.href == l.base + '#' + ref</code>
Liens intra-site	$\rightarrow$	regroupent les liens pointant vers des documents du même site	<code>DNS(l.href) == DNS(l.base)</code>
Liens extra-sites	$\Rightarrow$	regroupent les liens pointant vers des documents situés sur un site différent	<code>DNS(l.href) != DNS(l.base)</code>

- L'addition (+) représente l'opération de concaténation de chaînes de caractères

- `ref` est une chaîne de caractères représentant la référence

- `DNS` est une fonction qui renvoie le nom de domaine contenu dans une url

TAB. II.1 – Tableau de l'alphabet utilisé dans les expressions régulières du langage WebSQL

Exemple : Soit la requête ci-dessous qui recherche les urls et les titres des documents contenant les mots clés « *intelligence artificielle* » et reliés à la page «*http://www.lri.fr*» par seulement des liens intra-site dont le chemin n'excède pas trois liens :

```

SELECT  d.url,d.title
FROM    Document d  SUCH THAT  http://www.lri.fr= | → | →→ | →→→ d
WHERE   d            CONTAINS  «intelligence artificielle»
```

2.2 W3QL

W3QL [Konopnicki et Shmueli 1998] est un langage de requêtes web prenant en charge les requêtes de navigation. Ce langage est utilisé pour interroger les documents web du système W3QS [Ko-

nopnicki et Shmueli 1995] dans le cadre du projet WebSuite [Beer et al. 1998].

La philosophie et l'expressivité du langage W3QL sont très proches du langage WebSQL (voir section 2.1 page 26). Il n'existe donc pas de différences majeures entre les deux langages.

2.3 SQUEAL

SQUEAL [Spertus et Stein 2000a, Spertus et Stein 2000b, Spertus 1998] est un langage de requêtes de navigation fondé sur le modèle relationnel. Le schéma relationnel utilisé dans SQUEAL se décline en cinq tables relationnelles : **Page**, **Tag**, **Att**, **Link** et **Parse**. Ces tables sont décrites ci-dessous :

```
Page(url :string, contenu :string, taille :integer, modif :date)
Tag(url :string, identifiant :integer, nom :string, position_debut :integer, position_fin :integer)
Att(tag_id :integer, nom :string, valeur :string)
Link(source_url :string, destination_url :string, anchor :string)
Parse(url_value :string, host :string, port :integer, path :string, ref :string, depth :integer)
```

où :	Page	est la table de stockage du contenu textuel d'un document.
	Tag	est la relation de représentation des balises HTML d'un document. Une balise HTML est décrite par : son nom (ex : BODY), sa position de début dans le document et sa position de fin dans le document.
	Att	est la relation complémentaire de la table Tag . Son contenu permet de répertorier les attributs de chaque balise de la table Tag . Un attribut se compose d'un nom et d'une valeur.
	Link	Cette relation décrit les liens hypertextes entre deux documents.
	Parse	cette table décompose les urls des documents de la table Page en leurs sous parties : le nom de domaine (host), le numéro de port de connection, le chemin du fichier à partir du serveur et la référence contenue dans l'url si elle existe.

Les cinq tables précédentes modélisent le web par une base de données virtuelle, de manière similaire au modèle WebSQL. Le peuplement et la matérialisation de cette base s'effectue au moment de l'évaluation d'une requête en faisant appel, si nécessaire, au moteur de recherche AltaVista [AltaVista site]. Deux différences majeures distinguent cependant SQUEAL des langages WebSQL et W3QL.

La première différence réside dans le schéma relationnel de représentation du web. En effet, les tables **Page**, **Tag** et **Att** décomposent le document en ses sous parties élémentaires à savoir ses balises HTML. Cette décomposition confère aux requêtes du langage SQUEAL une finesse d'expressivité et un niveau de granularité difficiles à atteindre avec les schémas proposés par le modèle relationnel du langage WebSQL.

La seconde différence réside dans les requêtes de navigation. En effet, SQUEAL ne propose aucun mécanisme déclaratif de formulation d’expressions régulières de chemins. Les requêtes de navigation peuvent néanmoins être simulées avec de multiples jointures sur la table `Link`. Trouver une combinaison de jointures équivalente à une expression régulière de chemin est une tâche ardue même si cette expression régulière paraît simple. De plus, les jointures entraînent une perte conséquente en performances de calcul pour évaluer une simple requête de navigation.

En résumé :

Les langages de requêtes de navigation web exploitent la puissance du modèle relationnel et l’expressivité du langage SQL pour offrir un moyen simple mais efficace pour rechercher des documents web. Comparativement à des requêtes mots clés simples, les langages de navigation possèdent une expressivité accrue qui permet de formuler des contraintes sur les chemins web. L’inconvénient majeur avec les langages de cette catégorie est l’incapacité de fournir un classement (*Ranking*) « objectif » des réponses. En effet, le modèle relationnel sur lequel s’appuient les langages de navigation et la nature SQL de ces langages reposent sur une sémantique d’évaluation stricte avec uniquement deux possibilités : le document satisfait la requête posée ou il ne la satisfait pas. Il n’existe alors aucun moyen de distinguer les documents satisfaisant la requête posée.

Notons, par ailleurs, la forte dépendance qui lie les langages de navigation web au langage SQL. Par exemple, les langages de navigation s’appuient uniquement sur les outils SQL de traitements textuels pour rechercher des mots clés dans un document. Ces outils sont peu nombreux et offrent des possibilités de traitement extrêmement limitées.

La précédente dépendance s’étend également à la sémantique des requêtes des langages de navigation web. En effet, la sémantique d’une requête de cette catégorie est définie par l’interprétation du moteur SQL de la requête `[SELECT...FROM...WHERE...]`. Cette interprétation est figée et ne peut être adaptée pour des besoins spécifiques.

Par exemple, supposons que l’occurrence des mots clés «*soap protocol*» dans le titre ou dans l’ancrage des liens entrants d’un document HTML est un critère suffisant pour affirmer que ce document est un document de cours sur le protocole SOAP mais que ce critère ne permet pas de couvrir tous les documents de cours sur le web traitant du protocole SOAP. Supposons également que 50% des documents de cours SOAP sur le web contiennent les mots clés «*web service*» dans leur contenu. Il est clair que le premier critère est le plus significatif dans la pertinence de la recherche de documents de cours sur SOAP. Un poids important doit être accordé à son évaluation. Ce poids reste cependant insuffisant pour couvrir le maximum de documents pertinents. Le premier critère doit nécessairement être combiné au second. Pour être efficace, cette combinaison doit refléter l’importance du premier critère sur le second. Les langages de navigation et les langages sur le contenu ne permettent pas de tenir compte de ce type de modulations dans l’interprétation d’une requête.

3 Langages de gestion et de restructuration de sites web

De manière similaire aux langages précédents, les langages de gestion et de restructuration de sites web proposent les options classiques de recherche de documents sur le web (par contenu et par navigation). Ils sont cependant peu utilisés pour ces tâches là. Les langages de cette catégorie sont plutôt utilisés pour créer, gérer, administrer et maintenir des sites web. Ils s’apparentent donc davantage à des outils d’intégration et de médiation de sources de données tels que le web, les bases

de données en ligne, *etc.* La création, la gestion, l'administration et la maintenance d'un site web consistent à créer un nouveau site à partir de données existantes contenues dans des documents web ou dans des sources en ligne. Le résultat de cette intégration offre une vue nouvelle et différente sur les données précédentes. Ce processus d'intégration s'effectue souvent en plusieurs étapes.

Dans un premier temps, l'administrateur du site web définit ses besoins en formulant une requête appelée *requête de création du site*. Cette requête définit le squelette et le schéma du site à créer. Elle regroupe pour cela les descriptions de chaque structure de groupe de documents à générer ainsi que leur manière de s'interconnecter entre-eux.

La deuxième étape consiste à formuler, dans la même requête de création ou dans une nouvelle requête, les conditions et les critères que doivent satisfaire les documents web existants ou les sources de données pour être intégrés. L'évaluation de ces critères permet d'identifier les documents web et les sources de données conformes aux besoins du site. Le contenu de ces documents et les données des sources retenues sont ensuite analysés pour extraire les parties d'informations à publier dans le site. La publication de ces informations peut être vue comme une matérialisation du schéma du site à créer car elle permet d'associer un nouveau contenu au squelette du site précédemment défini.

La deuxième étape peut être répétée régulièrement pour mettre à jour le site. La réévaluation des critères d'intégration permettent de refléter sur le contenu du site les éventuels changements survenus dans les documents et les sources intégrées. Le choix de la fréquence de cette mise à jour est déterminé par l'administrateur du site.

Notons, par ailleurs, que les étapes de conception, décrites précédemment, définissent le site créé par un ensemble de groupes de documents. La particularité de chaque groupe réside dans la structure commune et le schéma unique partagés par ses documents. Cette particularité permet à l'administrateur du site d'automatiser et d'uniformiser certains traitements sur la totalité du site ou sur une de ses parties. Par exemple, certains langages de gestion de sites web proposent une panoplie de modèles de présentation HTML (feuilles de style CSS par exemple) pour définir l'apparence et l'esthétique du site ou d'une de ses parties. L'application de ces modèles est souvent prise en charge par des outils déclaratifs proposés par le langage même. À défaut de ces outils déclaratifs, l'automatisation de l'application des précédents modèles de présentation HTML peut facilement s'effectuer par des scripts spécialement conçus à cet effet.

Nous avons retenu de la littérature scientifique trois langages pour illustrer la catégorie des langages de gestion de sites web : WebOQL [Arocena et Mendelzon 1998], PENELOPE & ULIXES [Atzeni et al. 1997] et STRUQL [Fernandez et al. 1998b]. Ces langages s'inspirent du paradigme objet pour offrir des outils de gestion de sites web.

3.1 Langages fondés sur le paradigme objet

Les langages fondés sur le paradigme objet s'appuient sur des modèles objets conformes, pour la plupart, aux recommandations du groupe *ODMG (Object Data Management Group)* [Cattell et al. 2000, Bancilhon et Ferran 1994] sur la modélisation objet. Ces recommandations, également utilisées dans les bases de données et les langages de programmation orientés objet, garantissent aux langages de cette catégorie certaines propriétés telles que la portabilité sur différentes plateformes, l'interopérabilité avec d'autres systèmes, *etc.* C'est ainsi que la syntaxe des langages fondés sur le

paradigme objet est souvent dérivée du langage OQL.

La modélisation explicite de la structure arborescente d’un document web constitue la principale caractéristique du modèle objet employé par les langages de cette catégorie. Un document web qui existe déjà ou qui doit être créé est vu par ces langages comme un objet représentable par un arbre. Cet arbre traduit la hiérarchie entre les différents composants internes d’un document.

Créer de nouveaux documents consiste donc à leur définir, via une ou plusieurs requêtes, des arbres structurés contenant, dans un premier temps, des nœuds vides. Ces nœuds sont par la suite remplis avec des données provenant de l’évaluation d’une autre requête sur les documents web existants ou provenant du résultat de l’intégration de certaines sources de données. Il s’agit ensuite de définir l’interconnexion entre les documents créés pour constituer le site web.

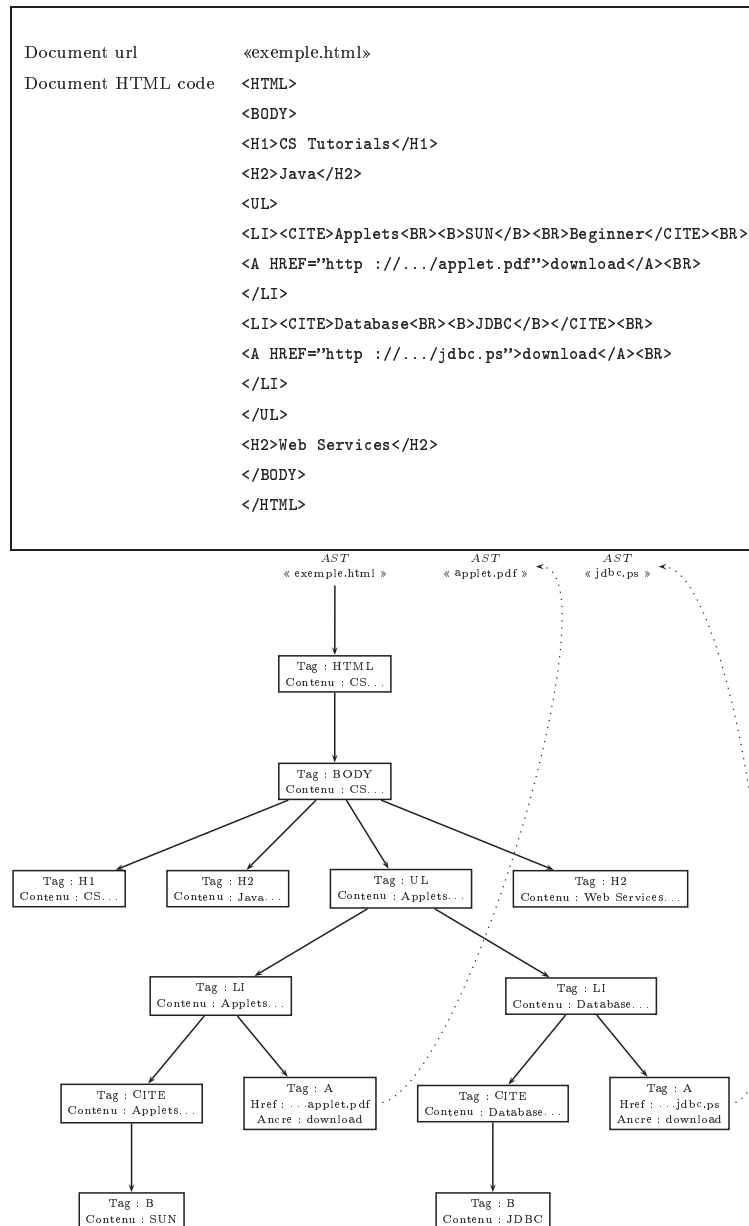
L’opération de restructuration d’un site s’effectue par requête. La requête de restructuration permet de cibler deux niveaux différents : les documents du site et les composants internes des documents. La restructuration des documents du site consiste à changer la manière avec laquelle les documents sont connectés. La restructuration des composants d’un document consiste à modifier la structure interne de son arbre représentatif. Cette modification permet : de supprimer une ou plusieurs branches de l’arbre du document, de réorganiser la hiérarchie interne des nœuds de cet arbre, *etc.*

3.1.1 WebOQL

WebOQL [Arocena 1997, Arocena et Mendelzon 1998] est un langage orienté objet qui s’inscrit dans la catégorie des langages d’aide à la création et à la gestion de sites web. WebOQL a été développé dans l’optique de proposer les fonctionnalités de gestion de sites web absentes dans le langage WebSQL (voir 2.1 page 26).

WebOQL modélise un document HTML par un arbre appelé arbre syntaxique abstrait (*abstract syntax tree*) et que nous notons *AST*. WebOQL utilise pour cela un analyseur syntaxique guidé par le type MIME [Freed et Borenstein 1993, Freed et Borenstein 1996a, Freed et Borenstein 1996b] du document considéré. L’arbre *AST* associé à un document HTML traduit, par exemple, la relation d’inclusion entre balises HTML.

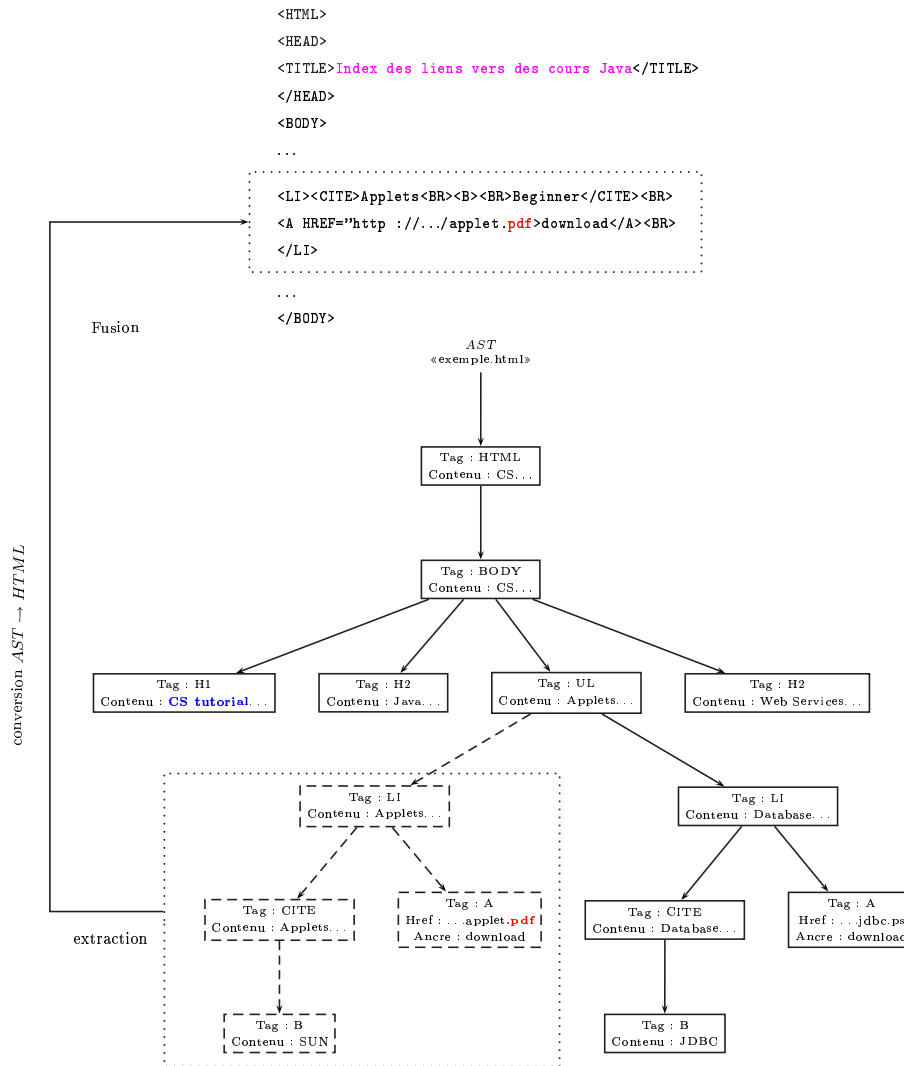
Les nœuds de l’arbre *AST* associé à un document représentent les composants internes de ce document sous forme d’enregistrements. Chaque enregistrement identifie les propriétés d’un composant par une suite de sections de type attribut-valeur. Pour un document HTML, les nœuds représentent les balises HTML. Chaque balise est identifiée par ses propriétés : nom de la balise, son contenu, sa position dans le document, *etc.* La figure II.2 page 33 illustre la transformation d’un document HTML en un arbre *AST*.

FIG. II.2 – Représentation *AST* d'un document HTML sous WebOQL

Pour illustrer les possibilités de création et de restructuration d'un document avec le langage WebOQL, considérons l'exemple suivant :

Créer un nouveau document regroupant tous les tags `<LI>` contenant un lien vers un document de type « *pdf* » et provenant de documents contenant dans leur titre « *cs tutorial* ». Le nouveau document ainsi créé aura pour titre « *Index des liens vers des cours Java* ».

Pour évaluer la précédente requête, WebOQL repère, en premier lieu, les documents contenant dans leur arbre *AST* des nœuds de balise `<A>` accessibles depuis un nœud `<LI>` et ayant le mot

FIG. II.3 – Évaluation d'une requête WebOQL sur l'arbre *AST* de la figure II.2 page 33

«pdf» dans le contenu de la propriété **href**. Les documents trouvés sont analysés pour extraire les sous arbres de nœuds satisfaisant les critères précédents. Ces sous arbres sont fusionnés dans un nouvel arbre *AST* correspondant au document créé. Un résumé de l'application de ce processus sur le document de la figure II.2 page 33 est schématisé dans la figure II.3 page 34.

3.1.2 PENELOPE & ULIXES

PENELOPE & ULIXES [Atzeni et al. 1997] sont deux langages développés par l'équipe bases de données de l'université de Rome III dans le cadre du projet ARANEUS [ARANEUS 1998]. Ce projet propose une plate-forme d'intégration de sites web fondée sur des techniques de bases de données. L'objectif visé consiste à offrir aux utilisateurs un ensemble de vues structurées d'un ou plusieurs sites web. Les requêtes du langage PENELOPE permettent de définir le schéma du site à créer. Les requêtes du langage ULIXES servent à fournir le contenu à mettre dans le schéma défini par une requête PENELOPE.

Le modèle de données utilisé pour représenter un document est le modèle *ADM* (ARANEUS *Data Model*) [Atzeni et al. 1997]. Dans ce modèle, un document est représenté par un schéma semblable au schéma de définition (interface) d'une classe C++. Ce schéma se compose d'un identifiant unique, de l'url du document et d'un certain nombre d'attributs. Un attribut peut être atomique ou composé. Un attribut atomique est découvert dans la structure même du document et correspond par exemple au texte du document, à une image, un lien, *etc.* Un attribut complexe est défini comme étant une liste ou un ensemble d'attributs atomiques.

L'ordre des étapes de création ou de restructuration d'un site web via des requêtes PENELOPE-ULIXES est inversé par rapport à l'ordre habituel suivi par les autres langages (définition du schéma et de la structure du site → recherche du contenu du site → matérialisation du site). En effet, dans le système PENELOPE-ULIXES la collecte du contenu du futur site est effectuée en premier, ce qui peu paraître légèrement contre-intuitif. Ce contenu est obtenu par l'évaluation d'une requête ULIXES sur l'ensemble des sites et des documents à considérer. Le résultat de cette évaluation est stocké dans des tables relationnelles dont le schéma est défini dans la requête ULIXES.

Une fois les tables relationnelles du contenu du site matérialisées, l'administrateur du site peut définir le schéma *ADM* de son site en fonction du contenu des tables et via une requête PENELOPE.

Exemple : Soit **BibADMSites** l'ensemble des schémas *ADM* de sites bibliographiques donnés. Considérons une application ayant pour but d'indexer les publications scientifiques accessibles à partir de ces sites bibliographiques. Considérons également que ses publications scientifiques sont atteignables de manière identique depuis la page d'entrée (**home**) de chaque site bibliographique par le chemin $\mathcal{P}$ suivant : « *accéder depuis cette page d'accueil du site à l'index des auteurs répertoriés, puis récupérer pour chaque auteur le liens vers une page personnelle. Suivre ce lien et extraire dans la page correspondante les liens vers des documents postscript ou pdf* ».

La requête ULIXES à écrire est :

```

DEFINE TABLE  AuthorPapers(Author,Title,Bibtex,Year)
AS              $\mathcal{P}$  : Site.home.LinkList.link->AuthorPage.LinkList.link
IN             BibADMSites
WHERE          AuthorPage.LinkList.link  CONTAINS «ps» OR «pdf»
               AuthorPage.title           CONTAINS «homepage»
               Site.home.LinkList.name     CONTAINS «index»
               Site.home.LinkList.link     CONTAINS «~»

```

L'évaluation de cette requête produit la table relationnelle **AuthorPapers** dont le schéma est :

```

AuthorPapers(Author,Title,Bibtex,Year)

```

La table **AuthorPapers** est annotée par le chemin $\mathcal{P}$ décrit dans la requête précédente.

La génération d'un nouveau site décrivant les publications scientifiques de chaque auteur de la base **AuthorPapers** par année s'effectue via la requête PENELOPE ci-dessous :

```

DEFINE PAGE  AuthorPublicationsPage
AS           URL : <<http://www.../>> + AuthorPapers.Author + <<_publication.html>>
           title : AuthorPapers.Author + «scientific publications»
           Publications : LIST OF[
                           Year : AuthorPapers.Year,
                           Publications : LIST OF[title : AuthorPapers.Title,
                           Bibtex : AuthorPapers.Bibtex]
                           ]
FROM         AuthorPapers

```

L'efficacité des langages PENELOPE et ULIXES repose sur deux hypothèses que nous considérons contraignantes limitant sérieusement le passage à l'échelle de leur approche.

La première hypothèse considère que les sites à intégrer par les requêtes ULIXES possèdent des structures uniformes et similaires. La seconde hypothèse suppose que les documents ciblés par les requêtes ULIXES sont accessibles depuis des chemins ayant des topologies identiques ou très proches. Ces deux hypothèses rendent la combinaison ULIXES/PENELOPE inadaptée à l'intégration de sites hétérogènes présentant peu de similarités dans leur structuration.

Sur l'exemple précédent, les deux requêtes écrites ne fonctionnent que si le chemin d'accès aux publications scientifiques est le même pour tous les sites bibliographiques. La première hypothèse impose de son côté que les pages d'auteurs sont référencées depuis la page d'accueil de chaque site bibliographique dans une section intitulée «index».

3.1.3 StruQL

STRUQL [Fernandez et al. 1998b, Fernandez et al. 1997a, Fernandez et al. 1997b] est un langage déclaratif dont les requêtes servent à spécifier et générer le contenu de sites web à partir de sources de données existantes. Ce langage a été conçu par les membres des laboratoires AT&T Labs dans le cadre du projet STRUDEL [Fernandez et al. 1998a].

Les documents manipulés par STRUQL sont représentés par des objets modélisés par un graphe *OEM* [Papakonstantinou et al. 1995]. Le modèle *OEM* est le modèle de référence dans la représentation et l'interrogation de données semi-structurées [Abiteboul et al. 1997, Abiteboul 1997, Quass et al. 1995].

Excepté le modèle de données et la syntaxe des requêtes, STRUQL possède un fonctionnement très proche des langages PENELOPE-ULIXES.

4 Conclusion

Nous avons présenté dans cet état de l'art les différentes techniques d'interrogation du web. Nous avons distingué trois catégories de langages de requêtes web : les langages mots clés, les langages de navigation et les langages de restructuration de sites web. Un récapitulatif de la classification des différents langages présentés dans cet état de l'art est montré dans la figure II.4 page 37.

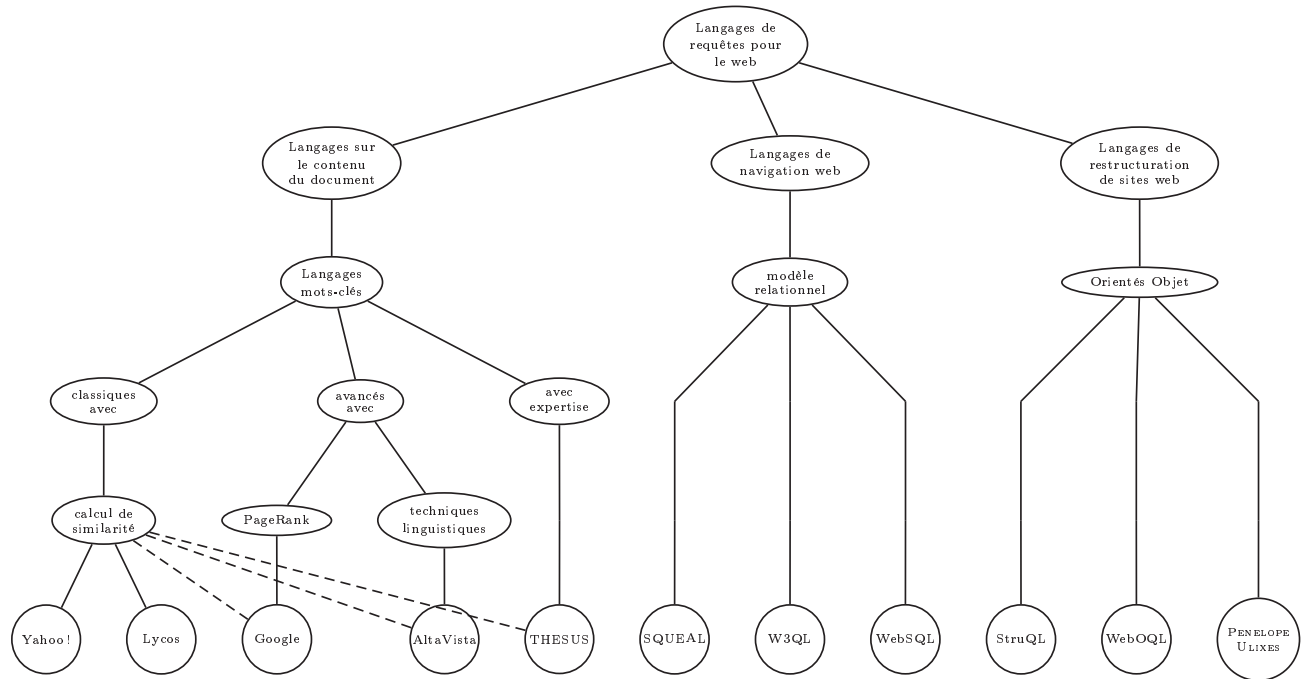


FIG. II.4 – Récapitulatif de la classification des langages de requêtes pour le web

Pour chaque langage présenté, nous avons décrit les caractéristiques suivantes : le modèle de représentation du web et de ses documents, la syntaxe des requêtes, la méthode d'évaluation des requêtes, les fonctionnalités spécifiques, *etc.* Cette description est résumée dans le tableau II.2 page 39.

Dans le cadre de notre thèse, nous avons focalisé nos travaux sur le domaine de la recherche de documents thématiques sur le web. Nous proposons pour cela WeQueL [Mezaour 2003, Mezaour 2004b] un langage web multi-critères offrant des mécanismes déclaratifs pour combiner des requêtes mots clés. L'objectif visé consiste à accroître, d'une part, le pouvoir expressif des requêtes mots clés simples et d'augmenter, d'autre part, la précision dans la recherche de documents web.

Le développement du langage WeQueL a été essentiellement motivé par le peu de possibilités qu'offrent les langages web existants dans l'expression des besoins de recherche. Ce manque d'expressivité s'accroît davantage lorsqu'il s'agit de rechercher des documents thématiques difficilement caractérisables par les requêtes des langages vus dans cet état de l'art. La précision de la recherche documentaire avec les langages existants est très insuffisante pour satisfaire les exigences des applications web actuelles (moteurs de recherche spécialisés, entrepôts thématiques de données, *etc.*).

Une requête WeQueL est une requête construite par combinaison de plusieurs requêtes mots

clés. Son évaluation s’applique sur une modélisation par sections prédéfinies du document web à considérer. Cette modélisation, que nous appelons *WDM : WeQueL Data Model*, consiste à découper un document en un ensemble de sections logiques prédéfinies et délimitées par des balises HTML spécifiques, retenues pour leur intérêt dans une recherche documentaire sur le web. Chaque section contient une chaîne de caractères représentant le contenu textuel correspondant à cette section dans le document nettoyé de toute balise HTML.

Contrairement aux langages de requêtes de navigation, nous n’avons envisagé aucune structure pour modéliser les chemins hypertextes entre documents web. Les besoins de navigation sont par conséquent inexprimables par les requêtes de notre langage WeQueL. Les requêtes de navigation ont d’ailleurs été inutiles dans les exemples applicatifs de test du langage.

Précisons, par ailleurs, que la sémantique d’une requête WeQueL est déterminée par les opérateurs choisis dans la combinaison des requêtes mots clés. Le langage WeQueL propose deux types d’opérateurs : opérateurs logiques et opérateurs numériques. Les opérateurs logiques donnent lieu à des évaluations binaires (*vrai* ou *faux*). Les opérateurs numériques permettent au moteur de requêtes WeQueL de calculer un score d’évaluation et de classer les documents évalués en fonction de ce score.

Dans la construction d’une requête, notre langage permet d’employer à la fois des opérateurs logiques et des opérateurs numériques. Cette souplesse permet aux concepteurs de requêtes WeQueL de moduler de manière déclarative le procédé d’évaluation de la requête finale. On parle dans ce cas d’une *sémantique ajustable* par opposition à la sémantique figée des requêtes de la plupart des langages vus dans cet état de l’art.

Le langage WeQueL est présenté dans le chapitre III page 41.

L'EXISTANT POUR INTERROGER LE WEB

Langages	Modèle de données	Syntaxe	Méthode d'évaluation	Particularités	Requêtes de navigation	Génération de documents web
Mots clés	Vecteur de mots	Mots clés	similarité requête-document	×	×	×
Google	Vecteur de mots	Mots clés	similarité requête-document	PageRank	×	×
Google avancé	Vecteur de mots	Mots clés	similarité requête-document	PageRank + ciblage des sous parties d'un document	×	×
THESUS	Vecteur de mots	Mots clés	similarité requête-document	document $\equiv$ ses liens entrants + WordNet + Ontologie	×	×
WebSQL	tables relationnelles	SQL	moteur SQL	×	✓	×
W3QL	tables relationnelles	SQL	moteur SQL	×	✓	×
SQUEAL	tables relationnelles	SQL	moteur SQL	×	×	×
STRUQL	OEM	OQL-SQL	moteur SQL	×	✓	✓
WebOQL	AST	OQL	moteur OQL	×	✓	✓
PENELOPE	ADM	OQL-SQL	moteur SQL	×	✓	✓
ULIXES	ADM	OQL-SQL	moteur SQL	×	✓	×

NOTRE APPROCHE

WeQueL	<i>WDM</i>	XML	moteur WeQueL	combinaisons avancées des requêtes mots clés	×	×
--------	------------	-----	---------------	--	---	---

TAB. II.2 – Tableau comparatif des langages de requêtes pour le web

WeQueL : *Web Query Language*

Dans ce chapitre, nous présentons notre contribution principale dans le domaine des langages de requêtes pour le web. Nous proposons à cet effet le langage **WeQueL**, un langage de requêtes multi-critères dédié à la recherche de documents thématiques. WeQueL offre la possibilité de combiner des mécanismes descriptifs par mots clés suffisamment fins et variés pour bien caractériser et décrire les documents à rechercher. L'objectif visé est d'accroître de manière significative la précision de la recherche par mots clés sans devoir recourir aux techniques habituelles pour stimuler la qualité des résultats de la recherche, comme par exemple l'apprentissage automatique sur des corpus ou le PageRank [Brin et Page 1998].

Nous avons organisé ce chapitre en trois sections. Dans la première section, nous exposons le contexte qui nous a motivé à développer un langage de requêtes mots clés multi-critères pour décrire les documents souhaités lors d'une recherche donnée sur le web.

Dans la deuxième section, nous décrivons en détail le mode d'emploi du langage WeQueL et ses principales spécificités. Cette description comprend une présentation de la syntaxe des requêtes de notre langage ainsi que la sémantique formelle associée aux requêtes.

Dans la dernière section de ce chapitre, nous montrons le gain en précision des requêtes du langage WeQueL dans diverses recherches thématiques. À cet effet et conformément au protocole expérimental défini, nous avons mené une série de tests expérimentaux pour cibler la recherche de documents sur quatre thématiques différentes : recherche de cours informatiques en-ligne, recherche de pages d'accueil de sites de tourisme, recherche de photos de formule 1 et recherche de documents traitant du risque alimentaire. Sur cette dernière recherche thématique, nous présentons en détail la manière avec laquelle nous avons intégré le langage WeQueL en tant que support de filtrage du processus d'acquisition de données de l'entrepôt du projet « eDot » (cf. section 4 page 9).

1 Motivations

Dans l'état de l'art précédent sur les langages de requêtes pour le web (voir chapitre II page 17), nous avons présenté deux catégories de langages souvent employées pour la recherche documentaire sur le web : la catégorie des langages mots clés sur le contenu du document et la catégorie des langages type SQL pour exprimer des besoins de navigation web.

Contrairement aux requêtes de la catégorie des langages du web de type SQL, les langages mots clés possèdent une syntaxe familière et une construction intuitive rendant leurs requêtes simples et faciles à utiliser. De plus, les techniques élémentaires de la recherche d'informations (*Information Retrieval*), telles que le calcul de similarité ou le F-score [Van-Risbergen 1979], sont parfaitement applicables lors de l'évaluation d'une requête mots clés sur un document du web. Cet avantage permet aux systèmes à base de langages mots clés de proposer un classement (*Ranking*) fort utile pour apprécier le degré de pertinence de chaque réponse par rapport à la requête soumise.

Il nous a donc paru légitime de construire les requêtes de notre langage sur la base de requêtes mots clés basiques afin de pouvoir y intégrer les principales mesures de recherche d'informations (F-score [Roche 2003], précision, rappel, cosinus, ...). Ainsi, après évaluation des requêtes de notre langage, notre système est capable de proposer un classement « *objectif* » des réponses à une requête donnée en fonction du score calculé pour chaque document évalué.

Nous avons montré dans le précédent chapitre le manque évident d'expressivité des langages mots clés simples. En effet, les seules possibilités qui s'offrent à un utilisateur pour décrire ses besoins se limitent, dans la plupart des cas, à l'énumération d'une liste de mots clés à rechercher dans le contenu textuel des documents à évaluer. Ce contenu textuel est généralement modélisé par un vecteur de mots qui ne permet pas de retrouver la provenance d'origine des mots dans la structure du document (titre, tableau, lien, ...). Il est donc impossible de cibler la recherche des mots clés d'une requête sur une partie spécifique de la structure d'un document. En l'absence de cette possibilité, la description résultante est souvent vague et les réponses données sont nombreuses et peu précises (ne correspondent pas forcément aux documents souhaités).

Par ailleurs, nous avons constaté un manque de souplesse des langages mots clés classiques et des langages de navigation dans l'interprétation de leurs requêtes. En effet, le procédé d'évaluation des requêtes des deux catégories précédentes est défini en interne par leurs systèmes respectifs d'interprétation et d'évaluation des requêtes. Il est quasiment impossible de modifier ce procédé d'évaluation de requêtes pour l'adapter à des besoins spécifiques. Par exemple, dans le cadre du projet « eDot » (cf. section 4 page 9), les experts microbiologistes ont déterminé que, parmi les critères de pertinence possibles pour le domaine du risque alimentaire, la présence de certaines combinaisons de mots clés de l'ontologie Sym'Previus [SymPrevius site] dans des tableaux HTML représentait un fort potentiel dans l'estimation de la pertinence d'un document. Outre le fait qu'ils ne permettent pas la combinaison de plusieurs critères de recherche à la fois, les langages web existants ne proposent aucun moyen déclaratif pour accorder une plus grande importance à l'évaluation du critère précédent par rapport aux autres critères définis par les experts. Cette « rigidité » dans l'évaluation des requêtes peut, dans certains cas, être à l'origine de l'imprécision des réponses données.

Pour combler ces lacunes, nos efforts de développement du langage WeQueL ont porté sur quatre

points essentiels :

1. Une modélisation de documents qui préserve la provenance des mots dans le contenu du document.
2. Un ensemble varié de requêtes atomiques ciblant différentes parties de la structure d'un document (exemple : le titre d'un document, son url, ses liens sortants, ses tableaux, ...).
3. Un ensemble de requêtes atomiques pour décrire des aspects autres que le contenu d'un document, exemple : le type MIME (Multipurpose Internet Mail Extensions) d'un document, les propriétés de son url, ses liens entrants *etc.*
4. Une syntaxe déclarative permettant de combiner à la fois différentes requêtes atomiques des deux ensembles précédents tout en spécifiant la manière avec laquelle chaque requête combinée est évaluée.

2 Présentation générale du langage WeQueL

Nous avons défini un langage de requêtes de type «attribut-valeur» qui permet à un utilisateur de combiner, à l'aide de divers opérateurs, plusieurs critères pour caractériser et décrire les documents qui l'intéressent. Chaque critère spécifié dans une requête permet de cibler la recherche de mots-clés sur une partie déterminée de la structure d'un document (exemple : son titre, son url, le voisinage de ses liens sortants, ...) ou de caractériser une propriété particulière d'un document du web (exemple : la longueur de l'url, son type MIME, les mots dans ses liens entrants, ...). Ces critères de caractérisation constituent les requêtes atomiques de notre langage à partir desquelles il est possible d'exprimer des requêtes plus complexes.

Une requête atomique est une requête mots-clés de type « attribut = valeurs » où l'attribut peut être choisi parmi un ensemble d'attributs prédéfinis à l'avance. Chaque attribut prédéfini désigne, de manière unique, une seule propriété ou une seule partie de la structure d'un document. Pour simplifier le vocabulaire de ce chapitre, nous désignons une partie de la structure d'un document ou une de ses propriétés par une *section*. Les sections que nous avons retenues dans la caractérisation d'un document du web sont : le titre, l'url, la longueur de l'url, le corps (contenu de la balise <BODY>), le voisinage des liens entrants et sortants, le titre du document parent, les objets HTML, les mots clés (contenu de l'attribut `content` de la balise <meta name="keywords">) et le type MIME [Freed et Borenstein 1993, Freed et Borenstein 1996a, Freed et Borenstein 1996b].

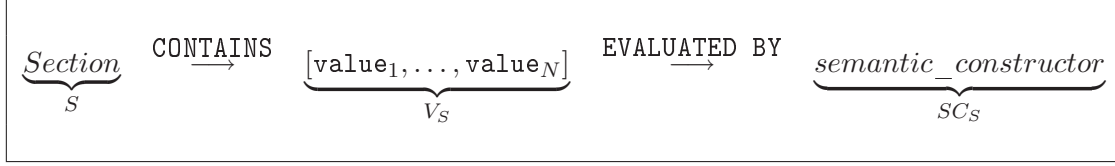
Les valeurs d'une requête atomique peuvent être multiples et doivent être séparées par des virgules. Chaque valeur peut contenir un ou plusieurs mots clés séparés par le caractère blanc.

En résumé, une requête atomique WeQueL permet à l'utilisateur de décrire les mots clés que doivent contenir les documents souhaités dans la section ciblée par cette requête atomique. En formulant et combinant plusieurs requêtes atomiques différentes, l'utilisateur peut décrire divers aspects à la fois des documents souhaités dans une requête complexe.

3 Syntaxe du langage

3.1 Syntaxe d'une requête atomique

Une requête atomique, notée $\mathcal{Q}_a$, s'écrit dans le langage WeQueL sous la forme suivante :



- "CONTAINS" et "EVALUATED BY" sont deux mots réservés du langage WeQueL

TAB. III.1 – Syntaxe d’une requête atomique WeQueL

Trois parties distinctes composent une requête atomique WeQueL :

1. un attribut pour cibler la recherche sur la section S des documents à évaluer ;
2. un ensemble de valeurs V_S où les valeurs value_i sont séparées par des virgules ;
3. une méthode d’évaluation des valeurs de l’ensemble V_S , appelée constructeur sémantique SC_S .

3.1.1 L’attribut d’une requête atomique

L’attribut d’une requête atomique désigne la section du document à cibler lors de la recherche des mots clés de chaque valeur de l’ensemble V dans les documents à évaluer. Rappelons qu’une section désigne une partie de la structure d’un document ou une de ses propriétés.

Le langage WeQueL offre neuf attributs prédéfinis pour construire une requête atomique :

1. *url* : permet de cibler la recherche des mots clés de chaque valeur de l’ensemble V_S sur les urls des documents à évaluer.
2. *page_title* : permet de cibler la recherche des mots clés de chaque valeur de l’ensemble V_S sur le titre des documents à évaluer. Pour un document HTML, le titre est défini par l’union des mots contenus entre les balises : `<title>` et `</title>`, `<h1>` et `</h1>` et par la valeur de l’attribut « **content** » de la balise `<meta name="title">`.
3. *title_incoming_page* : permet de cibler la recherche des mots clés de chaque valeur de l’ensemble V_S sur le titre des documents pointant sur les documents à évaluer. Pour le cas HTML, les titres des documents pointant sur les documents à évaluer sont déterminés par l’union des mots contenus dans les balises décrites précédemment.
4. *keywords* : permet de cibler la recherche des mots clés de chaque valeur de l’ensemble V_S sur la partie « mots clés » des documents à évaluer. La partie « mots clés » d’un document HTML est déterminée par la valeur de l’attribut « **content** » de la balise `<meta name="keywords">`.
5. *outgoing_links* : permet de cibler la recherche des mots clés de chaque valeur de l’ensemble V_S sur le voisinage de mots des liens sortants des documents à évaluer. Ce voisinage est constitué à partir de l’ancree du lien ainsi qu’à partir d’une fenêtre de k mots ($k = 10$ dans nos expérimentations) pris avant et après la balise `<a>`.
6. *incoming_links* : permet de cibler la recherche des mots clés de chaque valeur de l’ensemble V_S sur le voisinage des mots des liens entrants des documents à évaluer.
7. *url_length* : permet de restreindre l’évaluation des documents proposés aux seuls documents ayant une url satisfaisant la contrainte de longueur spécifiée comme valeur de la requête atomique. La spécification d’une contrainte de longueur s’effectue par l’emploi de l’une des expressions réservées suivantes : « *atmost*[l] » ou « *atleast*[l] ». La sémantique associée à chacune de ses expressions est montrée dans la section 4.1.1 page 56.

8. *object* [*type* = '*T*'] : permet de cibler la recherche des mots clés de chaque valeur de l'ensemble V_S sur le contenu textuel de l'objet T lorsqu'il existe dans les documents à évaluer. Les types d'objets supportés par le langage WeQueL sont ceux définis dans le langage HTML [Berners-Lee et Connolly 1995, Raggett et al. 1999] : *applet*, *script*, *table*, *image* *etc.* Dans nos expérimentations, nous avons essentiellement utilisé deux types : '*table*' et '*list*'. Ces objets sont déterminés par les valeurs des balises : `<table>`, `<dl>`, `<ul>` et `<ol>` dans les documents HTML.
9. *MIME* : permet de restreindre l'évaluation des documents aux seuls documents possédant un type MIME parmi les types MIME spécifiés dans les valeurs de la requête atomique.

Remarque 3.1 Nous notons l'ensemble de ces neuf attributs par WA : *WeQueL Attributes*.

3.1.2 Les valeurs V_S d'une requête atomique

L'ensemble V_S d'une requête atomique contient les valeurs à rechercher dans la section désignée par la requête atomique Q_a de chaque document à évaluer. Cet ensemble se compose d'une suite de N valeurs séparées par des virgules. Chaque valeur comporte un ou plusieurs mots clés. Dans le cas où il y a plusieurs mots clés dans une même valeur, ceux-ci sont séparés par des caractères blancs :

$$\text{value}_{i,i=1..N} \equiv " w_1^i w_2^i \dots w_{k-1}^i w_k^i "$$

Exemple : L'ensemble suivant « `cours java,cours cplusplus,informatique` » comporte trois valeurs. Les deux premières valeurs contiennent deux mots clés, la dernière valeur contient un seul mot clé.

3.1.3 Le constructeur sémantique SC d'une requête atomique

Le constructeur sémantique SC détermine la manière avec laquelle le moteur de requêtes WeQueL évalue la requête atomique Q_a sur chaque document à évaluer. Plus précisément, le constructeur sémantique d'une requête atomique Q_a peut être vu comme une fonction qui attribue un score de pertinence pour chaque document évalué. Le calcul de ce score est spécifique à chaque constructeur sémantique et dépend uniquement du nombre de valeurs de l'ensemble V_S retrouvées dans la section S ciblée par Q_a dans le document évalué. Le langage WeQueL propose deux types de constructeurs sémantiques : constructeurs logiques et constructeurs numériques.

Constructeur sémantique logique : Un constructeur sémantique logique permet à un utilisateur d'associer une interprétation propositionnelle des valeurs de l'ensemble V_S . Dans ce cas, chaque valeur ($\text{value}_{i,i=1..N}$) représente une variable propositionnelle évaluée à *vrai* ou à *faux*. Pour un document d donné, la variable propositionnelle associée à la valeur « valeur_i » n'est évaluée à *vrai* que si tous les mots clés $w_{t=1..k}^i \in \text{value}_i$ sont présents dans la section S du document d désignée par la requête Q_a . De plus, cette présence dans la section S doit préserver l'ordre d'occurrences dans lequel les mots clés $w_{t=1..k}^i \in \text{value}_i$ sont déclarés dans la valeur « valeur_i ». Les détails de la sémantique d'une requête atomique employant un constructeur logique sont donnés dans la section 4.2.2 page 59.

Dans l'implantation actuelle du langage WeQueL, nous proposons cinq constructeurs logiques différents :

- `Disjunct()` : permet d'appliquer une disjonction logique entre toutes les variables propositionnelles (associées aux valeurs de l'ensemble V_S) de la requête atomique Q_a .

- **Conjunct()** : permet d’appliquer une conjonction logique entre toutes les variables propositionnelles de la requête atomique $\mathcal{Q}_a$.
- **Atleast(k)**, **AtMost(k)**, **Exactly(k)** où $(0 \leq k \leq N)$: permettent d’appliquer des contraintes de cardinalités sur le nombre de valeurs de l’ensemble V_S à retrouver dans les documents à évaluer.

Remarque 3.2 *Les requêtes atomiques définies avec un constructeur sémantique logique renvoient la valeur 0 ou 1 à la suite de leur évaluation. Ce mode binaire impose une forte contrainte sur les documents à évaluer. En effet, pour totaliser un score non nul (évaluation à 1 : vrai), le document doit satisfaire toutes les conditions exprimées par le constructeur logique.*

Constructeur sémantique numérique : Un constructeur sémantique numérique permet de définir une méthode d’évaluation des valeurs de l’ensemble V_S plus souple que les méthodes définies par les constructeurs logiques. En effet, plutôt que de renvoyer une valeur booléenne binaire, l’interprétation proposée consiste à traduire la présence des valeurs « value_i », dans la section S du document à évaluer, par un score numérique compris entre 0 et 1. Le score numérique ainsi calculé permet de quantifier le degré de correspondance entre le contenu de la section S du document évalué et la description faite par l’utilisateur dans la requête atomique $\mathcal{Q}_a$. Ce mode d’évaluation s’inspire des techniques classiques de recherche d’information (*Information Retrieval*). Les détails de la sémantique d’une requête atomique à interprétation numérique sont donnés dans la section 4.2.2 page 59.

Nous proposons quatre constructeurs sémantiques numériques dans la version actuelle du langage WeQueL :

- **Precision()** : permet de mesurer le taux de correspondance de la section du document désignée par la requête atomique $\mathcal{Q}_a$ avec l’ensemble V_S .
- **Recall()** : permet de mesurer la proportion de valeurs de l’ensemble V_S couvertes par les mots de la section du document désignée par la requête atomique $\mathcal{Q}_a$.
- **Fscore()** : ce constructeur offre la possibilité de combiner les deux précédents constructeurs pour apprécier à la fois la précision et la couverture de la section désignée par la requête atomique $\mathcal{Q}_a$ dans le document évalué.
- **Cosine()** : permettent de calculer le cosinus de l’angle défini par le vecteur des mots des valeurs de l’ensemble V_S et le vecteur des mots de la section du document désignée par la requête atomique $\mathcal{Q}_a$.

Remarque 3.3 *La nature d’un document web est définie de manière unique par son type MIME. Pour un document donné, ce type ne prend qu’une et une seule valeur à la fois parmi les types MIME définis dans [Freed et Borenstein 1993, Freed et Borenstein 1996a, Freed et Borenstein 1996b]. Par conséquent, et pour des raisons de cohérence, la requête atomique portant sur le type MIME d’un document ne s’utilise qu’avec le constructeur logique **Exactly(1)**.*

Remarque 3.4 *De par sa construction, la requête atomique `url_length` définit des contraintes sur la longueur d’une url indépendamment des mots qui composent cette url. Il n’est donc pas nécessaire de spécifier sa partie valeur (l’ensemble V_S). De plus, seuls les constructeurs **Atleast(l)**, **AtMost(l)** et **Exactly(l)** peuvent être utilisés avec cette requête atomique.*

3.1.4 Exemples de requêtes atomiques du langage WeQueL

Dans les exemples ci-dessous, les constructeurs sémantiques des requêtes atomiques ne sont pas précisés.

Exemple 1 : Soit Q_a^{ex1} une requête atomique qui cible la recherche des valeurs « `cours java` » et « `cours cpp` » sur l'url des documents à évaluer :

$$Q_a^{ex1} : \boxed{url \xrightarrow{\text{CONTAINS}} [\text{cours java}, \text{cours cpp}] \xrightarrow{\text{EVALUATED BY}} SC1}$$

TAB. III.2 – Exemple d'une requête atomique ciblant l'url des documents à évaluer

Exemple 2 : Soit Q_a^{ex2} une requête atomique ciblant la recherche des valeurs « `support cours` », « `java` » et « `cpp` » sur le titre des documents à évaluer :

$$Q_a^{ex2} : \boxed{title \xrightarrow{\text{CONTAINS}} [\text{support cours}, \text{java}, \text{cpp}] \xrightarrow{\text{EVALUATED BY}} SC2}$$

TAB. III.3 – Exemple d'une requête atomique ciblant le titre des documents à évaluer

Exemple 3 : Soit Q_a^{ex3} une requête atomique ciblant la recherche des valeurs « `cours` », « `java` », « `cpp` » et « `tutoriel` » sur la partie « mots clé » des documents à évaluer :

$$Q_a^{ex3} : \boxed{keywords \xrightarrow{\text{CONTAINS}} [\text{cours}, \text{java}, \text{cpp}, \text{tutoriel}] \xrightarrow{\text{EVALUATED BY}} SC3}$$

TAB. III.4 – Exemple d'une requête atomique ciblant le titre des documents à évaluer

Exemple 4 : Soit Q_a^{ex4} une requête atomique ciblant la recherche des valeurs « `cours java` » et « `cours cpp` » sur les liens entrants des documents à évaluer :

$$Q_a^{ex4} : \boxed{incoming_links \xrightarrow{\text{CONTAINS}} [\text{cours java}, \text{cours cpp}] \xrightarrow{\text{EVALUATED BY}} SC4}$$

TAB. III.5 – Exemple d'une requête atomique ciblant les mots apparaissant dans les liens entrants

3.2 Combinaison de requêtes atomiques

Pour cibler plusieurs sections d'un document, le langage WeQueL permet de combiner différentes requêtes atomiques. Cette combinaison consiste à agréger, dans une même entité, les requêtes atomiques ciblant les sections d'une catégorie *MIME* donnée de documents. L'influence de chaque requête atomique agrégée sur l'évaluation globale de la combinaison ainsi formée, est pondérée par

un poids « α » choisi par l'utilisateur. Par ce mécanisme de poids, l'utilisateur est capable de contrôler déclarativement la contribution de chaque requête atomique dans le calcul final du score de la combinaison sur chaque document à évaluer.

Une combinaison WeQueL de requêtes atomiques, notée $\mathcal{C}$, comporte deux parties duales : la partie nécessaire et la partie pondérée. En supposant que N soit le nombre de requêtes atomiques dans la partie nécessaire et M soit le nombre de requêtes atomiques de la partie pondérée, la syntaxe d'une combinaison $\mathcal{C}$ du langage WeQueL est définie comme suit :

$$\mathcal{C} : 1 \cdot \underbrace{Q_{MIME} \cdot Q_a^1 \cdot Q_a^2 \dots Q_a^{N-1} \cdot Q_a^N}_{\text{Partie nécessaire}} \langle \underbrace{\alpha_1 \cdot \widehat{Q}_a^1, \dots, \alpha_M \cdot \widehat{Q}_a^M}_{\text{Partie pondérée}} \rangle$$

TAB. III.6 – Syntaxe d'une combinaison WeQueL par catégorie de documents

$$\text{où : } \left\{ \begin{array}{ll} Q_{MIME} : & \text{est la requête atomique sur le type MIME des documents visés par la} \\ & \text{combinaison } \mathcal{C} \\ \forall i = 1..N : & Q_a^i \text{ est une requête atomique avec un constructeur sémantique logique } . \\ \forall j = 1..M : & \widehat{Q}_a^j \text{ est une requête atomique dont l'influence (sur le score d'évaluation de} \\ & \mathcal{C}) \text{ est pondérée par un poids } \alpha_j. \text{ Le langage WeQueL n'impose aucune} \\ & \text{restriction sur le choix du constructeur sémantique à employer pour } \widehat{Q}_a^j \\ \forall \alpha_j \cdot \widehat{Q}_a^j : & \sum_{j=1}^M \alpha_j = 1 \end{array} \right.$$

Pour simplifier l'évaluation des combinaisons WeQueL, nous avons fixé six règles pour combiner des requêtes atomiques :

1. Une combinaison doit contenir une et une seule requête atomique *MIME* portant sur le type MIME des documents recherchés.
2. Une combinaison ne peut contenir deux requêtes atomiques différentes portant sur la même section. Par exemple, une combinaison WeQueL ne peut pas contenir deux requêtes atomiques portant sur le titre des documents (attribut « *page_title* »).
3. Les requêtes atomiques combinées avec une requête sur un type *MIME* donné doivent cibler des sections existantes dans les documents de ce type *MIME*. Par exemple, les documents au format '*pdf*' ne possèdent pas de liens sortants. Il n'est donc pas possible de combiner une requête atomique « *outgoing_links* » avec une requête « *MIME* » sur le type '*application/pdf*'. Le tableau III.7 page 49 montre les requêtes atomiques qu'il est possible de combiner par type MIME.
4. Un document ne peut être que d'un et un seul type *MIME* à la fois. Pour cette raison, nous avons imposé l'utilisation du constructeur sémantique logique « **Exactly**(1) » avec une requête sur le type *MIME*. L'évaluation de cette requête sur un document donné renvoie 1 lorsque

le type MIME de ce document coïncide exactement avec l’une des valeurs spécifiées dans la requête. Dans le cas contraire, le score d’évaluation de ce document est nul. Nous avons, par ailleurs, imposé le poids 1 à une requête portant sur le type *MIME* dans n’importe quelle combinaison WeQueL. Le langage WeQueL n’autorise aucun changement de ce poids.

5. Pour garantir un score de combinaison compris entre 0 et 1, le langage WeQueL impose l’utilisation d’un constructeur sémantique logique pour toute requête atomique pondérée avec un poids égal à 1. Ces requêtes sont évaluées en priorité par le moteur de requête WeQueL et peuvent être considérées comme des contraintes fortes imposées aux documents à évaluer. L’échec de l’évaluation de ces contraintes sur un document donné annule systématiquement le score final de ce document sur la combinaison considérée. Il n’est alors plus nécessaire d’évaluer le reste des requêtes atomiques de la combinaison.

Dans une combinaison donnée, la distinction entre les requêtes pondérées à 1 et les autres requêtes permet à l’utilisateur d’imposer un certain nombre de contraintes qu’il juge incontournables dans la détermination de la pertinence d’un document. Par exemple, dans notre recherche thématique de pages d’accueil de tourisme, la longueur de l’url est primordiale (nécessaire mais non suffisante) dans la détermination de la pertinence des résultats. Nous avons formulé une requête atomique sur la longueur de l’url pondérée avec un poids égal à 1 dans la combinaison prévue pour cette recherche thématique. Ainsi, le score de tout document ayant une url ne correspondant pas à une url d’accueil est nul.

6. L’attribution des poids α_j pour les requêtes de la partie pondérée d’une combinaison doit respecter la condition suivante :

$$\sum_{j=1}^M \alpha_j = 1$$

type MIME	<i>page_title</i>	<i>url</i>	<i>url_length</i>	<i>keywords</i>	<i>object</i>	<i>outgoing_links</i>	<i>incoming_links</i>	<i>title_incoming_page</i>
'text/html'	✓	✓	✓	✓	✓	✓	✓	✓
'text/plain'								
'text/richtext'	×	✓	✓	✓	×	×	✓	✓
'text/x-setext'								
autres	×	✓	✓	×	×	×	✓	✓

✓ : utilisable avec le type MIME mentionné

×

TAB. III.7 – Compatibilité des requêtes atomiques WeQueL avec les types MIME de documents

Remarque 3.5 *Une requête atomique employant un constructeur sémantique numérique peut être pondérée avec un poids égal à 1. Son évaluation n’étant pas binaire, cette requête est classée par le moteur WeQueL dans la partie pondérée de la combinaison. Dans ce cas et en vertu de la condition 6, la partie pondérée ne peut contenir que cette requête atomique (pour satisfaire la condition $\sum_{\alpha_i \neq 1} \alpha_i = 1$).*

Exemple : En réutilisant les exemples 1, 2, 3 et 4 présentés à la page 47, il est possible de combiner les requêtes atomiques $\mathcal{Q}_a^{ex1}$, $\mathcal{Q}_a^{ex2}$, $\mathcal{Q}_a^{ex3}$ et $\mathcal{Q}_a^{ex4}$ pour rechercher des documents de cours java ou c++ au format html. Pour effectuer une telle recherche, nous présentons, ci-dessous, une combinaison possible à partir des requêtes atomiques précédentes :

$$\mathcal{C}_1 : 1 \cdot \left[\text{MIME} \xrightarrow{\text{CONTAINS}} \text{'text/html'}, \text{EVALUATED BY} \xrightarrow{\text{Exactly}(1)} \right] \left\langle \frac{1}{4} \cdot \mathcal{Q}_a^{ex1}, \frac{1}{4} \cdot \mathcal{Q}_a^{ex2}, \frac{1}{4} \cdot \mathcal{Q}_a^{ex3}, \frac{1}{4} \cdot \mathcal{Q}_a^{ex4} \right\rangle$$

TAB. III.8 – Exemple d’une combinaison de requêtes atomiques WeQueL

Dans cette combinaison, la partie nécessaire se compose d’une seule requête atomique qui cible les documents au format HTML : $\mathcal{Q}_{\text{MIME}} \equiv \left[\text{MIME} \xrightarrow{\text{CONTAINS}} \text{'text/html'}, \text{EVALUATED BY} \xrightarrow{\text{Exactly}(1)} \right]$. Tout document ayant un type MIME autre que `'text/html'` obtient un score d’évaluation nul pour cette combinaison sans même évaluer les autres requêtes atomiques.

La partie pondérée comporte quatre requêtes atomiques possédant le même poids d’importance ($\frac{1}{4}$ chacune) dans l’évaluation de la combinaison $\mathcal{C}_1$. Pour cette dernière combinaison, chaque requête atomique contribue à hauteur de 25% du score final d’évaluation de $\mathcal{C}_1$ sur n’importe quel document.

Remarque 3.6 *Le choix du même poids ($\frac{1}{4}$) pour $\mathcal{Q}_a^{ex1}$, $\mathcal{Q}_a^{ex2}$, $\mathcal{Q}_a^{ex3}$ et $\mathcal{Q}_a^{ex4}$ permet d’équilibrer l’influence de chacune de ces requêtes atomiques dans le calcul du score total de la combinaison $\mathcal{C}_1$ sur n’importe quel document. Avec l’attribution suivante des poids : $(\mathcal{C}'_1 \equiv 1 \cdot \mathcal{Q}_{\text{MIME}} \langle \frac{3}{4} \cdot \mathcal{Q}_a^{ex1}, \frac{1}{16} \cdot \mathcal{Q}_a^{ex2}, \frac{1}{16} \cdot \mathcal{Q}_a^{ex3}, \frac{1}{8} \cdot \mathcal{Q}_a^{ex4} \rangle)$, la requête atomique $\mathcal{Q}_a^{ex1}$ ciblant l’url des documents bénéficie de la plus grande influence, i.e. 75% du score de la combinaison $\mathcal{C}'_1$ sur n’importe quel document.*

3.3 Syntaxe d’une requête WeQueL complexe

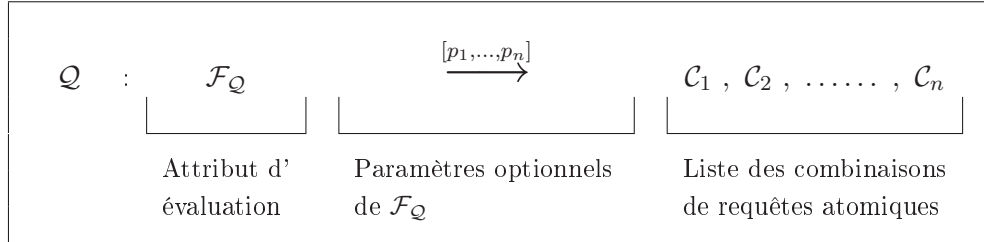
Nous avons vu qu’il était possible via une combinaison WeQueL de cibler plusieurs critères de recherche à la fois sur différentes sections de documents d’un ou plusieurs types MIME donnés. Cette factorisation de critères constitue un premier niveau de combinaison de requêtes atomiques. Au-dessus de ce niveau, le langage WeQueL offre un second niveau de combinaison où l’utilisateur peut rassembler différentes combinaisons de requêtes atomiques portant sur différents types MIME de documents. La requête ainsi formée permet, par exemple, de cibler des catégories *MIME* différentes avec plusieurs combinaisons ou de cibler une même catégorie *MIME* avec différentes combinaisons.

Une requête WeQueL complexe nécessite une fonction d’évaluation pour déterminer la méthode de calcul du score de cette requête à partir des scores de ses combinaisons. À cet effet, une requête WeQueL complexe dispose d’un attribut, appelé *attribut d’évaluation*. L’instanciation de cet attribut, dans une requête WeQueL complexe, permet de spécifier déclarativement la méthode de calcul du score final d’un document. Le langage WeQueL propose quatre fonctions d’évaluation prédéfinies pour instancier un attribut d’évaluation d’une requête complexe :

1. **min()** : renvoie le plus petit score d’évaluation d’un document donné sur les combinaisons de la requête complexe.
2. **max()** : renvoie le plus grand score d’évaluation d’un document donné sur les combinaisons de la requête complexe.
3. **average()** : renvoie la moyenne arithmétique des scores d’évaluation obtenus par un document donné sur chaque combinaison de la requête complexe.

4. **balanced_sum()** : renvoie la somme pondérée des scores obtenus par un document donné à la suite de son évaluation sur chaque combinaison de la requête complexe. Pour cette fonction, l'utilisateur doit spécifier les poids associés à chaque combinaison en les séparant par des virgules.

Une requête WeQueL complexe, notée $\mathcal{Q}$, est définie par une liste de n combinaisons différentes et une fonction d'évaluation notée $\mathcal{F}_{\mathcal{Q}}$, conformément à la syntaxe suivante :



TAB. III.9 – Syntaxe d'une requête WeQueL complexe

Nous distinguons trois parties dans une requête complexe :

1. Une partie pour déclarer la fonction d'évaluation de la requête complexe (partie $\mathcal{F}_{\mathcal{Q}}$).
2. Une partie optionnelle pour déclarer les paramètres de la fonction d'évaluation. Par exemple, la fonction « **balanced_sum()** » requiert pour son calcul de score la liste des poids de chaque combinaison de la requête $\mathcal{Q}$.
3. Une troisième partie pour spécifier les combinaisons formant la requête $\mathcal{Q}$.

Exemple : Dans l'exemple présenté à la page 50, nous avons formulé une combinaison de requêtes atomiques pour rechercher des cours c++/java au format HTML seulement. Nous souhaitons étendre cette recherche à d'autres formats tels que le postscript et le pdf. Il nous suffit, pour cela, de construire une requête WeQueL complexe que nous notons par $\mathcal{Q}$ et qui se compose de deux combinaisons : la combinaison $\mathcal{C}_1$ et une nouvelle combinaison $\mathcal{C}_2$ spécialement conçue pour rechercher des documents de cours c++/java au format PDF ou PS. Ci-dessous une possibilité de combinaison de requêtes atomiques pour effectuer la recherche de documents PDF et PS :

$$\left\{ \begin{array}{ll} \mathcal{Q}_{MIME} & : \text{MIME} \xrightarrow{\text{CONTAINS}} \text{'application/pdf', 'application/postscript'} \xrightarrow{\text{EVALUATED BY}} \text{Exactly}(1) \\ \mathcal{Q}_a^{ex1} & : \text{nous avons réutilisé la même requête atomique sur l'url que la combinaison } \mathcal{C}_1 \\ \mathcal{Q}_a^{ex5} & : \text{incoming_links} \xrightarrow{\text{CONTAINS}} \text{'programmation objet', 'cours cpp', 'cours java'} \xrightarrow{\text{EVALUATED BY}} SC5 \end{array} \right.$$

Pour ces requêtes atomiques, nous avons retenu la combinaison suivante :

$$\mathcal{C}_2 : 1 \cdot \mathcal{Q}_{MIME} \quad \left\langle \frac{2}{3} \cdot \mathcal{Q}_a^{ex1} \quad , \quad \frac{1}{3} \cdot \mathcal{Q}_a^{ex5} \right\rangle$$

Dans cette combinaison, nous avons volontairement privilégié la pertinence de l'url à celle des mots des liens entrants pour ne pas trop pénaliser les documents pour lesquels il est difficile

d'obtenir des liens entrants. L'absence des liens entrants pour un document donné n'engendre pas plus de 30% de perte dans son score total pour la combinaison $\mathcal{C}_2$.

La requête complexe $\mathcal{Q}$ que nous avons formée à partir des deux combinaisons $\mathcal{C}_1$ et $\mathcal{C}_2$ est :

$$\mathcal{Q} : \mathcal{F}_{\mathcal{Q}} \xrightarrow{[p_1, \dots, p_n]} \mathcal{C}_1, \mathcal{C}_2$$

TAB. III.10 – Exemple d'une requête WeQueL complexe

Remarque 3.7 Dans cet exemple, la fonction d'évaluation de la requête complexe $\mathcal{Q}$ n'est pas précisée.

3.4 Syntaxe opérationnelle du langage WeQueL

Afin d'éviter le développement d'un analyseur syntaxique spécifique au langage WeQueL, nous avons utilisé le langage XML et ses propriétés pour fournir, à moindre coût, une syntaxe opérationnelle conforme et équivalente à la syntaxe abstraite présentée précédemment. Nous avons défini, à cet effet, un schéma XML de description du langage WeQueL conformément aux spécifications développées dans la section 3 page 43. Ce schéma XML, disponible à l'adresse <http://www.lri.fr/~mezaour/WeQueL/WeQueLschema.xsd>, est montré dans la figure III.1 page 53.

Ainsi, une requête WeQueL complexe $\mathcal{Q}$ est définie par un document XML ayant pour racine un élément de type `WeQueLComplexQuery`. Ce type décrit les trois informations à fournir pour composer une requête atomique. La première information est obligatoire et consiste à déclarer pour l'attribut `evaluation_function` un nom de fonction d'évaluation $\mathcal{F}_{\mathcal{Q}}$ parmi les valeurs prédéfinies du type `WeQueLEvaluationFunctions`. La deuxième information, représentée par l'attribut `evaluation_function_parameters`, est optionnelle et ne concerne que les fonctions $\mathcal{F}_{\mathcal{Q}}$ nécessitant des paramètres (ex : la fonction `balanced_sum()`). La troisième information consiste à introduire chaque combinaison de la requête $\mathcal{Q}$ sous forme d'un élément de type `combination`.

Conformément aux règles de composition d'une combinaison de requêtes atomiques, le type `combination` contraint les utilisateurs du langage WeQueL à fournir au minimum deux éléments de type `atomic_query` ou au maximum neuf éléments de ce même type pour contruire une combinaison de requêtes atomiques. La borne inférieure de deux éléments correspond à une combinaison unitaire constituée obligatoirement d'une requête atomique portant sur le type MIME et d'une deuxième requête atomique portant sur une autre section. La borne maximale de neuf éléments dans une combinaison correspond aux neuf requêtes atomiques différentes qu'il est possible de combiner sous le langage WeQueL.

Le type `atomic_query`, défini dans le schéma XML de description du langage WeQueL, permet de construire une requête atomique avec cinq attributs et un contenu textuel compris entre la balise `<atomic_query>` et sa balise fermante. Ce contenu textuel correspond aux valeurs de l'ensemble $V_{\mathcal{Q}_a}$ d'une requête atomique $\mathcal{Q}_a$ (voir section 3.1 page 43). Les cinq attributs de définition d'une requête atomique $\mathcal{Q}_a$ sont :

```

<?xml version='1.0' encoding='UTF-8'?>

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.lri.fr/~mezaour/WebServices/xml-schemas"
  xmlns="http://www.lri.fr/~mezaour/WebServices/xml-schemas">
  <xsd:element name="WeQueLComplexQuery">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="combination" minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
      <xsd:attribute name="evaluation_function" type="WeQueLEvaluationFunctions" use="required"/>
      <xsd:attribute name="evaluation_function_parameters" type="xsd:string" use="optional"/>
    </xsd:complexType>
  </xsd:element>
  <xsd:simpleType name="WeQueLEvaluationFunctions">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="max"/>
      <xsd:enumeration value="min"/>
      <xsd:enumeration value="average"/>
      <xsd:enumeration value="balanced_sum"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:element name="combination">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="atomic_query" minOccurs="2" maxOccurs="9"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="atomic_query">
    <xsd:complexType>
      <xsd:simpleContent>
        <xsd:extension base="xsd:string">
          <xsd:attribute name="name" type="WeQueLAtomicQueries" use="required"/>
          <xsd:attribute name="type" type="ObjectType" use="optional"/>
          <xsd:attribute name="weight" type="xsd:decimal" use="required"/>
          <xsd:attribute name="semantic_builder" type="WeQueLSemanticBuilders" use="required"/>
          <xsd:attribute name="semantic_builder_parameters" type="xsd:string" use="optional"/>
        </xsd:extension>
      </xsd:simpleContent>
    </xsd:complexType>
  </xsd:element>
  <xsd:simpleType name="WeQueLSemanticBuilders">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="conjunct"/>
      <xsd:enumeration value="disjunct"/>
      <xsd:enumeration value="atleast"/>
      <xsd:enumeration value="atmost"/>
      <xsd:enumeration value="exactly"/>
      <xsd:enumeration value="precision"/>
      <xsd:enumeration value="recall"/>
      <xsd:enumeration value="Fscore"/>
      <xsd:enumeration value="cosine"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:simpleType name="WeQueLAtomicQueries">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="title"/>
      <xsd:enumeration value="url"/>
      <xsd:enumeration value="incoming_links"/>
      <xsd:enumeration value="outgoing_links"/>
      <xsd:enumeration value="titles_incoming_pages"/>
      <xsd:enumeration value="object"/>
      <xsd:enumeration value="keywords"/>
      <xsd:enumeration value="url_length"/>
      <xsd:enumeration value="mime"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:simpleType name="ObjectType">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="tables"/>
      <xsd:enumeration value="lists"/>
      <xsd:enumeration value="tables_lists"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:schema>

```

FIG. III.1 – schéma XML associé à une requête WeQueL

1. **semantic_builder** : désigne le constructeur sémantique à appliquer sur $\mathcal{Q}_a$ lors de son évaluation. Cet attribut doit obligatoirement être renseigné.
2. **semantic_builder_parameters** : contient l'énumération des paramètres du constructeur sémantique choisi. Certains constructeurs ne nécessitant pas de paramètres, cet attribut est optionnel.
3. **name** : désigne le nom de la section ciblée par la requête atomique. Ce nom est obligatoire et prend ses valeurs parmi les neuf valeurs prédéfinies par le type **WeQueLAtomicQueries**.
4. **type** : cet attribut n'accompagne que les requêtes atomiques portant sur les objets d'un document.

5. **weight** : détermine la pondération associée à la requête atomique lors du calcul du score de la combinaison. Cet attribut est obligatoire.

Pour résumer, la conversion de la syntaxe abstraite d'une requête WeQueL complexe Q vers une requête en syntaxe XML s'effectue comme suit :

$$Q \equiv \mathcal{F}_Q \xrightarrow{[p_1, \dots, p_n]} C_1, C_2, \dots, C_n$$

Où : $\forall i \in \{1, \dots, n\}$ $C_i \equiv 1 \cdot Q_{MIME}^i \cdot Q_a^{1i} \cdot Q_a^{2i} \dots Q_a^{(N-1)i} \cdot Q_a^{Ni} \langle \alpha_1^i \cdot \widehat{Q_a^{1i}}, \dots, \alpha_M^i \cdot \widehat{Q_a^{Mi}} \rangle$



```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<WeQueLComplexQuery xmlns="http://www.lri.fr/~mezaour/WebServices/xml-schemas"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.lri.fr/~mezaour/WebServices/xml-schemas
  http://www.lri.fr/~mezaour/WeQueL/WeQueLschema.xsd"
  evaluation_function="F_Q"
  evaluation_function_parameters="p1,...,pn">

  <combination>
  <atomic_query name="MIME" weight="1"
    semantic_builder="Exactly" semantic_builder_parameters="1">
    MIME1,MIME2,.....,MIMEX
  </atomic_query>
  <atomic_query name="Q_a^{11}" weight="1"
    semantic_builder="SC_{11}" semantic_builder_parameters="...">
    value1,value2,.....,valueX
  </atomic_query>
    :
  <atomic_query name="Q_a^{1i}" weight="alpha_1^i"
    semantic_builder="SC_{1i}" semantic_builder_parameters="...">
    value1,value2,.....,valueX
  </atomic_query>
    :
  </combination>
  <combination>
    :
  </combination>
    :
  </WeQueLComplexQuery>
```

TAB. III.11 – Conversion de la syntaxe abstraite d'une requête WeQueL en syntaxe XML

Exemple : La figure ci-dessous présente le document XML correspondant à la requête Q de l'exemple de la page 51.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<WeQueLComplexQuery xmlns="http://www.lri.fr/~mezaour/WebServices/xml-schemas"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.lri.fr/~mezaour/WebServices/xml-schemas
    http://www.lri.fr/~mezaour/WeQueL/WeQueLSchema.xsd"
  evaluation_function="max">
  <combination>
    <atomic_query name="mime" weight="1" semantic_builder="exactly" semantic_builder_parameters="1">
      text/html
    </atomic_query>
    <atomic_query name="title" weight="0.25" semantic_builder="Fscore">
      support cours,java,cpp
    </atomic_query>
    <atomic_query name="url" weight="0.25" semantic_builder="Fscore">
      cours java,cours cpp
    </atomic_query>
    <atomic_query name="keywords" weight="0.25" semantic_builder="Fscore">
      cours,java,cpp,tutoriel
    </atomic_query>
    <atomic_query name="incoming_links" weight="0.25" semantic_builder="Fscore">
      cours java,cours cpp
    </atomic_query>
  </combination>
  <combination>
    <atomic_query name="mime" weight="1" semantic_builder="exactly" semantic_builder_parameters="1">
      application/pdf,application/postscript
    </atomic_query>
    <atomic_query name="url" weight="0.66" semantic_builder="Fscore">
      cours java,cours cpp
    </atomic_query>
    <atomic_query name="incoming_links" weight="0.34" semantic_builder="Fscore">
      programmation objet,cours cpp,cours java
    </atomic_query>
  </combination>
</WeQueLComplexQuery>
```

FIG. III.2 – schéma XML associé à la requête WeQueL Q de l'exemple de la page 51

4 Évaluation et sémantique des requêtes du langage WeQueL

Nous avons développé en java un moteur spécifique pour interpréter et évaluer les requêtes du langage WeQueL. Ce module logiciel, appelé **WeQueLQueryEngine**, prend en entrée un document du web et une requête WeQueL sous forme d'un fichier XML bien formé respectant le schéma XSD montré dans la figure III.1 page 53. **WeQueLQueryEngine** fournit en sortie un score d'évaluation.

Dans un souci d'efficacité dans les opérations d'évaluation, nous avons développé *WDM* : *WeQueL Document Model*. *WDM* est un modèle de représentation des documents du web spécialement conçu pour fournir les informations nécessaires à l'évaluation des requêtes du langage WeQueL par accès direct.

4.1 WDM : modèle de représentation des documents du web pour le langage WeQueL

Le modèle *WDM* représente un document du web sous forme d'une table contenant neuf entrées. Ces neuf entrées correspondent aux neuf sections qu'il est possible de cibler par les requêtes atomiques du langage WeQueL.

Chaque entrée de la table du modèle *WDM* est indexée par un nom d'attribut A de l'ensemble $\mathcal{WA}^1$ rendant immédiat l'accès au contenu textuel des différentes sections d'un document. Le contenu d'une entrée indexée par l'attribut $A \in \mathcal{WA}$ est organisé sous forme d'une chaîne de caractères qui correspond aux mots formant la section du document visée par A . Précisons que cette chaîne peut être vide dans le cas où le document ne dispose pas de la section correspondant à l'attribut A . Par exemple, un document au format *PDF* a une chaîne de caractères vide dans son entrée *outgoing_links*. La définition du contenu de chaque section d'un document est donnée dans la section 3.1.1 page 44.

Remarque 4.1 *Un document HTML peut contenir à la fois plusieurs balises définissant une même section. Dans ce cas, le contenu de l'entrée de la table WDM associé à cette section correspond à la fusion du contenu textuel de chaque balise de la section en question. Par exemple, dans le code HTML d'un document donné d , la balise `<a>`, définissant un lien hypertexte, apparaît plusieurs fois pour pointer sur différents documents. L'entrée *outgoing_link* de la table WDM contient une chaîne de caractères issue de la fusion des contenus textuels de chaque balise `<a>` du document d . Rappelons que, dans le cadre du langage WeQueL, le contenu textuel d'une balise `<a>` est défini comme étant la chaîne de caractères formée par les k mots précédant la balise `<a>`, de la décomposition de l'url référencée, de l'ancre de la balise `<a>` et des k mots suivant cette balise.*

4.1.1 Cas particuliers

L'entrée *object* : le contenu de l'entrée *object* de la table *WDM* est obtenu par la fusion des chaînes de caractères correspondant aux différents types d'objets HTML qu'il est possible de cibler par une requête atomique *object* (tables, listes, images, ...). Avant d'être fusionnée, chaque chaîne de caractère non vide est préfixée par le type de l'objet qu'elle représente. Par exemple, si « **bleu noir vert** » est la chaîne de caractères extraite d'une liste d'un document HTML, sa fusion dans le modèle *WDM* correspond à « **list :bleu noir vert** ».

L'entrée *url_length* : La longueur d'une url est définie comme étant le nombre de niveaux hiérarchiques dans l'arborescence d'accès au document désigné par une url. Sachant que le caractère « / » permet de séparer les différents niveaux d'une url, nous avons défini le contenu de l'entrée *url_length* d'une table *WDM* comme étant le nombre de caractères « / » présents dans l'url du document à représenter. Signalons que les 2 occurrences du caractère « / » qui suivent le protocole d'une url ne sont pas comptabilisées dans le précédant calcul.

Remarque 4.2 *Pour optimiser l'espace mémoire occupé par une table WDM, nous supprimons du contenu de chaque entrée de la table les caractères et les mots inutiles, c'est-à-dire : toute ponctuation, les chiffres, les mots communs (articles, prépositions, ...), les noms de protocoles (http, mail, ...), les extensions de fichiers (.html, .ps, ...), les noms de domaines (www, .com, ...), les caractères invisibles successifs.*

¹ensemble de tous les attributs possibles du langage WeQueL

Exemple : La figure III.3 page 57 représente la transformation d'un document HTML en une table *WDM*. L'url du document est : `http://www.lri.fr/~mezaour/exemple.html`. Le document ne comporte aucun document qui pointe sur lui. Par conséquent, les descriptions textuelles des sections *incoming_links* et *title_incoming_page* ont été représentées par une chaîne vide (le symbole $\emptyset$ dans la table *WDM*).

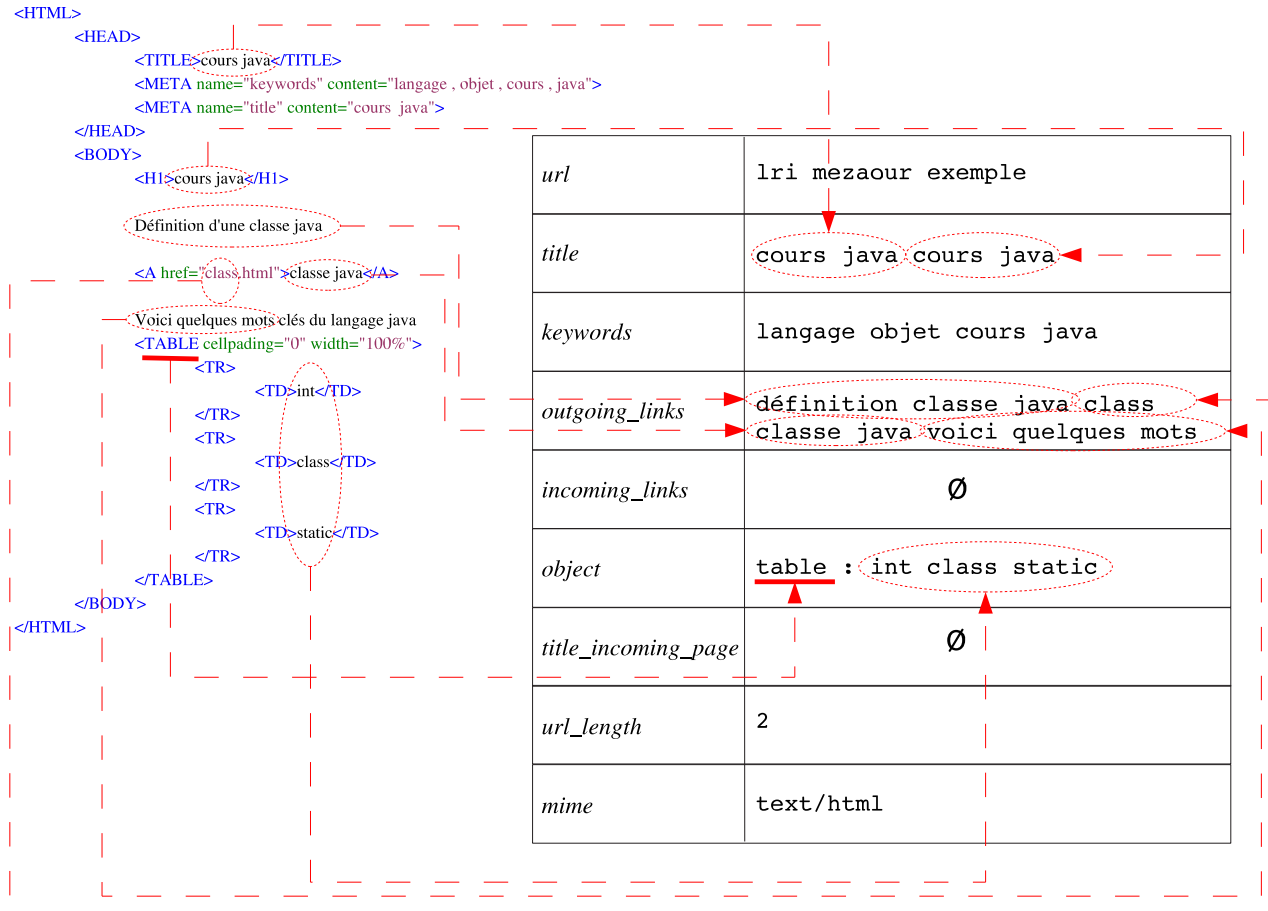


FIG. III.3 – Exemple d'un document HTML transformé en une table du modèle *WDM*

Notations : Nous notons la table *WDM* associée à un document par $\mathcal{T}_d$. Le contenu textuel " word₁ word₂ ... word_n " d'une entrée $A \in \mathcal{WA}$ de la table $\mathcal{T}_d$ est référencé par $\mathcal{T}_d[A]$.

4.2 Évaluation d'une requête atomique

WeQueLQueryEngine évalue une requête atomique du langage WeQueL sur un document du web en deux phases : phase d'appariement des valeurs de la requête $\mathcal{Q}_a$ sur la section S du document d , phase du calcul du score.

4.2.1 Appariement des valeurs d'une requête atomique sur une section d'un document du web

Cette phase débute par la transformation des valeurs de la requête atomique à évaluer en une série d'expressions régulières conformes à la syntaxe des expressions régulières définies en perl/java/unix². Cette transformation consiste à remplacer les caractères blancs, séparant les mots d'une valeur donnée, par le quantifieur universel « `.*` ». L'objectif de cette opération est de générer les motifs qui seront appliqués sur le document à évaluer pour vérifier, dans un premier temps, sa conformité avec les différentes valeurs données dans la requête atomique. Un document est dit conforme à une valeur si et seulement s'il contient tous les mots de cette valeur dans le même ordre de précedence que celui qui est défini dans la valeur de la requête atomique considérée.

Soient :

- $Q_a : S \xrightarrow{\text{CONTAINS}} \text{value}_1, \dots, \text{value}_N \xrightarrow{\text{EVALUATED BY}} SC_S$ une requête atomique du langage WeQueL où les valeurs « $\text{value}_{i,i=1..N}$ » sont des chaînes de caractères représentées comme suit :

$$\boxed{\text{value}_{i,i=1..N} \equiv \text{" word}_1^i \text{ word}_2^i \dots \text{word}_k^i \text{"}}$$

- d un document du web représenté par une table WDM notée $\mathcal{T}_d$ dans laquelle l'entrée S correspondant à l'attribut de la requête atomique Q_a possède le contenu suivant :

$$\mathcal{T}_d[S] = \text{" word}_1 \text{ word}_2 \dots \text{word}_n \text{"}$$

L'appariement des valeurs de la requête atomique Q_a sur la section S du document d revient à vérifier la conformité du contenu textuel de la section S avec chaque valeur de la requête atomique Q_a . Pour cela, le moteur d'interprétation des requêtes du langage WeQueL génère un ensemble d'expressions régulières à partir de chaque valeur $\text{value}_{i,i=1..N}$ de la requête Q_a . Cet ensemble, noté $REGEX$, est défini comme suit :

$$REGEX = \{regex_{i,i=1..N} / (regex_i = \text{" word}_1^i \text{.*word}_2^i \text{.*} \dots \text{.*word}_k^i \text{"})\}$$

Chaque expression régulière $regex_i \in REGEX$ est ensuite appariée sur le contenu textuel $\mathcal{T}_d[S]$ de la section S du document d . Nous avons utilisé à cet effet l'API `Regex` fournie par java². En sortie de ce processus, `WeQueLQueryEngine` produit deux ensembles pour résumer les succès et les échecs des opérations d'appariement.

Le premier ensemble, noté $\mathcal{MV}_{(Q_a,d)}$ (*Matched Values*), regroupe toutes les valeurs « $\text{value}_{i,i=1..N}$ » de la requête Q_a pour lesquelles l'appariement de l'expression régulière associée « $regex_i$ » a réussi sur $\mathcal{T}_d$.

Le second ensemble, noté $\mathcal{MW}_{(Q_a,d)}$ (*Matched Words*), contient tous les mots appartenant à la fois à $\mathcal{T}_d$ et à une valeur « $\text{value}_{i,i=1..N}$ » de l'ensemble $\mathcal{MV}_{(Q_a,d)}$. Cet ensemble sert principalement à regrouper les mots de la section S du document d qui sont « pertinents », c'est-à-dire mentionnés dans l'une des valeurs de la requête Q_a .

²voir <http://java.sun.com/j2se/1.4.2/docs/api/java/util/regex/Pattern.html>

4.2.2 Calcul du score d'une requête atomique

Le calcul du score d'un document évalué sur une requête atomique appariée dépend de deux paramètres : le constructeur sémantique spécifié dans cette requête et les ensembles $\mathcal{MV}_{(\mathcal{Q}_a, d)}$ et $\mathcal{MW}_{(\mathcal{Q}_a, d)}$ générés à la suite de la phase d'appariement. Le tableau III.12 page 59 présente le mode de calcul du score d'un document d évalué sur une requête atomique $\mathcal{Q}_a$.

Constructeurs logiques	
$SC = \text{Disjunct}()$	$score(\mathcal{Q}_a, d) = \begin{cases} 1 & \text{si } \mathcal{MV}_{(\mathcal{Q}_a, d)} \neq 0 \\ 0 & \text{sinon} \end{cases}$
$SC = \text{Conjunct}()$	$score(\mathcal{Q}_a, d) = \begin{cases} 1 & \text{si } \mathcal{MV}_{(\mathcal{Q}_a, d)} = V_S \\ 0 & \text{sinon} \end{cases}$
$SC = \text{Atleast}(k)$	$score(\mathcal{Q}_a, d) = \begin{cases} 1 & \text{si } \mathcal{MV}_{(\mathcal{Q}_a, d)} \geq k \\ 0 & \text{sinon} \end{cases}$
$SC = \text{Atmost}(k)$	$score(\mathcal{Q}_a, d) = \begin{cases} 1 & \text{si } \mathcal{MV}_{(\mathcal{Q}_a, d)} \leq k \\ 0 & \text{sinon} \end{cases}$
$SC = \text{Exactly}(k)$	$score(\mathcal{Q}_a, d) = \begin{cases} 1 & \text{si } \mathcal{MV}_{(\mathcal{Q}_a, d)} = k \\ 0 & \text{sinon} \end{cases}$
Constructeurs numériques	
$SC = \text{Precision}()$	$score(\mathcal{Q}_a, d) = P = \frac{ \mathcal{MW}_{(\mathcal{Q}_a, d)} }{ \mathcal{T}_d[S] }$
$SC = \text{Recall}()$	$score(\mathcal{Q}_a, d) = R = \frac{ \mathcal{MV}_{(\mathcal{Q}_a, d)} }{ V_S }$
$SC = \text{Fscore}()$	$score(\mathcal{Q}_a, d) = \frac{(\beta^2+1) \times P \times R}{\beta^2 \times P + R}$

TAB. III.12 – Calcul du score d'une requête atomique en fonction du constructeur sémantique employé

Remarque 4.3 Dans notre implantation du langage *WeQueL*, nous avons utilisé ($\beta = 1$) pour l'évaluation des requêtes atomiques muni du constructeur sémantique **Fscore()** car il donne le même poids aux deux mesures de précision et de rappel que nous avons défini.

Cas particulier : Pour le constructeur **Cosine()**, **WeQueLQueryEngine** calcule le cosinus de l'angle compris entre le vecteur des occurrences des mots contenus dans les valeurs de la requête $\mathcal{Q}_a$ et le vecteur des occurrences des mots de la section $\mathcal{T}_d[S]$ du document à évaluer. Ce calcul ne dépend pas de la phase d'appariement.

Soient :

- $\mathcal{V}$ le vocabulaire constitué des mots de la section $\mathcal{T}_d[S]$ et des mots des valeurs « **value** _{$i, i=1..N$} » de la requête atomique :

$$\mathcal{V} = \{\text{word}_i / (\text{word}_i \in \mathcal{T}_d[S]) \vee (\exists \text{valeur}_i \in V_S : \text{word}_i \in \text{valeur}_i)\}$$

- V le nombre de mots du vocabulaire $\mathcal{V}$;
- v_i^d le nombre d'occurrences du mot « $\text{word}_i \in \mathcal{V}$ » dans la section $\mathcal{T}_d[S]$;
- $v_i^{\mathcal{Q}_a}$ le nombre d'occurrences du mot « $\text{word}_i \in \mathcal{V}$ » dans la fusion de toutes les valeurs de l'ensemble V_S ;
- $\vec{V}_d = \langle v_1^d, \dots, v_V^d \rangle$ le vecteur représentatif de la section $\mathcal{T}_d[S]$;
- $\vec{V}_{\mathcal{Q}_a} = \langle v_1^{\mathcal{Q}_a}, \dots, v_V^{\mathcal{Q}_a} \rangle$ le vecteur représentatif des valeurs de la requête atomique $\mathcal{Q}_a$.

Le calcul du cosinus de l'angle qui sépare le vecteur V_d du vecteur $V_{\mathcal{Q}_a}$ s'appuie sur la formule suivante :

$$\cos(\vec{V}_d, \vec{V}_{\mathcal{Q}_a}) = \frac{\vec{V}_d \cdot \vec{V}_{\mathcal{Q}_a}}{\|\vec{V}_d\| \cdot \|\vec{V}_{\mathcal{Q}_a}\|}$$

En utilisant la formule usuelle du calcul du produit cartésien de deux vecteurs et la norme euclidienne pour le calcul de la norme de chaque vecteur, la formule précédente s'écrit :

$$\cos(\vec{V}_d, \vec{V}_{\mathcal{Q}_a}) = \frac{\sum_{i=1}^V v_i^d \cdot v_i^{\mathcal{Q}_a}}{\sqrt{\sum_{i=1}^V (v_i^d)^2} \cdot \sqrt{\sum_{i=1}^V (v_i^{\mathcal{Q}_a})^2}}$$

4.2.3 Exemples d'évaluations de requêtes atomiques

Soient $\mathcal{D} = \{d_1, d_2, d_3, d_4, d_5\}$ un ensemble de cinq documents sur lequel nous souhaitons évaluer les requêtes atomiques présentées à la page 47. Les urls correspondant à ces documents sont respectivement :

- d_1 : <http://www.infres.enst.fr/~charon/coursJava/> ;
- d_2 : <http://www.liafa.jussieu.fr/~hf/verif/ens/cours/java/cours01.ps> ;
- d_3 : <http://www-adele.imag.fr/~donsez/cours/> ;
- d_4 : <http://java.developpez.com/cours/> ;
- d_5 : <http://www.iut-bethune.univ-artois.fr/~caron/courscpp/courscpp.html>.

Les tables *WDM* associées à ces documents sont données ci-dessous :

Remarque 4.4 *En raison du fait que les requêtes des exemples de la page 47 ne ciblent ni les liens sortants ni les titres entrants ni les objets d'un document, nous avons volontairement omis de préciser les contenus des entrées `outgoing_links`, `title_incoming_pages` et `objects` dans les tables *WDM* ci-dessous.*

<i>url</i>	infres enst charon coursjava
<i>title</i>	apprendre java
<i>keywords</i>	∅
<i>outgoing_links</i>	/
<i>incoming_links</i>	apprendre java cours langage
<i>object</i>	/
<i>title_incoming_page</i>	/
<i>url_length</i>	3
<i>MIME</i>	text/html

TAB. III.13 – Table *WDM* associée au document d_1

<i>url</i>	liafa jussieu hf verif ens cours java cours
<i>title</i>	∅
<i>keywords</i>	∅
<i>outgoing_links</i>	∅
<i>incoming_links</i>	liaison dynamique coeur programmation orientée objet
<i>object</i>	∅
<i>title_incoming_page</i>	/
<i>url_length</i>	6
<i>MIME</i>	application/postscript

TAB. III.14 – Table *WDM* associée au document d_2

<i>url</i>	adele imag donsez cours
<i>title</i>	didier donsez transparents cours didier donsez transparents cours
<i>keywords</i>	∅
<i>outgoing_links</i>	/
<i>incoming_links</i>	didier donsez université grenoble adele imag donsez cours
<i>object</i>	/
<i>title_incoming_page</i>	/
<i>url_length</i>	3
<i>MIME</i>	text/html

TAB. III.15 – Table *WDM* associée au document d_3

<i>url</i>	java developpez cours
<i>title</i>	cours tutoriels java club entraide développeurs francophones
<i>keywords</i>	java javabeans ejb applets jdk jdbc download trucs programmation cours cour francophone
<i>outgoing_links</i>	/
<i>incoming_links</i>	besoin aide java java developpez cours
<i>object</i>	/
<i>title_incoming_page</i>	/
<i>url_length</i>	2
<i>MIME</i>	text/html

TAB. III.16 – Table *WDM* associée au document d_4

<i>url</i>	iut bethune univ artois caron courscpp courscpp
<i>title</i>	support cours c c
<i>keywords</i>	$\emptyset$
<i>outgoing_links</i>	/
<i>incoming_links</i>	cours programmation patrick trau iut bethune univ artois caron courscpp courscpp support cours c c olivier caron
<i>object</i>	/
<i>title_incoming_page</i>	/
<i>url_length</i>	3
<i>MIME</i>	text/html

TAB. III.17 – Table *WDM* associée au document d_5

Ci-dessous, les différents ensembles *REGEX* générés par *WeQueLQueryEngine* à partir des valeurs des requêtes Q_a^{ex1} , Q_a^{ex2} , Q_a^{ex3} et Q_a^{ex4} .

Requête atomique	<i>REGEX</i> généré
Q_a^{ex1}	{cours.*java , cours.*cpp}
Q_a^{ex2}	{support.*cours , java , cpp}
Q_a^{ex3}	{cours , java , cpp , tutoriel}
Q_a^{ex4}	{cours.*java , cours.*cpp}

TAB. III.18 – Ensembles *REGEX* générés à partir des valeurs des requêtes atomiques de la page 47

Nous reportons, dans les deux tableaux ci-dessous, les contenus des ensembles $\mathcal{MV}$ et $\mathcal{MW}$ générés suite à l'évaluation de chaque expression régulière des ensembles $REGEX$ précédents sur les sections correspondantes des documents de l'ensemble $\mathcal{D}$.

	Q_a^{ex1}	Q_a^{ex2}	Q_a^{ex3}	Q_a^{ex4}
d_1	{cours java}	{java}	$\emptyset$	$\emptyset$
d_2	{cours java}	$\emptyset$	$\emptyset$	$\emptyset$
d_3	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
d_4	$\emptyset$	{java}	{cours , java}	$\emptyset$
d_5	{cours cpp}	{support cours}	$\emptyset$	{cours cpp}

TAB. III.19 – Ensembles $\mathcal{MV}$ générés à la suite de l'évaluation des requêtes atomiques de la page 47 sur l'ensemble $\mathcal{D}$

	Q_a^{ex1}	Q_a^{ex2}	Q_a^{ex3}	Q_a^{ex4}
d_1	{coursjava}	{java}	$\emptyset$	$\emptyset$
d_2	{cours, java, cours}	$\emptyset$	$\emptyset$	$\emptyset$
d_3	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
d_4	$\emptyset$	{java}	{java, cours}	$\emptyset$
d_5	{courscpp, courscpp}	{support cours}	$\emptyset$	{cours, courscpp, courscpp, cours}

TAB. III.20 – Ensembles $\mathcal{MW}$ générés à la suite de l'évaluation des requêtes atomiques de la page 47 sur l'ensemble $\mathcal{D}$

Les scores d'évaluation des cinq documents de l'ensemble $\mathcal{D}$ sur les requêtes atomiques Q_a^{ex1} , Q_a^{ex2} , Q_a^{ex3} , Q_a^{ex4} en fonction du constructeur sémantique SC sont présentés dans les tableaux ci-dessous :

	Q_a^{ex1}							
	Constructeurs logiques					Constructeurs numériques		
	Disjunct()	Conjunct()	Atleast(1)	Atmost(1)	Exactly(1)	precision()	recall()	Fscore()
d_1	1	0	0	1	0	$\frac{1}{4}$	$\frac{1}{2}$	$\frac{1}{3}$
d_2	1	0	0	1	0	$\frac{1}{3}$	$\frac{1}{2}$	$\frac{1}{4}$
d_3	0	0	0	1	0	0	0	∞
d_4	0	0	0	1	0	0	0	∞
d_5	1	0	0	1	0	$\frac{2}{7}$	$\frac{1}{2}$	$\frac{2}{11}$

TAB. III.21 – Scores d'évaluation des documents de l'ensemble $\mathcal{D}$ sur la requête atomique Q_a^{ex1}

	Q_a^{ex2}							
	Constructeurs logiques					Constructeurs numériques		
	Disjunct()	Conjunct()	Atleast(1)	Atmost(1)	Exactly(1)	precision()	recall()	Fscore()
d_1	1	0	1	1	1	$\frac{1}{2}$	$\frac{1}{3}$	$\frac{1}{5}$
d_2	0	0	0	1	0	0	0	∞
d_3	0	0	0	1	0	0	0	∞
d_4	1	0	1	1	1	$\frac{1}{2}$	$\frac{1}{3}$	$\frac{1}{10}$
d_5	1	0	1	1	1	$\frac{2}{4}$	$\frac{3}{3}$	$\frac{5}{5}$

TAB. III.22 – Scores d'évaluation des documents de l'ensemble $\mathcal{D}$ sur la requête atomique Q_a^{ex2}

	Q_a^{ex3}							
	Constructeurs logiques					Constructeurs numériques		
	Disjunct()	Conjunct()	Atleast(1)	Atmost(1)	Exactly(1)	precision()	recall()	Fscore()
d_1	0	0	0	1	0	0	0	∞
d_2	0	0	0	1	0	0	0	∞
d_3	0	0	0	1	0	0	0	∞
d_4	1	0	1	0	0	$\frac{2}{12}$	$\frac{1}{2}$	$\frac{1}{8}$
d_5	0	0	0	1	0	0	0	∞

TAB. III.23 – Scores d'évaluation des documents de l'ensemble $\mathcal{D}$ sur la requête atomique Q_a^{ex3}

	Q_a^{ex4}							
	Constructeurs logiques					Constructeurs numériques		
	Disjunct()	Conjunct()	Atleast(1)	Atmost(1)	Exactly(1)	precision()	recall()	Fscore()
d_1	0	0	0	1	0	0	0	∞
d_2	0	0	0	1	0	0	0	∞
d_3	0	0	0	1	0	0	0	∞
d_4	0	0	0	1	0	0	0	∞
d_5	1	0	1	1	1	$\frac{4}{17}$	$\frac{1}{2}$	$\frac{4}{25}$

TAB. III.24 – Scores d'évaluation des documents de l'ensemble $\mathcal{D}$ sur la requête atomique Q_a^{ex4}

4.3 Évaluation d'une combinaison de requêtes atomiques

Soit $\mathcal{C}$ une combinaison, dans une requête complexe $\mathcal{Q}$, contenant N requêtes atomiques dans sa partie nécessaire et M requêtes atomiques dans sa partie pondérée, *i.e.* :

$$\mathcal{C} \equiv 1 \cdot \mathcal{Q}_{MIME} \cdot \mathcal{Q}_a^1 \cdot \mathcal{Q}_a^2 \dots \mathcal{Q}_a^N \langle \alpha_1 \cdot \widehat{\mathcal{Q}}_a^1, \dots, \alpha_M \cdot \widehat{\mathcal{Q}}_a^M \rangle$$

Une fois le score de chaque requête atomique de la combinaison $\mathcal{C}$ déterminé, le calcul du score de la combinaison $\mathcal{C}$ s'effectue comme suit :

$$score(d, \mathcal{C}) = score(d, \mathcal{Q}_{MIME}) \times \prod_{i=1}^N score(d, \mathcal{Q}_a^i) \sum_{j=1}^M \alpha_j \cdot score(d, \widehat{\mathcal{Q}}_a^j)$$

Rappelons que les requêtes $\mathcal{Q}_{MIME}$ et $\mathcal{Q}_a^{i=1..N}$ représentent des conditions nécessaires que doivent satisfaire les documents pour être évalués sur le reste des requêtes atomiques de la partie pondérée. Si l'une des requêtes de la partie nécessaire est insatisfaite, le produit montré dans la formule précédente annule le score global du document sur la combinaison.

Exemple : Soit $\mathcal{C}_1$ la combinaison de requêtes atomiques présentée dans l'exemple à la page 50. Le score d'évaluation d'un document d quelconque sur cette combinaison s'exprime comme suit :

$$score(\mathcal{C}_1, d) = score(d, \mathcal{Q}_{MIME}) \times \left[\frac{1}{4} \cdot score(\mathcal{Q}_a^{ex1}) + \frac{1}{4} \cdot score(\mathcal{Q}_a^{ex2}) + \frac{1}{4} \cdot score(\mathcal{Q}_a^{ex3}) + \frac{1}{4} \cdot score(\mathcal{Q}_a^{ex4}) \right]$$

Le tableau ci-dessous résume ce calcul sur les cinq documents de l'ensemble $\mathcal{D}$.

		Q_a^{ex1}					
		Conjunct()	Disjunct()	Precision()	Recall()		
Q_a^{ex4}	Atleast(1)	$score(C_1, d_1) = 0$	$score(C_1, d_1) = 0.25$	$score(C_1, d_1) = 0.06$	$score(C_1, d_1) = 0.12$	Conjunct()	Q_a^{ex2}
		$score(C_1, d_2) = 0$	$score(C_1, d_2) = 0$	$score(C_1, d_2) = 0$	$score(C_1, d_2) = 0$		
		$score(C_1, d_3) = 0$	$score(C_1, d_3) = 0$	$score(C_1, d_3) = 0$	$score(C_1, d_3) = 0$		
		$score(C_1, d_4) = 0.25$	$score(C_1, d_4) = 0.25$	$score(C_1, d_4) = 0.15$	$score(C_1, d_4) = 0$		
		$score(C_1, d_5) = 0.25$	$score(C_1, d_5) = 0.5$	$score(C_1, d_5) = 0.32$	$score(C_1, d_5) = 0.49$		
	Precision()	$score(C_1, d_1) = 0.25$	$score(C_1, d_1) = 0.5$	$score(C_1, d_1) = 0.31$	$score(C_1, d_1) = 0.37$	Disjunct()	
		$score(C_1, d_2) = 0$	$score(C_1, d_2) = 0$	$score(C_1, d_2) = 0$	$score(C_1, d_2) = 0$		
		$score(C_1, d_3) = 0$	$score(C_1, d_3) = 0$	$score(C_1, d_3) = 0$	$score(C_1, d_3) = 0$		
		$score(C_1, d_4) = 0.5$	$score(C_1, d_4) = 0.33$	$score(C_1, d_4) = 0.4$	$score(C_1, d_4) = 0.25$		
		$score(C_1, d_5) = 0.33$	$score(C_1, d_5) = 0.58$	$score(C_1, d_5) = 0.4$	$score(C_1, d_5) = 0.57$		
	Recall()	$score(C_1, d_1) = 0.25$	$score(C_1, d_1) = 0.5$	$score(C_1, d_1) = 0.31$	$score(C_1, d_1) = 0.37$	Atleast(1)	
		$score(C_1, d_2) = 0$	$score(C_1, d_2) = 0$	$score(C_1, d_2) = 0$	$score(C_1, d_2) = 0$		
		$score(C_1, d_3) = 0$	$score(C_1, d_3) = 0$	$score(C_1, d_3) = 0$	$score(C_1, d_3) = 0$		
		$score(C_1, d_4) = 0.5$	$score(C_1, d_4) = 0.33$	$score(C_1, d_4) = 0.4$	$score(C_1, d_4) = 0.25$		
		$score(C_1, d_5) = 0.41$	$score(C_1, d_5) = 0.66$	$score(C_1, d_5) = 0.48$	$score(C_1, d_5) = 0.65$		
	Fscore()	$score(C_1, d_1) = 0.07$	$score(C_1, d_1) = 0.32$	$score(C_1, d_1) = 0.14$	$score(C_1, d_1) = 0.2$	Cosine()	
		$score(C_1, d_2) = 0$	$score(C_1, d_2) = 0$	$score(C_1, d_2) = 0$	$score(C_1, d_2) = 0$		
		$score(C_1, d_3) = 0.06$	$score(C_1, d_3) = 0.06$	$score(C_1, d_3) = 0.06$	$score(C_1, d_3) = 0.06$		
		$score(C_1, d_4) = 0.05$	$score(C_1, d_4) = 0.13$	$score(C_1, d_4) = 0.2$	$score(C_1, d_4) = 0.05$		
		$score(C_1, d_5) = 0.15$	$score(C_1, d_5) = 0.4$	$score(C_1, d_5) = 0.22$	$score(C_1, d_5) = 0.39$		
		Atleast(1)	Precision()	Recall()	Fscore()		
Q_a^{ex3}							

TAB. III.25 – Évaluation d'une combinaison de requêtes atomiques sur un exemple de documents

Le document d_2 étant le seul à ne pas être au format HTML, son évaluation échoue sur la requête $\mathcal{Q}_{MIME}$. Cet échec implique un score nul pour toute la combinaison quel que soit le score des autres requêtes atomiques.

4.4 Évaluation d'une requête WeQueL complexe

Soit $\mathcal{Q}$ une requête WeQueL complexe composée de n combinaisons de requêtes atomiques :

$$\mathcal{Q} \equiv \xrightarrow{\mathcal{F}_{\mathcal{Q}}^{[p_1, \dots, p_n]}} \mathcal{C}_1, \mathcal{C}_2, \dots, \mathcal{C}_n$$

Le score d'évaluation de la requête $\mathcal{Q}$ sur un document web d est calculé comme suit :

$$score(d, \mathcal{Q}) = \mathcal{F}_{\mathcal{Q}}^{[p_1, \dots, p_n]}(score(d, \mathcal{C}_1), \dots, score(d, \mathcal{C}_n))$$

Pour les quatre fonctions d'évaluations définies pour les requêtes complexes du langage WeQueL, le calcul précédent correspond aux formules suivantes :

Fonction d'évaluation	paramètres	calcul du score
<code>min()</code>	$\emptyset$	$score(d, \mathcal{Q}) = \min_{i=1..n} [score(d, \mathcal{C}_i)]$
<code>max()</code>	$\emptyset$	$score(d, \mathcal{Q}) = \max_{i=1..n} [score(d, \mathcal{C}_i)]$
<code>average()</code>	$\emptyset$	$score(d, \mathcal{Q}) = \frac{1}{n} \times \sum_{i=1}^n score(d, \mathcal{C}_i)$
<code>balanced_sum()</code>	$(\gamma_1, \dots, \gamma_n)$ tel que $\sum_{i=1}^n \gamma_i = 1$	$score(d, \mathcal{Q}) = \sum_{i=1}^n \gamma_i \cdot score(d, \mathcal{C}_i)$

TAB. III.26 – Calcul du score d'une requête complexe suivant la fonction d'évaluation employée

Exemple : Soit $\mathcal{Q}$ la requête complexe présentée à la page 52. Cette requête se compose de deux combinaisons : $\mathcal{C}_1$ dont l'évaluation a été montrée dans l'exemple précédent (voir page 65) et $\mathcal{C}_2$ dont l'évaluation est montrée ci-dessous :

		Q_a^{ex1}			
		Conjunct()	Disjunct()	Precision()	Recall()
Q_a^{ex4}	Disjunct()	$score(C_2, d_1) = 0$	$score(C_2, d_1) = 0$	$score(C_2, d_1) = 0$	$score(C_2, d_1) = 0$
		$score(C_2, d_2) = 0.33$	$score(C_2, d_2) = 1$	$score(C_2, d_2) = 0.55$	$score(C_2, d_2) = 0.66$
		$score(C_2, d_3) = 0$	$score(C_2, d_3) = 0$	$score(C_2, d_3) = 0$	$score(C_2, d_3) = 0$
		$score(C_2, d_4) = 0$	$score(C_2, d_4) = 0$	$score(C_2, d_4) = 0$	$score(C_2, d_4) = 0$
		$score(C_2, d_5) = 0$	$score(C_2, d_5) = 0$	$score(C_2, d_5) = 0$	$score(C_2, d_5) = 0$
	Precision()	$score(C_2, d_1) = 0$	$score(C_2, d_1) = 0$	$score(C_2, d_1) = 0$	$score(C_2, d_1) = 0$
		$score(C_2, d_2) = 0.11$	$score(C_2, d_2) = 0.77$	$score(C_2, d_2) = 0.33$	$score(C_2, d_2) = 0.44$
		$score(C_2, d_3) = 0$	$score(C_2, d_3) = 0$	$score(C_2, d_3) = 0$	$score(C_2, d_3) = 0$
		$score(C_2, d_4) = 0$	$score(C_2, d_4) = 0$	$score(C_2, d_4) = 0$	$score(C_2, d_4) = 0$
		$score(C_2, d_5) = 0$	$score(C_2, d_5) = 0$	$score(C_2, d_5) = 0$	$score(C_2, d_5) = 0$
	Recall()	$score(C_2, d_1) = 0$	$score(C_2, d_1) = 0$	$score(C_2, d_1) = 0$	$score(C_2, d_1) = 0$
		$score(C_2, d_2) = 0.11$	$score(C_2, d_2) = 0.77$	$score(C_2, d_2) = 0.33$	$score(C_2, d_2) = 0.44$
		$score(C_2, d_3) = 0$	$score(C_2, d_3) = 0$	$score(C_2, d_3) = 0$	$score(C_2, d_3) = 0$
		$score(C_2, d_4) = 0$	$score(C_2, d_4) = 0$	$score(C_2, d_4) = 0$	$score(C_2, d_4) = 0$
		$score(C_2, d_5) = 0$	$score(C_2, d_5) = 0$	$score(C_2, d_5) = 0$	$score(C_2, d_5) = 0$
	Fscore()	$score(C_2, d_1) = 0$	$score(C_2, d_1) = 0$	$score(C_2, d_1) = 0$	$score(C_2, d_1) = 0$
		$score(C_2, d_2) = 0.11$	$score(C_2, d_2) = 0.77$	$score(C_2, d_2) = 0.33$	$score(C_2, d_2) = 0.44$
		$score(C_2, d_3) = 0$	$score(C_2, d_3) = 0$	$score(C_2, d_3) = 0$	$score(C_2, d_3) = 0$
		$score(C_2, d_4) = 0$	$score(C_2, d_4) = 0$	$score(C_2, d_4) = 0$	$score(C_2, d_4) = 0$
		$score(C_2, d_5) = 0$	$score(C_2, d_5) = 0$	$score(C_2, d_5) = 0$	$score(C_2, d_5) = 0$

Vu le nombre important de possibilités de croisement de constructeurs sémantiques entre les différentes requêtes atomiques des deux combinaisons C_1 et C_2 , nous avons choisi d'illustrer l'évaluation de la requête complexe Q en imposant le constructeur sémantique **Fscore()** à toutes les requêtes atomiques contenues dans Q . Le détail de cette évaluation suivant les différentes fonctions proposées par le langage WeQueL est présenté dans le tableau III.27 page 67.

	Q				
	min()	max()	average()	balanced_sum()	
				$[\frac{1}{2}, \frac{1}{2}]$	$[\frac{3}{4}, \frac{1}{4}]$
d_1	0	0.08	0.04	0.04	0.06
d_2	0	0.38	0.19	0.04	0.09
d_3	0	0	0	0	0
d_4	0	0.17	0.08	0.08	0.02
d_5	0	0.20	0.10	0.10	0.05

TAB. III.27 – Évaluation d'une requête WeQueL complexe sur un exemple de documents

5 Évaluation expérimentale du langage WeQueL

Afin d’apprécier l’expressivité du langage WeQueL et afin de mesurer la précision de ses requêtes, nous avons mené nos expérimentations sur deux tâches différentes : la recherche thématique de documents dans le web et le filtrage de documents dans le cadre du projet eDot [eDot site] (voir section 4 page 9).

Pour la première tâche d’expérimentation, nous avons manuellement défini un échantillon de requêtes pour évaluer un ensemble de documents acquis depuis le web. L’objectif de cette évaluation est de retrouver, parmi les documents précédents, les documents traitant d’une des trois thématiques ciblées par notre expérimentation, à savoir : les cours informatiques en-ligne, les pages d’accueil de sites web sur le tourisme et des photos de formule 1. La précision de chaque requête de l’échantillon défini a été mesurée et comparée à la précision d’une recherche équivalente sous Google.

Pour la seconde tâche d’expérimentation, nous avons intégré le langage WeQueL dans le projet eDot [eDot site] afin de distinguer, parmi les documents rapatriés par le robot explorateur du projet, ceux traitant du risque alimentaire. Nous avons conçu, à cet effet, une requête dont l’évaluation sur un document quelconque fournit un score numérique pour apprécier la pertinence du document évalué par rapport à la thématique du projet eDot. Par ce score numérique, nous avons mis en œuvre un classement « objectif » de tous les documents rapatriés afin de ne garder que les documents totalisant un score élevé.

5.1 WeQueL : l’outil de recherche thématique

Dans cette section, nous montrons expérimentalement la précision des requêtes du langage WeQueL sur des recherches thématiques simples ciblant chacune diverses catégories (types MIME) de documents. Pour les trois thématiques choisies, les documents souhaités sont :

- des documents en français et au format **HTML**, **postscript**, **pdf** et **powerpoint** pour les cours en informatique ;
- des documents en français et au format **HTML** pour les pages d’accueil de sites web sur le tourisme ;
- des photos de formule 1 au format (**jpeg**, **bmp**, **gif**...) mais aussi des diapositives **HTML** regroupant plusieurs photos de formule 1 à la fois ;

Par ailleurs, nous avons défini et suivi au cours de nos tests un protocole expérimental spécifiant la construction de l’échantillon de requêtes à tester, l’acquisition du corpus de pages web à évaluer et les mesures utilisées pour estimer la qualité du langage.

5.1.1 L’échantillon des requêtes testées

Les requêtes, que nous avons testées, ont été manuellement élaborées en nous mettant à la place d’un utilisateur expérimenté dans le langage WeQueL et dans les trois thématiques choisies. Dans un premier temps, nous avons conçu 3 requêtes de référence " Q_{info} ", " Q_{tour} " et " Q_{F1} " pour décrire respectivement les caractéristiques des documents de cours informatiques en ligne, des documents de tourisme et des photos de formule 1. Nous appelons chacune de ces trois requêtes « *la requête descriptive de la thématique T d’expérimentation* » où $T \in \{info, tour, F1\}$.

Pour élaborer les requêtes descriptives des trois thématiques d’expérimentation, nous nous sommes appuyés sur un ensemble de documents pertinents recueillis manuellement depuis le web en

utilisant Google et l'annuaire web de Yahoo!. De ces documents, nous avons extrait les termes et les mots clés qui nous ont parus pertinents par rapport à chaque thématique d'expérimentation. Nous avons ensuite injecté manuellement ces termes et ces mots clés dans les sections correspondantes de la requête descriptive. Par exemple, pour la requête " Q_{info} ", nous avons constitué un échantillon d'une vingtaine de documents de cours informatiques en-ligne recueillis depuis la rubrique « [Guide Web > Sciences et technologies > Informatique > Cours et leçons en ligne](#) » de l'annuaire Yahoo! ainsi que depuis les réponses fournies par Google à quelques requêtes mots clés vagues (ex : « cours informatique »). Nous avons observé dans ces documents l'importance de la présence de certains termes dans des sections précises comme par exemple le terme « cours java » dans le titre ou « sommaire », « suivant » dans les liens sortants ... Nous avons en conséquence injecté ces mots clés dans les requêtes atomiques correspondantes. Les requêtes " Q_{info} ", " Q_{tour} " et " Q_{F1} " sont respectivement montrées dans les figures III.4, III.6 et III.5 pages 69, 71 et 70.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<WeQueLComplexQuery xmlns="http://www.lri.fr/~mezaour/WebServices/xml-schemas"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.lri.fr/~mezaour/WebServices/xml-schemas http://www.lri.fr/~mezaour/WeQueL/WeQueLschema.xsd"
  evaluation_function="max">

  <combination>
    <atomic_query name="mime" weight="1" semantic_builder="exactly" semantic_builder_parameters="1">
      text/html
    </atomic_query>
    <atomic_query name="title" weight="1" semantic_builder="Disjunct">
      cours informatique,cours java,cours algorithmique,cours intelligence artificielle,cours 'ia',cours html,cours programmation,support cours informatique,
      documentation cours informatique,cours 'c',cours bases de donnees,cours 'bd',tutoriel informatique,cours sgbd,documentation cours,cours automates,
      cours compilation,cours langage,cours calculabilite,cours logique,cours info,cours architecture,cours reseaux
    </atomic_query>
    <atomic_query name="url" weight="1" semantic_builder="Disjunct">
      cours informatique,cours java,cours algorithmique,cours 'ia',cours html,cours programmation,cours 'c',cours cpp,cours bases de donnees,cours 'bd',
      cours sgbd,cours algorithmique,cours automates,cours calcul,cours logique,cours info,cours architecture,cours compilation,cours reseaux,
      cours calculabilite,cours node,~ node,cours slide,cours transparent,~ slide,~ sld,cours sld,univ ~ cours,univ info cours
    </atomic_query>
    <atomic_query name="incoming_links" weight="1" semantic_builder="Disjunct">
      cours java,cours algorithmique,cours intelligence artificielle,cours html,cours programmation,cours informatique,support cours informatique,
      documentation cours informatique,cours 'c',cours cpp,cours bases de donnees,cours sgbd,cours 'bd',cours informatique en ligne,cours algo,
      cours calculabilite,cours logique,cours architecture,cours compilation,cours reseaux,note de cours,cours 'ia'
    </atomic_query>
    <atomic_query name="outgoing_links" weight="1" semantic_builder="Disjunct">
      introduction,table matieres,sommaire,chapitre,sommaire,introduction,preambule,precedent,suivant,nexts,previous
    </atomic_query>
  </combination>
  <combination>
    <atomic_query name="mime" weight="1" semantic_builder="exactly" semantic_builder_parameters="1">
      application/postscript,application/pdf,application/. *powerpoint,application/ppt,application/. *tar,application/. *zip,application/msword
    </atomic_query>
    <atomic_query name="url" weight="1" semantic_builder="Disjunct">
      cours informatique,cours java,cours algorithmique,cours ia,cours html,cours programmation,cours 'c',cours cpp,cours bases de donnees,cours 'bd',
      cours sgbd,cours algo,cours automates,cours calculabilite,cours logique,cours architecture,cours compilation,cours reseaux,cours slide,cours transparent,
      ~ slide,~ sld,cours sld,univ ~ cours,univ info cours
    </atomic_query>
    <atomic_query name="incoming_links" weight="1" semantic_builder="Disjunct">
      cours java,cours algorithmique,cours intelligence artificielle,cours html,cours programmation,cours informatique,notes de cours,
      support cours informatique,documentation cours informatique,cours 'c',cours cpp,cours bases de donnees,cours 'bd',cours sgbd,cours informatique en ligne,
      cours calculabilite,cours logique,cours architecture,cours compilation,cours reseaux,cours 'ia'
    </atomic_query>
  </combination>
  <combination>
    <atomic_query name="mime" weight="1" semantic_builder="exactly" semantic_builder_parameters="1">
      application/postscript,application/pdf,application/. *powerpoint,application/ppt,application/. *tar,application/. *zip,application/msword,text/html
    </atomic_query>
    <atomic_query name="titles incoming pages" weight="1" semantic_builder="exactly" semantic_builder_parameters="1">
      cours informatique,cours java,cours algorithmique,cours intelligence artificielle,cours 'ia',cours html,cours programmation,support cours informatique,
      documentation cours informatique,cours 'c',cours bases de donnees,cours 'bd',cours sgbd,documentation cours,cours automates,cours compilation,
      cours langage,cours calculabilite,cours logique,cours info,cours architecture,cours reseaux
    </atomic_query>
    <atomic_query name="url" weight="1" semantic_builder="Disjunct">
      cours informatique,cours java,cours algorithmique,cours ia,cours html,cours programmation,cours 'c',cours cpp,cours bases de donnees,cours 'bd',
      cours sgbd,cours algo,cours automates,cours calculabilite,cours logique,cours architecture,cours compilation,cours reseaux,cours slide,cours transparent,
      ~ slide,~ sld,cours sld,univ ~ cours,univ info cours
    </atomic_query>
  </combination>
</WeQueLComplexQuery>
```

FIG. III.4 – Q_{info} : Requête descriptive des cours informatiques en ligne

```

<?xml version="1.0" encoding="ISO-8859-1"?>
<WeQueLComplexQuery xmlns="http://www.lri.fr/~mezaour/WebServices/xml-schemas"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.lri.fr/~mezaour/WebServices/xml-schemas http://www.lri.fr/~mezaour/WeQueL/WeQueLschema.xsd"
  evaluation_function="max">
  <combination>
    <atomic_query name="mime" weight="1" semantic_builder="exactly" semantic_builder_parameters="1">
      text/html
    </atomic_query>
    <atomic_query name="title" weight="1" semantic_builder="Disjunct">
      photo gallery 'f1',photo gallery 'formule 1',photo gallery 'formule 1',photo 'f1',image 'formule 1',image 'f1',grand prix photo,
      'formula one' pic,'f1' picture,'formula one' gallery,formula one picture gallery,'f1' gallery,photo ferrari 'f1',ferrari f2002,renault formule 1,
      minardi asiatch f1,michael schumacher,schumacher photo,montoya photo,ralf schumacher photo,coulthard photo,jacques villeneuve photo,
      schumacher picture,montoya picture,ralf picture,coulthard picture,villeneuve picture
    </atomic_query>
    <atomic_query name="url" weight="1" semantic_builder="Disjunct">
      photo gallery 'f1',photo 'f1',photo,formule 1 photo,image formule 1,image 'f1',grand prix photo,formula one pic,pic formula one,picture 'f1',
      'f1' picture,formula one gallery,'f1' gallery,photo ferrari 'f1',ferrari f2002,formule 1 renault,'f1' renault,michael schumacher photo,photo montoya,
      schumacher ralf photo,coulthard photo,jacques villeneuve photo,schumacher picture,ralf picture,montoya picture,coulthard picture,villeneuve picture
    </atomic_query>
    <atomic_query name="incoming_links" weight="1" semantic_builder="Disjunct">
      photo formule 1,pic 'f1',image formule 1,image 'f1',grand prix photo,formula one pic,'f1' picture,formula one gallery,'f1' gallery,
      gallerie photos 'f1',gallerie photo formule 1,gallerie formule 1,gallerie 'f1',photo ferrari 'f1',photo mac laren 'f1',williams bmw 'f1',
      mac laren mercedes,ferrari f2002,renault formule 1,minardi asiatch f1,schumacher photo,montoya photo,ralf schumacher photo,coulthard photo,
      jacques villeneuve photo,schumacher picture,ralf picture,montoya picture,coulthard picture,villeneuve picture
    </atomic_query>
    <atomic_query name="outgoing_links" weight="1" semantic_builder="Disjunct">
      suivant,precedent,retour,next,previous,prev,back
    </atomic_query>
  </combination>
  <combination>
    <atomic_query name="mime" weight="1" semantic_builder="exactly" semantic_builder_parameters="1">
      image.*
    </atomic_query>
    <atomic_query name="url" weight="1" semantic_builder="Disjunct">
      f1 photo,photo formule 1,photo f1,image formule 1,image f1,grand prix photo,formula one pic,f1 picture,formula one gallery,f1 gallery,
      gallerie photo f1,gallerie photo formule 1,gallerie formule 1,gallerie f1,photo ferrari f1,photo mac laren f1,williams bmw f1,
      mac laren mercedes,ferrari f2002,renault formule 1,minardi asiatch f1,michael schumacher photo,montoya photo,ralf schumacher photo,
      coulthard photo,jacques villeneuve photo,schumacher picture,ralf picture,montoya picture,coulthard picture,villeneuve picture
    </atomic_query>
    <atomic_query name="incoming_links" weight="1" semantic_builder="Disjunct">
      photo formule 1,photo f1,image formule 1,image f1,grand prix photo,formula one pic,f1 picture,formula one gallery,f1 gallery,gallerie photo f1,
      gallerie photo formule 1,gallerie formule 1,gallerie f1,gallerie formule 1,photo ferrari f1,photo mac laren f1,williams bmw f1,
      mac laren mercedes,ferrari f2002,renault formule 1,minardi asiatch f1,schumacher photo,schumacher picture,montoya photo,montoya picture,
      ralf schumacher photo,ralf schumacher picture,coulthard photo,coulthard picture,jacques villeneuve photo,jacques villeneuve picture
    </atomic_query>
  </combination>
  <combination>
    <atomic_query name="mime" weight="1" semantic_builder="exactly" semantic_builder_parameters="1">
      text/html,image.*
    </atomic_query>
    <atomic_query name="url" weight="1" semantic_builder="Disjunct">
      f1 photo,photo formule 1,photo f1,image formule 1,image f1,grand prix photo,formula one pic,f1 picture,formula one gallery,f1 gallery,gallerie photo f1,
      gallerie photo formule 1,gallerie formule 1,gallerie f1,photo ferrari f1,photo mac laren f1,williams bmw f1,mac laren mercedes,ferrari f2002,
      renault formule 1,minardi asiatch f1,michael schumacher photo,montoya photo,ralf schumacher photo,coulthard photo,jacques villeneuve photo,
      schumacher picture,ralf picture,montoya picture,coulthard picture,villeneuve picture
    </atomic_query>
    <atomic_query name="titles incoming pages" weight="1" semantic_builder="exactly" semantic_builder_parameters="1">
      formula one picture,formula 1 picture,photo gallery 'f1',photo gallery 'formule 1',photo gallery 'formule 1',photo 'formule 1',photo 'f1',
      image 'formule 1',image 'f1',grand prix photo,'formula one' pic,'f1' picture,'formula one' gallery,'f1' gallery,photo ferrari 'f1',schumacher photo,
      schumacher picture,montoya photo,montoya picture,ralf schumacher photo,ralf schumacher picture,coulthard photo,coulthard picture,
      jacques villeneuve photo,jacques villeneuve picture,schumacher picture,montoya picture,coulthard picture,villeneuve picture
    </atomic_query>
  </combination>
</WeQueLComplexQuery>

```

FIG. III.5 – Q_{F1} : Requête descriptive des photos de formule 1 en ligne

```

<?xml version="1.0" encoding="ISO-8859-1"?>
<WeQueLComplexQuery xmlns="http://www.lri.fr/~mezaour/WebServices/xml-schemas"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.lri.fr/~mezaour/WebServices/xml-schemas http://www.lri.fr/~mezaour/WeQueL/WeQueLschema.xsd"
  evaluation_function="max">

  <combination>
    <atomic_query name="mime" weight="1" semantic_builder="exactly" semantic_builder_parameters="1">
      text/html
    </atomic_query>
    <atomic_query name="title" weight="1" semantic_builder="Disjunct">
      agence de tourisme, agence de voyage, agence de voyage en ligne, tour operateur, office de tourisme, vol charter, croisiere tourisme,
      reservation avion, vacances, vacances week end, vacances tourisme, vols discount, vol reguliers, voyage discount, achat reservation voyage,
      achat reservation voyages, achat reservation billet, achat reservation billets, sejours degriff7s, promotions voyages, promotion voyage,
      voyage en ligne, circuits croisiere, sejours, compagnie aerienne, train
    </atomic_query>
    <atomic_query name="url" weight="1" semantic_builder="Disjunct">
      agence, voyage, tourisme, sejour, tour operateur, office de tourisme, vol charter, croisiere, reservation avion, vacances, vols discount, vol reguliers,
      reservation voyage, reservation billet, circuits, croisiere, air
    </atomic_query>
    <atomic_query name="incoming_links" weight="1" semantic_builder="Disjunct">
      agence de tourisme, agence de tourisme en ligne, agence de voyage, agence de voyage en ligne, tour operateur, tour operateur en ligne,
      office de tourisme, office de tourisme en ligne, vol charter, croisiere tourisme, reservation avion, vacances week end, vacances tourisme,
      vols discount, vol reguliers, voyage discount, achat reservation voyage, achat reservation voyages, achat reservation billet,
      achat reservation billets, sejours degriffes, promotion voyage, voyage en ligne, location de voiture, circuits croisiere, sejour,
      société de transport en train
    </atomic_query>
    <atomic_query name="outgoing_links" weight="1" semantic_builder="Disjunct">
      reservez, reservation, achat, vols, offre, sejour, vacances, promotion, contact, liens utiles, hebergement, restauration, loisirs, plan, brochure, hotel,
      restaurant, sites, lieux, musée, monument
    </atomic_query>
    <atomic_query name="url_length" weight="1" semantic_builder="atmost" semantic_builder_parameters="2"/>
  </combination>
</WeQueLComplexQuery>

```

FIG. III.6 – Q_{tour} : Requête descriptive des services de tourisme en ligne

La diversité des requêtes atomiques que nous avons combinées dans chaque requête thématique WeQueL, nous a permis d'exprimer une description fine et détaillée des documents souhaités. À notre connaissance, aucun langage de requête pour le web ne permet de spécifier autant de critères de recherche dans une même requête.

Dans le but de montrer l'intérêt du choix des combinaisons dans les requêtes descriptives de chaque thématique, nous avons généré à partir de " Q_{info} ", " Q_{tour} " et " Q_{F1} " un certain nombre de requêtes moins restrictives que nous appelons *requêtes relaxées*.

Soit " Q " une requête du langage WeQueL contenant C combinaisons " $\mathcal{C}_i$ ". Chaque combinaison " $\mathcal{C}_i$ " combine K_i requêtes atomiques employant uniquement des constructeurs logiques.

Une requête relaxée d'ordre k , notée " $Q \downarrow k$ " et générée à partir de " Q ", est définie comme étant l'union de toutes les combinaisons de k ($k < K_i$) requêtes atomiques qu'il est possible de former à partir des K_i requêtes atomiques de chaque combinaison " $\mathcal{C}_i$ ". Pour des raisons évidentes de cohérence sémantique, la requête atomique relative au type MIME n'est pas relaxable. Autrement dit, la requête MIME d'une combinaison " $\mathcal{C}_i$ " de la requête " Q " est retrouvée dans toute combinaison relaxée d'ordre k générée à partir de " $\mathcal{C}_i$ ".

Exemple : Soit Q une requête WeQueL contenant une seule combinaison $\mathcal{C}$:

$$Q = 1 \cdot Q_{MIME} \cdot Q_1 \cdot Q_2 \cdot Q_3 \langle \emptyset \rangle$$

La combinaison de la requête Q combine trois requêtes atomiques Q_1 , Q_2 et Q_3 avec une requête sur le type MIME Q_{MIME} . La partie pondérée de cette combinaison est vide.

Compte tenu du nombre de requêtes atomiques de la combinaison $\mathcal{C}$, il n'est possible de générer

que deux relaxations à partir de la requête Q : $Q \downarrow 1$ et $Q \downarrow 2$. Ces deux relaxations s'expriment comme suit :

$$Q \downarrow 1 = (\mathcal{Q}_{MIME} \cdot \mathcal{Q}_1 \langle \emptyset \rangle) \vee (\mathcal{Q}_{MIME} \cdot \mathcal{Q}_2 \langle \emptyset \rangle) \vee (\mathcal{Q}_{MIME} \cdot \mathcal{Q}_3 \langle \emptyset \rangle)$$

$$Q \downarrow 2 = (\mathcal{Q}_{MIME} \cdot \mathcal{Q}_1 \cdot \mathcal{Q}_2 \langle \emptyset \rangle) \vee (\mathcal{Q}_{MIME} \cdot \mathcal{Q}_1 \cdot \mathcal{Q}_3 \langle \emptyset \rangle) \vee (\mathcal{Q}_{MIME} \cdot \mathcal{Q}_2 \cdot \mathcal{Q}_3 \langle \emptyset \rangle)$$

Dans le cadre de nos expérimentations, nous avons généré toutes les requêtes relaxées qu'il est possible de former à partir de " Q_{info} ", " Q_{tour} " et " Q_{F1} ", à savoir :

- 3 requêtes relaxées à partir de " Q_{info} ";
- 4 requêtes relaxées à partir de " Q_{tour} ";
- 3 requêtes relaxées à partir de " Q_{F1} ".

De par leur construction, chaque relaxation d'ordre k associée à une requête " Q " est moins restrictive et donc moins précise que sa requête d'origine " Q ". De plus, une relaxation d'ordre k est moins restrictive (donc moins précise) qu'une relaxation d'ordre $k + 1$.

5.1.2 Corpus des documents évaluées

Pour évaluer nos requêtes, nous avons été confrontés au problème du choix du graphe de documents web à construire. Pour des raisons de limitation de capacité de stockage et de temps d'exploration du web, nous ne pouvons pas évaluer nos requêtes sur la totalité du web ou du moins sur un index équivalent en taille à celui de Google par exemple. Nous avons alors étudié d'autres alternatives moins coûteuses et mieux adaptées à nos moyens matériels. Nous avons identifié deux possibilités différentes : construire un graphe de manière aléatoire ou construire 3 graphes thématiques en rapport avec les 3 domaines fixés dans la section précédente.

La construction d'un graphe aléatoire est incontestablement la solution la plus triviale à mettre en œuvre et la moins coûteuse en terme de développement. Cependant, cette solution ne garantit pas l'existence de documents pertinents parmi les documents du graphe aléatoire construit. Une telle situation engendre inévitablement un biais dans les résultats des évaluations. Nous avons, par conséquent, retenu la construction de 3 graphes thématiques.

Pour construire nos 3 graphes thématiques, nous avons utilisé Google [Google site] pour recueillir, dans un premier temps, des urls en rapport avec les 3 thématiques fixées. Ces urls correspondent aux réponses obtenues à la suite de la soumission à Google avancé [Google_advanced site] d'un ensemble de requêtes équivalentes aux trois requêtes descriptives de chaque thématique d'expérimentation. Aussi, nous avons développé en java un programme qui génère automatiquement les requêtes Google avancé à partir d'une requête WeQueL quelconque en procédant comme suit :

- Seules les requêtes atomiques de la requête WeQueL " $\mathcal{Q}_T$ ", ciblant des sections de documents reconnues par Google avancé, sont retenues pour être traduites. Ceci correspond aux requêtes atomiques portant sur le titre (*page_title*), l'url (*url*) et les liens entrants et sortants (*incoming_links*, *outgoing_links*).
- Pour toute requête atomique " $\mathcal{Q}_a^T$ " traduisible en Google avancé et ciblant la section S d'un document, nous générons une disjonction de requêtes Google avancé correspondant chacune à une valeur donnée de la requête " $\mathcal{Q}_a^T$ " :

$$\mathcal{Q}_a^T \xrightarrow{\text{Advanced Google}} \bigvee_{i=1}^N (\text{allin} "S" : value_i) \text{ telle que } value_i \in \mathcal{Q}_a^T$$

avec :

$$\begin{aligned} S &= title & si & Q_a^T \text{ porte sur le titre} \\ S &= url & si & Q_a^T \text{ porte sur l'url} \\ S &= anchor & si & Q_a^T \text{ porte sur les liens entrants ou sortants} \end{aligned}$$

Exemple : Soit Q_T la requête composée d'une seule combinaison de trois requêtes atomiques :
 $Q_T = 1 \cdot Q_{MIME} \cdot Q_{title} \cdot Q_{url}(\emptyset)$ avec :

Q_{MIME}	$=$	$MIME$	CONTAINS	text/html	$\xrightarrow{\text{EVALUATED BY}}$	Exactly(1)
Q_{title}	$=$	page_title	CONTAINS	cours java,cours cpp	$\xrightarrow{\text{EVALUATED BY}}$	Disjunct()
Q_{url}	$=$	url	CONTAINS	univ,cours	$\xrightarrow{\text{EVALUATED BY}}$	Conjunct()

À partir des requêtes atomiques traduisibles de Q_T , notre programme java génère la requête Google avancé suivante :

$$\begin{aligned} &(\text{allintitle : cours java}) \vee (\text{allintitle : cours cpp}) \vee \\ &(\text{allinurl : univ}) \vee (\text{allinurl : cours}) \end{aligned}$$

Chaque expression d'une disjonction en syntaxe Google avancé est soumise à Google pour récupérer en réponse au plus 100 urls (limite imposée par l'API Google [Google_API site]). Pour chaque url récupérée, nous gardons également son rang dans le classement de réponses fourni par Google. Précisons, par ailleurs, que les réponses de Google à nos requêtes ne sont pas nécessairement toutes pertinentes par rapport aux domaines que nous ciblons par notre expérimentation.

Les urls recueillies, à la suite de l'interrogation de Google, sont rapatriées et leurs documents correspondants sont stockés en local dans une base de données MySQL. Nous avons conçu le schéma de cette base pour contenir les informations nécessaires aux évaluations des requêtes WeQueL c'est-à-dire le contenu textuel de chaque section d'un document rapatrié : son titre, son url, ses liens entrants et sortants, ses tables, ses mots clés... Les documents contenus dans la base à ce niveau forment trois ensembles thématiques représentant chacun les points de départ à la construction des graphes d'évaluation associés à chaque thématique. En moyenne, ces ensembles de départ représentent environ 2000 documents (voir le tableau III.28 page 74).

Afin d'étendre les 3 ensembles de départ précédents, nous avons conçu un robot explorateur du web en java pour suivre les liens sortants des documents contenus dans les ensembles de départ de chaque thématique. La tâche principale de ce robot consiste à rapatrier récursivement les documents pointés par les urls de l'ensemble de départ et de les stocker dans la base MySQL prévue à cet effet. Dans le but d'augmenter les chances de rapatrier des documents pertinents par exploration, nous avons choisi de paramétrer notre robot de sorte qu'il ne suive que les liens sortant des documents les plus pertinents des ensembles thématiques de départ. Ces documents correspondent aux documents ayant leurs urls placées aux premières positions du classement de réponses fourni par Google. Ce choix se justifie par la qualité et la fiabilité avérées des premières réponses du classement d'urls fourni par Google à des requêtes mots clés.

À la fin de cette première étape, nous obtenons un niveau supplémentaire pour chaque graphe thématique à construire. Ce niveau correspond à l'extension des meilleures réponses de Google d'un pas. Le processus d'extension des graphes est répété p fois donnant un graphe final par domaine de profondeur p . Chaque niveau de ces graphes est construit comme suit :

- le niveau zéro contient les pages dont les urls sont fournies par Google en réponses à nos requêtes thématiques ;

- le niveau 1 est constitué de l'extension des pages les plus pertinentes du niveau 0 (les 2 meilleures réponses par chaque requête mots clés soumises) ;
- Les niveaux " i " ($1 < i < p$) sont construits à partir des pages issues de l'exploration semi-systématique du web à partir des liens sortants du niveau " $i - 1$ ". Afin de satisfaire les contraintes inhérentes à nos moyens matériels, nous avons défini un facteur de branchement " b " variable pour réduire le nombre de liens à suivre par page. Ce facteur de branchement est calculé par la formule $b = \frac{1}{i+1}$. Il permet de retenir uniquement $b\%$ liens choisis aléatoirement parmi les liens sortants d'un document du niveau " i " du graphe à construire.

Ce processus d'extension des réponses obtenues de Google nous garantit la possibilité d'inclure davantage de documents correspondant à nos thématiques d'expérimentation en vertu de la forte connectivité thématique qui existe dans le web [Kleinberg 1999, Brin et Page 1998, Diligenti et al. 2000]. En effet, un document pertinent pointe souvent sur d'autres documents pertinents traitant du même thème. Suivre les liens d'un tel document augmente forcément la chance d'inclure d'autres documents pertinents dans le graphe thématique. Une forte présence de documents pertinents dans nos graphes permet de mieux apprécier la précision de l'échantillon de requêtes WeQueL constituées. La taille du graphe construit pour chaque thématique est montrée dans le tableau III.28 page 74.

Thématique	# Réponses Google	# de documents dans le graphe	# de liens suivis
cours informatiques en ligne	2.404	271.850	4.347.930
annuaire de documents de tourisme	2.027	239.817	5.916.248
photos de formule 1	1.599	149.634	8.990.615

TAB. III.28 – Taille des trois graphes thématiques d'expérimentation

5.1.3 Mesures de qualité

Nous avons retenu deux mesures pour évaluer la qualité des réponses obtenues à la suite de l'évaluation de notre échantillon de requêtes : la précision et la couverture.

La précision d'une requête " Q_T " de l'échantillon des requêtes testées correspond à la proportion de documents réellement pertinents parmi les documents pertinents par rapport à la requête " Q_T ". Compte tenu du fait que les requêtes atomiques de chacune de nos requêtes d'expérimentation emploient des constructeurs logiques, un document d est pertinent par rapport à la requête " Q_T " si et seulement si son score d'évaluation sur la requête " Q_T " est non nul, c'est-à-dire égal à 1. Ce document est dit réellement pertinent lorsque l'expert humain de l'expérimentation considère que le contenu de d est en rapport avec la thématique T ciblée par la requête " Q_T ".

La couverture d'une requête " Q_T " de l'échantillon des requêtes testées correspond au taux de documents réellement pertinents trouvés dans les réponses de " Q_T " parmi les documents réellement pertinents contenus dans le graphe correspondant à la thématique " T ". Soient :

- " $\mathcal{D}_T$ " l'ensemble des documents du graphe associé à la thématique " T " ;
- " $\mathcal{S}_{Q_T}$ " l'ensemble des documents de " $\mathcal{D}_T$ " qui sont pertinents par rapport à la requête " Q_T " (documents évalués avec un score égal à 1) ;

La modalité de pertinence nous permet de diviser chacun des deux ensembles " $\mathcal{D}_T$ " et " $\mathcal{S}_{Q_T}$ " en deux sous-ensembles de documents réellement pertinents et de documents non-pertinents : $\mathcal{D}_T = \mathcal{D}_T^{rp} \cup \mathcal{D}_T^{np}$, $\mathcal{S}_{Q_T} = \mathcal{S}_{Q_T}^{rp} \cup \mathcal{S}_{Q_T}^{np}$.

La précision et la couverture d'une requête " Q_T " s'exprime en fonction des sous-ensembles précédents comme suit :

$$\text{précision} = \frac{|\mathcal{S}_{Q_T}^{rp}|}{|\mathcal{S}_{Q_T}|} \quad \text{couverture} = \frac{|\mathcal{S}_{Q_T}^{rp}|}{|\mathcal{D}_T^{rp}|}$$

La taille de chaque graphe thématique (voir tabelau III.28 page 74) et des réponses retournées lors des évaluations étant importantes, nous avons eu recours à des techniques d'échantillonnage pour estimer la précision et la couverture. Nous avons constitué, à chaque évaluation, un échantillon aléatoire de 100 pages et nous l'avons manuellement étiqueté en pertinent ou pas (estimation de $|\mathcal{S}_{Q_T}^{rp}|$). Nous avons fait de même pour estimer la proportion de documents réellement pertinents dans l'ensemble $\mathcal{D}_T$ du graphe thématique consrui ($|\mathcal{D}_T^{rp}|$).

5.1.4 Résultats expérimentaux

Les résultats obtenus sur les trois thèmes fixés pour cette première expérimentation, sont résumés dans le tableau III.29 page 75. Les faibles résultats dans certains cas (notamment pour Google) s'expliquent en partie par l'extrême sévérité de notre processus d'étiquetage manuel. En effet, seuls les documents que nous jugeons en rapport direct avec la thématique ciblée sont étiquetés à réellement pertinents. Les autres sont considérés comme non pertinents et cela même s'il sont vaguement en rapport avec la thématique considérée. Par exemple, les documents qui en soi ne sont pas pertinents mais qui font référence à des documents réellement pertinents sont étiquetés comme non pertinents.

requête	Thématique T					
	cours informatiques		annuaire de tourisme		photos de formule 1	
	<i>précision</i>	<i>couverture</i>	<i>précision</i>	<i>couverture</i>	<i>précision</i>	<i>couverture</i>
Google classique	9,00%	6,68%	10,00%	7,29%	9,00%	22,99%
Google avancé	26,00%	5,82%	21,00%	4,14%	32,00%	9,47%
" Q_T "	71,00%	11,16%	100,00%	2,30%	65,53%	9,44%
" $Q_T \downarrow 4$ "	∞	∞	29,29%	10,56%	∞	∞
" $Q_T \downarrow 3$ "	56,00%	56,57%	8,00%	21,74%	41,00%	36,11%
" $Q_T \downarrow 2$ "	65,00%	63,31%	7,00%	61,56%	36,00%	35,99%
" $Q_T \downarrow 1$ "	10,00%	100%	2,00%	100%	8,00%	93,01%

TAB. III.29 – Précision et couverture des requêtes thématiques du langage WeQueL

Remarque 5.1 *Les cases du tableau III.29 page 75 contenant la valeur infinie (∞) représentent des requêtes pour lesquelles il n'est pas possible de calculer la relaxation. Plus précisément, les requêtes " Q_{info} " et " Q_{F1} " contiennent des combinaisons contenant moins de 4 requêtes atomiques.*

L'estimation de la précision de Google avancé sur une thématique T est calculée sur les documents du niveau 0 du graphe construit pour la thématique. En effet, les documents du niveau 0 de ce graphe proviennent des réponses fournies par Google avancé à une requête en syntaxe Google avancé équivalente à la requête WeQueL décrivant la thématique T .

L'estimation de la précision de Google classique sur une thématique T est calculée sur les documents donnés en réponse à la soumission d'une requête mots clés à Google classique. Google classique ne permettant pas de cibler des mots clés sur des sections de documents, nous avons conçu

cette requête pour qu'elle décrive au mieux la thématique T et cela en effectuant, dans une seule requête mots clés, l'union de toutes les valeurs (mots clés) contenues dans la requête décrivant la thématique T .

5.1.5 Discussion

Il apparaît très clairement des deux premières lignes du tableau III.29 page 75 qu'une requête ciblée sur une section particulière d'un document (Google avancé) est plus précise qu'une requête recherchant les mots clés n'importe où dans le contenu d'un document (Google classique).

En comparant, cette fois-ci, les résultats de la ligne Google avancé du tableau précédent avec les résultats de nos requêtes descriptives pour chaque thématique, nous remarquons qu'une requête WeQueL, combinant diverses requêtes ciblées, est en moyenne trois fois plus précise que Google avancé (requête ciblée sans combinaison). Ce gain en précision conforte expérimentalement notre intuition sur l'utilité d'avoir différentes combinaisons de plusieurs requêtes ciblées à la fois. Ceci nous permet de conclure que les possibilités de construction de requêtes sous le langage WeQueL sont suffisamment riches et expressives pour permettre à un utilisateur de mieux formuler ses besoins de recherche de documents.

En comparant les résultats de l'évaluation de chacune des requêtes " Q_T " aux résultats des relaxations qui leur correspondent, nous remarquons que la précision décroît au fur et à mesure que le nombre de requêtes atomiques combinées décroît. Cette perte met en évidence le lien étroit qui existe entre la précision d'une requête et le nombre de requêtes atomiques combinées à la fois dans une même requête. En d'autres mots, une combinaison est vue comme un ensemble de contraintes (requêtes atomiques) imposées aux documents à évaluer. Plus l'utilisateur multiplie les contraintes dans sa requête, plus les documents évalués à 1 ont de chances de correspondre aux documents qu'il souhaitait et vice versa. Cependant, comme dans tout système à base de contraintes, plus les contraintes sont fortes et nombreuses, moins il y a de documents qui les satisfont toutes à la fois. Ceci explique en partie les faibles résultats de couverture de nos requêtes les plus contraignantes à savoir " Q_T ".

Le faible taux de couverture des trois requêtes " Q_{info} ", " Q_{tour} " et " Q_{F1} " s'explique par le mode restrictif d'appariement des valeurs de nos requêtes atomiques sur les documents du web. En effet, en limitant l'appariement des valeurs aux seules valeurs mentionnées dans les requêtes atomiques, nous avons conscience de ne pas couvrir tous les documents réellement pertinents. Par exemple, les documents réellement pertinents utilisant un vocabulaire différent de celui employé dans la requête sont sous estimés lors de leur évaluation. L'utilisation de dictionnaires en ligne ou de thésaurus/ontologies de domaine peut être envisagée comme solution complémentaire pour enrichir le vocabulaire d'une requête WeQueL.

Par ailleurs, cette expérimentation soulève un nouveau problème inhérent, cette fois-ci, à la nature des requêtes combinées, c'est-à-dire à l'utilisation exclusive de constructeurs sémantiques logiques. En effet, si les trois requêtes thématiques prévues pour cette expérimentation se sont montrées suffisamment précises pour caractériser des documents thématiques, elles demeurent néanmoins inexploitable pour proposer un classement par pertinence décroissante. L'évaluation binaire (0 ou 1) des documents sur ces requêtes ne permet pas de distinguer le degré de pertinence des documents totalisant le même score (score 1 par exemple).

5.2 WeQueL : l'outil de filtrage du projet « eDot »

Dans le cadre du projet eDot [eDot site] (voir présentation du projet dans le chapitre d'introduction de ce manuscrit à la section 4 page 9), nous avons mis à profit la souplesse et la précision du langage WeQueL pour mettre en place un système de filtrage de documents web. Nous avons développé un module logiciel intitulé « **eDotHTMLFilter** » sous forme d'un web service disponible à l'adresse <http://edot.lri.fr:8080/axis/services/eDotHTMLFilter?wsdl>.

eDotHTMLFilter [Mezaour 2004a, Mezaour 2005] est un service générique qui permet de filtrer un ensemble d'urls de documents fournis en entrée par rapport à leur pertinence vis-à-vis d'une requête thématique du langage WeQueL, fournie également en entrée. Dans le contexte du projet eDot, le filtrage est effectué par rapport à la requête WeQueL utilisée pour décrire les documents pertinents souhaités dans l'entrepôt thématique sur le risque alimentaire. Les experts microbiologistes du projet ont défini un document pertinent pour l'entrepôt comme étant un document au format HTML doté de tables contenant des données sur le risque alimentaire. Ces données correspondent aux termes et aux mots clés de l'ontologie sur le risque alimentaire utilisée dans eDot.

Partant de cette expertise, nous avons rédigé une requête WeQueL dont l'évaluation permet de tester, chez les documents à filtrer, la présence des termes de l'ontologie dans des tableaux HTML. Nous désignons cette requête par la requête de filtrage. La requête de filtrage ainsi que le protocole expérimental suivi pour la tâche de filtrage sont décrits ci-dessous.

5.2.1 Protocole expérimental

Pour valider notre approche de filtrage, nous avons défini un protocole expérimental dans lequel nous spécifions les documents filtrés, la construction de la requête de filtrage et la méthodologie de mesure de la qualité du processus de filtrage. Nous spécifions également une requête concurrente à la requête de filtrage à des fins de comparaison de performances.

La requête de filtrage : Pour décrire les documents traitant du risque alimentaire, nous avons élaboré une requête WeQueL numérique, notée " Q_{Num} ", contenant une seule combinaison WeQueL pour cibler des documents au format HTML. Nous avons intégré dans cette combinaison huit requêtes atomiques décrivant chacune un aspect particulier des documents visés (l'url, le titre, les liens entrants/sortants, ...). Nous avons choisi d'utiliser le constructeur numérique **Fscore()** pour toutes les requêtes atomiques de la précédente combinaison. Le constructeur **Fscore()** combine deux constructeurs numériques **precision()** et **recall()** (voir tableau III.12 page 59) qui permettent respectivement d'apprécier le taux de mots clés de la requête atomique présent dans la section ciblée et le taux de valeurs de la requête atomique couvertes par cette même section. La requête " Q_{Num} " est montrée dans la figure III.7 page 78.

```

<?xml version="1.0" encoding="iso8859-1"?>

<WeQuELComplexQuery xmlns="http://www.lri.fr/~mezaour/WebServices/xml-schemas"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.lri.fr/~mezaour/WebServices/xml-schemas http://www.lri.fr/~mezaour/WeQuEL/WeQuELSchema.xsd"
  evaluation_function="max">

  <combination>
    <atomic_query name="mime" weight="1" semantic_builder="exactly" semantic_builder_parameters="1">
      text/html
    </atomic_query>
    <atomic_query name="title" weight="0.05" semantic_builder="Fscore">
      'aw', 'fs', 'ph', acidic food, alkaline, alkaline acid, alkaline food, alkaline water, anti microbial, approximate, bacteriostatic, biochemical pesticides,
      biocide, biological pesticides, biopesticide, complist, composition databank, composition detail, composition list, coralcalcium, coralcalcium food,
      databank food, databank list, database food, decontamination fruit, disinfectant, fact sheet, food, food alkaline, food chart, food composition, food databank,
      food database, food resources, food safety, food science, fruit eaten raw, fruit vegetable, fs food, fungicide, herbicide, insecticide, list component, microbial,
      nutrient, nutrition, nutritional, pesticide, phlife coralcalcium, product, rodenticide, sheet water, sterilant, sterilant disinfectant, surface decontamination,
      value, value vegetable, vegetable eaten raw, water, water activity, water fs
    </atomic_query>
    <atomic_query name="url" weight="0.05" semantic_builder="Fscore">
      'aw', 'fs', 'ph', chart, complist, coralcalcium, data, databank, database, details, fact sheet, food, food comp, lacf phs, list, nutrient, nutrition, nutritional, phlife,
      product, safety, value, water activity, water fs
    </atomic_query>
    <atomic_query name="keywords" weight="0.05" semantic_builder="Fscore">
      'aw', 'fs', 'ph', acidic, agriculture, alkaline, anti microbial, antimicrobial, approximate, bacteriostatic, bio, biochemical, biocide, chart, chemical, coralcalcium,
      database, databank, disinfectant, eps, fact sheet, food, food chart, fungicide, nutrient, nutrition, nutritional, pesticide, product, resources, rodenticide, sanitizer,
      sterilant, value, water, water activity
    </atomic_query>
    <atomic_query name="outgoing links" weight="0.05" semantic_builder="Fscore">
      'aw', 'ph', acid, added, alcohol, alka, alkaline food, alkaline water, alpha tokoferol, alphabetical list, ammonium compound, approximate, ascorbic acid, aureus
      clostridium, bacillus, bacillus cereus, beverages, biotin, bromine, calcium, calcium fact, campylobacter, campylobacter yersinia, canned, carbohydrate, carotene,
      cereal, cereus, cheese, chlorine, cholecalciferol, cholesterol, chromium, clostridium, component, component list, composition data, composition databank, compound,
      content, copper, coral calcium, cream, data food, databank, database, dehydroascorbic acid, dietary, diketogulonic acid, dietary, diketogulonic acid, dioxide,
      edible ice, egg, energy, enterocolitica, ergocalciferol, escherichia coli, fact sheet, fat oil, fatty acid, fibre, fish, fisheries, folacin, food, food composition,
      food data, food group, food name, food safety, food science, food test, food work, foodcomp, fruit, groupsearch, hacpp approach, honey, hydrogen,
      hydroxycholecalciferol, ice, index, ingredient, iodine, iodine manganese, ionic, iron, iron magnesium, iron phosphorus, irradiation, list component, list search,
      listeria, listeria monocytogenes, low acid food, magnesium, manganese, meat, microbiol, milk, mixed, mixed food, moisture, monounsaturated, name food, namesearch,
      niasin, nickel, nutrition, nutritional, oil, pantothenic acid, parasite, pathogen, per unit, phlife, phosphorus, polyunsaturated fatty acid, potassium, poultry,
      product, protein, ammonium, refractometry, result, retinol, riboflavin, risk, safety food, safety hacpp, salmonella, selenium, sheet water, shigella, sodium,
      source, list, spice, staphylococcus, staphylococcus aureus, structure, sugar, summary, thiamin, total, trisodium, phosphate, tryptophan, unit, variation source,
      vegetable, vibrio, virus, vitamin, water, water activity, water alkaline, water phlife, water structure, wateract, yersinia enterocolitica, zinc
    </atomic_query>
    <atomic_query name="incoming links" weight="0.025" semantic_builder="Fscore">
      'aw', 'fs' water, 'ph', acid base, acidified, activity food, approximate, canned, chemistry, complist, component list, corresponding food, cream, databank food,
      database food, drying food, fact, fact sheet, food, food comp, food composition, food data, food databank, food database, food drug, food group, food manufacturer,
      food name, food number, food nutrient, food report, food safety, food standard, group search, lacf phs, list, low acid food, neutral, nutrient, nutrient database,
      nutrient list, nutrition, nutrition data, nutrition fact, phs approximate, product, properties, property, search food group, value, value food, water, water
      activity, water fs
    </atomic_query>
    <atomic_query name="titles incoming pages" weight="0.025" semantic_builder="Fscore">
      'aw', 'fs', 'ph', acid food, acid regulation, acidified, aflatoxin, agriculture, bacillus myxomatosis, behavior, biology, biology food, biotechnology, birimeter,
      calcium, canned, carbon monoxide, care feeding, chart, chart brix, chemistry, coccidia, coccidia intestine, colic vomit, comm lacf, composition databank,
      conductivity, databank, database, deficiency selenium, diet, diet care, diet stenosis, drying, entomology food, fact food, fact sheet, feeding, food, food
      composition, food data, food history, food industry, food nutrient, food nutrition, food preservation, food resource, food safety, food science, food standard,
      food technology, food timeline, foodcomp, freeze, freezing, fruit, fruit vegetable, growing, hacpp, health, home food, horticulture food, hyperthyroidism, lacf,
      low acid, multocida, mycotoxins, mycotoxin, myxomatosis, neuter preparation, nutrient database, nutrition, nutrition information, nutrition nourishment,
      organic, oxalates, pasteurization, pasteurization, phosphide, physiology nutrition, pickling, poison, preservation, preservation canning, quality, refractometer,
      resource food, science fact, science food, search food, selenium, standard, substance, technology food, timeline food, toxin, vegetable quality, water activity,
      water food
    </atomic_query>
    <atomic_query name="object" type="tables lists" weight="0.75" semantic_builder="Fscore">
      'f', 'c', 'anise', 'apple', 'ash', 'aw', 'bean', 'bran', 'brine', 'broth', 'ca', 'chop', 'cider', 'chive', 'chive', 'cob', 'cod', 'liver', 'corn', 'cow', 'cr', 'cu', 'date', 'dill',
      'dish', 'dry', 'drying', 'fe', 'flan', 'flour', 'germ', 'gr', 'gram', 'grams', 'gruel', 'ham', 'ions', 'iron', 'item', 'kj', 'lamb', 'lime', 'list', 'loin', 'mg', 'mn',
      'moss', 'na', 'ne', 'nest', 'okra', 'pack', 'pasta', 'paste', 'pate', 'per cent', 'per unit', 'per', 'pet', 'food', 'ph', 'pie', 'quince', 'range', 'raw', 'ripe', 'salt',
      'se', 'seed', 'slice', 'smelt', 'sour', 'soy', 'soya', 'steak', 'stew', 'tofu', 'tuna', 'unit', 'wine', 'zn', 'abalone, acid, acidified, acidity, acidophilus, acorn,
      activity, added, additive, adipose tissue, aged, air, alcohol, alcoholic drink, alkaline, aloe, aloe vera, alpha tokoferol, analytical, anchovies, anchovy, anise
      fennel, anon, asoc, approach, approximate, apricot, arrowroot, artichoke, ascorbic, asparagus, asparagus bud, asparagus rattan, atmosphere conditioning, avocados,
      baby soup, backbone, bacon, bacteria, bacteria yeast, bacterial, bacteriological, baked, bakery, bamboo, banana, bantam, barbecued, bartlett pear, based, basil, basil
      pesto, bass, batavia, bean, beat, bechamel, beef, beef fat, beef striploin, beet, belgian endive, beverage, biochemical, biochemical reaction, biotin, biscuit, bisque,
      blackberries, blackberry, blue veined cheese, blueberries, bluefish, boiled, bologna, bolognese, bolognese, brain, bread, bread rye, bread spoil, breadfruit,
      brisket, brix, broccoli, brocoli, broiled, broth, broth shallot, bruised, brussel sprout, btw, bud, stalk, bulb, burger, butchery, butter, buttermilk, cabbage, cactus,
      cactus calamary, cactus cantaloupe, caesar, cake, cake mix, calamary, calcium, caloric, calorie, camembert, candied, cane sugar, canned, caper, caprine, carmel,
      carbohydrate, carbs, carcass, carotene, carp, carrot, category, cauliflower, caviar, celery, cephalopod, cereal, characteristic, carbohydrate, chard, chayote,
    </atomic_query>
  </combination>
</WeQuELQuery>

```

FIG. III.7 – La requête " Q_{Num} " de filtrage

Conformément aux spécifications des experts microbiologistes, nous avons consacré une requête atomique pour cibler la recherche des termes de l'ontologie d'eDot sur les tables et les listes d'un document HTML. Rappelons que l'objectif du projet est de compléter la base de données Sym'Previus en données issues d'expérimentations en biologie (voir description du projet eDot en introduction de ce manuscrit, cf. section 4 page 9). Les microbiologistes ont coutume de publier ce type de données sous forme de tableaux, ce qui justifie l'emploi d'une requête atomique ciblant les tables et listes HTML d'un document. Nous avons ainsi utilisé la requête atomique *object* paramétrée pour cibler les tables et les listes HTML [*type = table_list*]. Aussi, nous avons accordé un poids plus important à cette requête (= 0,75) compte tenu de l'importance de la présence des termes de l'ontologie d'eDot dans des tables HTML pour la détermination de la pertinence d'un document.

Nous avons, par ailleurs, exploité un ensemble d'exemples de documents pertinents fournis par

les experts pour alimenter en valeurs les autres requêtes atomiques. Ce processus consiste à injecter les mots clés du contenu de chaque section des documents exemples dans les requêtes atomiques correspondantes. À cet effet, nous avons conçu un programme java qui rapatrie, analyse et représente les documents exemple par une table *WDM*. À partir du contenu textuel de chaque entrée, notre programme suggère plusieurs valeurs à insérer dans la requête atomique de filtrage correspondant à cette entrée. Les valeurs suggérées sont obtenues par le déplacement, par pas de 1 mot à chaque fois, d'une fenêtre de deux mots clés successifs sur le contenu textuel de l'entrée considérée. Bien entendu, nous validons manuellement chaque valeur suggérée avant de l'injecter dans la requête atomique et cela afin d'éliminer les propositions présentant peu d'intérêt pour la tâche de filtrage.

Exemple : Le titre de l'un des documents pertinents fournis en entrée à notre programme java est " FDA/CFSCAN - Approximate pH of Foods and Food product ". Après chargement et élimination des mots communs de ce document, notre programme décompose le contenu de l'entrée *title* de la table *WDM* du document précédent (devenu maintenant " fda cfscan approximate ph foods food product ") en une série de deux mots successifs pour suggérer six valeurs différentes à insérer dans la requête *title* : fda cfscan, cfscan approximate, approximate ph, ph foods, foods food, food product. Dans le processus de validation manuelle, nous avons écarté les suggestions présentant peu d'intérêt, c'est-à-dire : fda cfscan, cfscan approximate, foods food.

Grâce au score numérique de l'évaluation de chaque document sur la requête " Q_{Num} ", nous montrons qu'il est possible d'établir un classement « objectif » de la pertinence des documents évalués.

Les documents filtrés : Les documents que nous avons eu à filtrer dans le cadre du projet eDot ont été récoltés à la suite du processus d'exploration systématique du web effectué par le robot explorateur de Xyleme [Sédrati-Dinet 2003, Mignet et al. 2001]. Ce robot a été paramétré pour explorer la totalité du web à la recherche de documents satisfaisant une requête mots clés définie par nos experts microbiologistes et que nous appelons « *la requête de préfiltrage* ». La stratégie d'exploration du robot de Xyleme consiste à privilégier le rapatriement des documents populaires (ayant un PageRank élevé). Pour le calcul de l'importance de chaque document exploré, le robot de Xyleme utilise l'algorithme OPIC [Abiteboul et al. 2003] (c.f. section 1.4 page 107).

La requête de préfiltrage, sous forme d'une combinaison logique des termes de l'ontologie d'eDot, est évaluée sur le contenu entier de chaque document exploré par le robot de Xyleme. L'évaluation à *vrai* d'un document d donné sur cette requête logique garantit la présence d'un certain nombre de termes de l'ontologie eDot dans le contenu de d sans que cette présence ne puisse être ciblée sur des tableaux HTML. Dans ce cas, le document d est considéré comme étant « *potentiellement* » pertinent. L'intérêt de notre filtrage réside dans la capacité de notre requête " Q_{Num} " à statuer sur la pertinence d'un document en ciblant la recherche des termes de l'ontologie eDot sur les tableaux HTML. Dans la suite, nous référons à l'ensemble des documents « potentiellement » pertinents par l'ensemble $\mathcal{D}_f$.

La requête comparative : Afin de montrer l'intérêt du score numérique produit par " Q_{Num} ", nous avons élaboré une requête comparative notée " Q_{Log} " produisant un score binaire : 0 ou 1. La requête " Q_{Log} " combine les mêmes requêtes atomiques avec les mêmes valeurs que les requêtes atomiques de " Q_{Num} ". À la différence des requêtes atomiques de " Q_{Num} ", les requêtes atomiques

de " Q_{Log} " emploient le constructeur logique `Disjunct()`. La requête " Q_{Log} " est montrée dans la figure III.8 page 80.

```
<?xml version="1.0" encoding="iso8859-1"?>
<WeQueLComplexQuery xmlns="http://www.lri.fr/~mezaour/WebServices/xml-schemas"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.lri.fr/~mezaour/WebServices/xml-schemas http://www.lri.fr/~mezaour/WeQueL/WeQueLschema.xsd"
  evaluation_function="MAX">
  <combination>
    <atomic_query name="mime" weight="1" semantic_builder="exactly" semantic_builder_parameters="1">
      text/html
    </atomic_query>
    <atomic_query name="title" weight="1" semantic_builder="disjunct">
      'aw', 'fs', 'ph', acidic food, alkaline, alkaline acidic, alkaline food, anti microbial, approximate, bacteriostatic, biochemical pesticides,
      biocide, biological pesticides, biopesticide, complist, composition databank, composition detail, composition list, coralcalcium, coralcalcium food,
      databank food, databank list, database food, decontamination fruit, disinfectant, fact sheet, food, food alkaline, food chart, food composition, food databank,
      food database, food resources, food safety, food science, fruit eaten raw, fruit vegetable, fs food, fungicide, herbicide, insecticide, list component, microbial,
      nutrient, nutrition, nutritional, pesticide, phlife coralcalcium, product, rodenticide, sheet water, sterilant, sterilant disinfectant, surface decontamination,
      value, value vegetable, vegetable eaten raw, water, water activity, water fs
    </atomic_query>
    <atomic_query name="url" weight="1" semantic_builder="disjunct">
      'aw', 'fs', 'ph', chart, complist, coralcalcium, data, databank, database, details, fact sheet, food, food comp, lacf phs, list, nutrient, nutrition, nutritional, phlife,
      product, safety, value, water activity, water fs
    </atomic_query>
    <atomic_query name="keywords" weight="1" semantic_builder="disjunct">
      'aw', 'fs', 'ph', acidic, agriculture, alkaline, anti microbial, antimicrobial, approximate, bacteriostatic, bio, biochemical, biocide, chart, chemical, coralcalcium,
      database, databank, disinfectant, epa, fact sheet, food, food chart, fungicide, nutrient, nutrition, nutritional, pesticide, product, resources, rodenticide, sanitizer,
      sterilant, value, water, water activity
    </atomic_query>
    <atomic_query name="outgoing links" weight="1" semantic_builder="disjunct">
      'aw', 'ph', acid, added, alcohol, alka, alkaline food, alkaline water, alpha tokoferol, alphabetical list, ammonium compound, approximate, ascorbic acid, aureus
      clostridium, bacillus, bacillus cereus, beverages, biotin, bromine, calcium, calcium fact, campylobacter, campylobacter yersinia, canned, carbohydrate, carotene,
      cereal, cereus, cheese, chlorine, cholecalciferol, cholesterol, chromium, clostridium, component, component list, composition data, composition databank, compound,
      content, copper, coral calcium, cream, data food, databank, database, dehydroascorbic, dehydroascorbic acid, dietary, diketogulonic, diketogulonic acid, dioxide,
      edible ice, egg, energy, enterocolitica, ergocalciferol, escherichia coli, fact sheet, fat oil, fatty acid, fibre, fish, fisheries, folacin, food, food composition,
      food data, food group, food name, food safety, food science, food test, food work, foodcomp, fruit, groupersearch, haccp approach, honey, hydrogen,
      hydroxycholecalciferol, ice, index, ingredient, iodine, iodine manganese, ionic, iron, iron magnesium, iron phosphorus, irradiation, list component, list search,
      listeria, listeria monocytogenes, low acid food, magnesium, manganese, meat, microbiol, milk, mixed, mixed food, moisture, monounsaturated, name food, namesearch,
      niacin, nickel, nutrition, nutritional, oil, pantothenic acid, parasite, pathogen, per unit, phlife, phosphorus, polyunsaturated fatty acid, potassium, poultry,
      product, protein, ammonium, refractometry, result, retinol, riboflavin, risk, safety food, safety haccp, salmonella, selenium, sheet water, shigella, sodium,
      sourcelist, spice, staphylococcus, staphylococcus aureus, structure, sugar, summary, thiamin, total, trisodium, phosphate, tryptophan, unit, variation source,
      vegetable, vibrio, virus, vitamin, water, water activity, water alkaline, water phlife, water structure, wateract, yersinia enterocolitica, zinc
    </atomic_query>
    <atomic_query name="incoming links" weight="1" semantic_builder="disjunct">
      'aw', 'fs' water, 'ph', acid base, acidified, activity food, approximate, canned, chemistry, complist, component list, corresponding food, cream, databank food,
      database food, drying food, fact, fact sheet, food, food comp, food composition, food data, food databank, food database, food drug, food group, food manufacturer,
      food name, food number, food nutrient, food report, food safety, food standard, group search, lacf phs, list, low acid food, neutral, nutrient, nutrient database,
      nutrient list, nutrition, nutrition data, nutrition fact, phs approximate, product, properties, property, search food group, value, value food, water, water
      activity, water fs
    </atomic_query>
    <atomic_query name="titles incoming pages" weight="1" semantic_builder="disjunct">
      'aw', 'fs', 'ph', acid food, acid regulation, acidified, aflatoxin, agriculture, bacillus myxomatosis, behavior, biology, biology food, biotechnology, birxmeter,
      calcium, canned, carbon monoxide, care feeding, chart, chart brix, chemistry, coccidia, coccidia intestine, colic vomit, comm lacf, composition databank,
      conductivity, databank, database, deficiency selenium, diet, diet care, diet stenosis, drying, entomology food, fact food, fact sheet, feeding, food, food
      composition, food data, food history, food industry, food nutrient, food nutrition, food preservation, food resource, food safety, food science, food standard,
      food technology, food timeline, foodcomp, freeze, freezing, fruit, fruit vegetable, growing, haccp, health, home food, horticulture food, hyperthyroidism, lacf,
      low acid, multocida, mycotoxins, mycotoxin, myxomatosis, neuter preparation, nutrient database, nutrition, nutrition information, nutrition nourishment,
      organic, oxalates, pasteurella, pasteurilosis, phosphide, physiology nutrition, pickling, poison, preservation, preservation canning, quality, refractometer,
      resource food, science fact, science food, search food, selenium, standard, substance, technology food, timeline food, toxin, vegetable quality, water activity,
      water food
    </atomic_query>
    <atomic_query name="object" type="tables lists" weight="1" semantic_builder="disjunct">
      'f', 'c', 'anise', 'apple', 'ash', 'aw', 'bean', 'bran', 'brine', 'broth', 'ca', 'chop', 'cider', 'chive', 'cob', 'cod' liver, 'corn', 'cow', 'cr', 'cu', 'date', 'dill',
      'dish', 'dry', 'drying', 'fe', 'flan', 'flour', 'germ', 'gr', 'gram', 'grams', 'gruel', 'ham', 'ions', 'iron', 'item', 'kj', 'lamb', 'lime', 'list', 'loin', 'mg', 'ml',
      'moose', 'na', 'ne', 'nest', 'okra', 'pack', 'paste', 'paste', 'pate', 'per cent', 'per unit', 'per', 'pet' food, 'ph', 'pie', 'quince', 'range', 'raw', 'ripe', 'salt',
      'se', 'seed', 'slice', 'smelt', 'sour', 'soy', 'soya', 'steak', 'stew', 'tofu', 'tuna', 'unit', 'vine', 'zn', abalone, acid, acidified, acidity, acidophilus, acorn,
      activity, added, additive, adipose tissue, aged, air, alcohol, alcoholic drink, alkaline, aloe, aloe vera, alpha tokoferol, analytical, anchovies, anchovy, anise
      fennel, anon, asoc, approach, approximate, apricot, arrowroot, artichoke, ascorbic, asparagus, asparagus bud, asparagus rattan, atmosphere conditioning, avocados,
      baby soup, backbone, bacon, bacteria, bacteria yeast, bacterial, bacteriological, baked, bakery, bamboo, banana, bantam, barbecued, bartlett pear, based, basil, basil
      pesto, bass, batavia, bean, beat, bechamel, beef, beef fat, beef striploin, beet, belgian endive, beverage, biochemical, biochemical reaction, biotin, biscuit, bisque,
      blackberries, blackberry, blue veined cheese, blueberries, bluefish, boiled, bologna, bolognese, bolognese, brain, bread, bread rye, bread spoil, breadfruit,
      brisket, brix, broccoli, broccoli, broiled, broth, broth shallot, bruised, brussel sprout, btw, bud stalk, bulb, burger, butchery, butter, buttermilk, cabbage, cactus,
      cactus calamary, cactus cantaloupe, caesar, cake, cake mix, calamary, calcium, caloric, calorie, camembert, candied, cane sugar, canned, caper, caprine, caramel,
      carbohydrate, carbs, carcass, carotene, carp, carrot, category, cauliflower, caviar, celery, cephalopod, cereal, characteristic, carbohydrate, chard, chayote,
      ....
    </atomic_query>
  </combination>
</WeQueLQuery>
```

FIG. III.8 – La requête " Q_{Log} " de filtrage

Mesures de qualité : Dans cette expérimentation, nous avons mesuré la qualité du processus de filtrage par la qualité du classement de documents établi par `eDotHTMLFilter` à la suite de l'évaluation de la requête " Q_{num} " sur les documents de l'ensemble $\mathcal{D}_f$. Nous nous sommes ainsi intéressés à la corrélation entre la position d'un document d donné dans le précédent classement et la pertinence effective de son contenu. Pour cela, nous avons utilisé une courbe ROC et une courbe LIFT pour mesurer et représenter graphiquement cette corrélation.

Courbe ROC : La courbe ROC (*Receiver Operating Characteristics Curve*) a été introduite dans les années 50 pour représenter les signaux radio parasités par du bruit. Reprise récemment pour les besoins des systèmes de classification supervisée à base de classement de réponses, cette courbe sert à montrer, sur un classement établi, la variation entre la « *spécificité* » et la « *sensibilité* » d'un système d'apprentissage [Ferri et al. 2002].

La sensibilité d'un système de classification est définie comme étant le nombre de vrais positifs (documents réellement pertinents). Un vrai positif est un élément jugé positif par le système et jugé aussi comme réellement positif par un expert humain.

La spécificité d'un système de classification est définie par le nombre de faux positifs. Un faux positif est un élément jugé négatif par un expert humain mais qui a été jugé positif par le système.

Dans une représentation ROC, les faux positifs sont disposés sur l'axe des abscisses et les vrais positifs sont disposés sur l'axe des ordonnées.

Soient :

- " $CL = \langle x_1, x_2, \dots, x_{N-1}, x_N \rangle$ " le classement établi par un système de classification ;
- " FP " l'ensemble de tous les faux positifs du classement CL ;
- " VP " l'ensemble de tous les vrais positifs du classement CL .

La courbe ROC, notée " $\mathcal{F}_{ROC}$ ", est construite en parcourant le classement CL du premier élément x_1 au dernier élément x_N (remarquez que $N = |VP| + |FP|$). Le calcul de l'ordonnée " $\mathcal{F}_{ROC}(x_i)$ " de chaque élément x_i du classement du système est montré ci-dessous :

$$\begin{aligned}
 \mathcal{F}_{ROC}(x_1) &= 0 && \text{si } x_1 \in FP \\
 \mathcal{F}_{ROC}(x_1) &= \frac{1}{|VP|} && \text{si } x_1 \in VP \\
 \forall i > 0 & \\
 \mathcal{F}_{ROC}(x_i) &= \mathcal{F}_{ROC}(x_{i-1}) + \frac{1}{|VP|} && \text{si } x_i \in VP \\
 \mathcal{F}_{ROC}(x_i) &= \mathcal{F}_{ROC}(x_{i-1}) && \text{si } x_i \in FP \\
 \forall x : \quad x_{i-1} < x < x_i & \quad \mathcal{F}_{ROC}(x) = \mathcal{F}_{ROC}(x_{i-1})
 \end{aligned} \tag{III.1}$$

Remarque 5.2 Les éléments de l'ensemble " FP " sont répartis sur l'axe des abscisses de manière uniforme ie :

$$\begin{aligned}
 x_1 &= 0 \\
 \forall i > 1 \quad x_i &= x_{i-1} + \frac{1}{|FP|} \\
 x_{|FP|} &= 1
 \end{aligned}$$

Propriété : La courbe ROC est continue et croissante. Elle est bornée par la valeur 1 ie $\forall x \in VP \cup FP : \mathcal{F}_{ROC}(x) \leq 1$.

L'efficacité d'un système de classification est jugée à partir de l'aire définie entre l'axe des abscisses et la courbe ROC associée à ce système. Cette mesure, appelée l'aire sous la courbe ROC et notée " AUC " (*Area Under Curve*), est calculée par la formule suivante :

$$AUC = \sum_{i=1}^{|FP|-1} (x_{i+1} - x_i) \mathcal{F}_{ROC}(x_{i+1}) \tag{III.2}$$

Le tableau III.30 page 82 présente la qualité d'un système de classification à partir de son AUC.

Aire sous la courbe ROC	qualité	catégorie
$0.9 < AUC \leq 1$	Excellente	A
$0.8 < AUC \leq 0.9$	bonne	B
$0.7 < AUC \leq 0.8$	moyenne	C
$0.6 < AUC \leq 0.7$	médiocre	D
$0 \leq AUC \leq 0.6$	mauvaise	F

TAB. III.30 – Classification de la qualité d'un système à partir de l'aire sous sa courbe ROC

Remarque 5.3 *Un système idéal (de catégorie A) correspond à un système ayant un classement qui positionne tous les vrais positifs aux premières positions et tous les faux positifs à la suite des positions occupées par les vrais positifs. La courbe correspondante est montrée dans la figure III.9 page 82.*

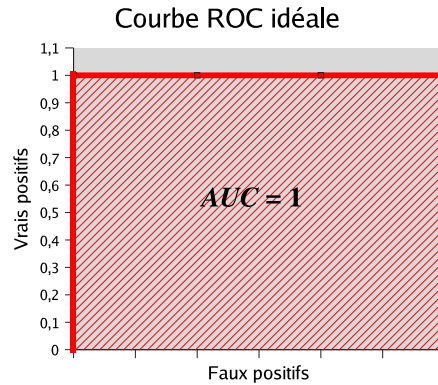


FIG. III.9 – La courbe ROC idéale

La nature de la requête " Q_{Log} " (évaluation binaire à 0 ou 1) ne permet pas d'établir un classement des documents de l'ensemble $\mathcal{D}_f$ à la suite de leurs évaluations et donc d'éditer une courbe ROC dans ce cas de figure. Pour comparer les deux requêtes " Q_{Log} " et " Q_{num} ", nous avons construit une courbe ROC pour la requête " Q_{Log} " à partir d'un classement de documents établi comme suit :

À partir de leurs scores binaires sur la requête " Q_{Log} ", nous avons divisé les documents de l'ensemble $\mathcal{D}_f$ en deux tranches. La première tranche couvre les premières positions (de la 1^{ère} à la $n^{\text{ème}}$) et regroupe tous les documents ayant obtenu un score d'évaluation égal à 1 sur la requête " Q_{Log} ". Plutôt que d'opter pour un ordre arbitraire entre les documents de cette tranche, nous avons réutilisé l'ordre suivant lequel le robot de Xyleme a découvert chacun des documents de cette tranche. Rappelons que la stratégie d'exploration du robot de Xyleme privilégie le rapatriement prioritaire des documents populaires. Par conséquent, si d_1 a été rapatrié plus tôt que d_2 , alors d_1 a une plus grande importance (PageRank) que d_2 . Ainsi, le classement de la première tranche représente les documents par ordre décroissant de popularité ce qui constitue un classement plus significatif qu'un ordre arbitraire entre les documents de cette tranche. La seconde tranche regroupe

le reste des documents (de la $n + 1^{\text{ère}}$ à la $|\mathcal{D}_f|^{\text{ème}}$) dont le score d'évaluation sur " Q_{Log} " est égal à 0. Les documents de cette tranche sont ordonnés suivant leur importance de la même manière que les documents de la première tranche.

À titre indicatif, nous avons créé une troisième courbe ROC à partir d'un classement virtuel alternant un vrai positif à la suite d'un faux positif (ou inversement). Un tel classement correspond à un filtrage aléatoire, de médiocre qualité, qui désigne aléatoirement les documents pertinents avec une probabilité de $\frac{1}{2}$ de se tromper dans la prédiction. La courbe ROC associée à ce filtrage virtuel est représentée par la médiane qui relie l'origine (0,0) au point (1,1) (courbe bleue de la figure III.11 page 85).

Remarque 5.4 *Vu le nombre important de documents filtrés (environ 2000), nous avons restreint le processus manuel de désignation des vrais et des faux positifs uniquement aux 100 premiers documents de chaque classement de filtrage.*

Courbe LIFT : La courbe d'élévation, que nous appelons également courbe LIFT, permet de mesurer l'évolution de la précision d'un système de classification en parcourant le classement établi depuis la première position et par pas d'une position à chaque fois. À une position " n " donnée dans le classement établi, la précision est définie comme étant la proportion de vrais positifs parmi les " n " premiers éléments du classement.

Les faux et les vrais positifs sont disposés sur l'axe des abscisses en respectant leur position dans le classement du système. La répartition de ces abscisses est uniforme, *i.e.* les abscisses sont séparées par pas de $\frac{1}{|VP|+|FP|}$. L'axe des ordonnées contient les précisions successives du système calculées en avançant d'une position dans le classement établi (donc par pas de $\frac{1}{|VP|+|FP|}$ sur l'axe des abscisses).

Soient :

- $CL = \langle x_1, x_2, \dots, x_N \rangle$ le classement établi par le système considéré ;
- $CL_{i,i \leq N} = \langle x_1, x_2, \dots, x_i \rangle$ le classement des i premiers éléments du classement CL .

La fonction " $\mathcal{F}_{LIFT}$ ", représentant la courbe LIFT, s'exprime comme suit :

$$\begin{aligned} \forall x_i \in CL \quad \mathcal{F}_{LIFT}(x_i) &= \frac{|\{(x \in X^i) \wedge (x \in VP)\}|}{|X^i|} \\ \forall x, x_i < x < x_{i+1} \quad \mathcal{F}_{LIFT}(x) &= \frac{\mathcal{F}_{LIFT}(x_i) - \mathcal{F}_{LIFT}(x_{i+1})}{x_i - x_{i+1}} \cdot x - \frac{\mathcal{F}_{LIFT}(x_{i+1}) \cdot x_i + \mathcal{F}_{LIFT}(x_i) \cdot x_{i+1}}{x_i - x_{i+1}} \end{aligned} \quad (III.3)$$

ie le point $(x, \mathcal{F}_{LIFT}(x))$ appartient à la droite qui passe par
le point $(x_i, \mathcal{F}_{LIFT}(x_i))$ et le point $(x_{i+1}, \mathcal{F}_{LIFT}(x_{i+1}))$

Remarque 5.5 *Une courbe LIFT idéale correspond à un système dont la précision reste égale à 1 sur tous les premiers éléments du classement et décroît ensuite progressivement sur le reste du classement jusqu'à atteindre la précision finale du système (voir figure III.10 page 84).*

À partir de leurs classements respectifs, nous avons construit deux courbes LIFT pour les requêtes " Q_{num} " et " Q_{Log} ".

Remarque 5.6 *Les deux mesures que nous avons retenues pour l'évaluation de la qualité du filtrage dépendent uniquement du classement établi par `eDotHTMLFilter`. Le nombre de vrais positifs collectés par le robot de Xyleme n'influe aucunement sur l'aspect général des courbes ROC et LIFT.*

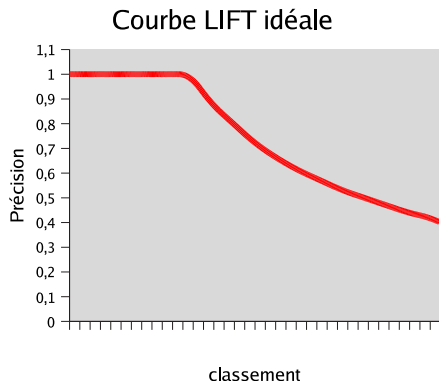


FIG. III.10 – La courbe LIFT idéale

5.2.2 Évaluation expérimentale du filtrage

Le robot explorateur de Xyleme a fourni à notre module 2037 urls correspondant à des documents « potentiellement » pertinents. `eDotHTMLFilter` a rapatrié les documents référencés par ces urls et évalué leurs contenus sur les deux requêtes " Q_{Log} ".

Sur la requête " Q_{Log} ", un seul document parmi les 2037 évalués a été évalué avec un score égal à 1. Ce document a été manuellement expertisé comme étant un faux positif (pertinent pour la requête " Q_{Log} " mais sans intérêt pour ce que nous recherchons pour l'entrepôt d'eDot). Néanmoins, le contenu de ce document était en rapport avec le risque alimentaire.

Sur la requête " Q_{Num} ", un seul document a obtenu un score très largement au dessus des scores de autres documents de l'ensemble $\mathcal{D}_f$. Les experts ont déterminé que ce document correspondait à un vrai positif. Précisons que ce document a été évalué avec un score nul sur la requête " Q_{Log} ", ce qui montre l'intérêt du score numérique de la requête " Q_{Num} ".

Nous avons, par ailleurs, manuellement expertisé le contenu des $2037 - 100 = 1937$ documents de l'ensemble $\mathcal{D}_f$ afin de déterminer les éventuels vrais positifs ayant pu échapper au classement de notre système (rappelons que ce classement est limité aux 100 premières positions seulement). Nous n'avons pas trouvé de documents réellement pertinents dans les 1937 documents expertisés, ce qui montre la précision des requêtes du langage WeQueL.

Toutefois, nous estimons que les résultats précédents (un seul document réellement pertinent sur les 2037 proposés au filtrage) sont insuffisants et non fiables pour valider notre approche de filtrage. De plus l'intérêt du classement numérique de la requête " Q_{Num} " n'est pas perceptible sur un seul vrai positif dans tout le classement. C'est ainsi que nous avons décidé d'injecter 21 nouveaux documents dans l'ensemble $\mathcal{D}_f$ (ce qui fait au total $2037 + 21 = 2058$ documents à filtrer). Recherchés manuellement par nos experts en utilisant Google, ces nouveaux documents se répartissent comme suit : 13 documents pertinents, huit documents en rapport avec le risque alimentaire mais ne correspondant pas aux documents souhaités et un dernier document n'ayant aucun rapport avec le risque alimentaire.

Nous avons relancer notre processus de filtrage (une fois avec " Q_{Num} " et une autre fois avec " Q_{log} ") en prenant en compte ces nouveaux documents. Les scores calculés nous ont permis de réajuster les

classements précédents. Nous avons observé une certaine conformité entre la pertinence des documents évalués sur la requête " Q_{Num} " et les positions qu'ils occupent dans le classement. En effet, sur les 14 vrais positifs que comporte le nouveau lot filtré, nous avons retrouvé neuf ($\frac{9}{14} \approx 64\%$) vrais positifs classés parmi les 10 premiers (soit 90% des premières positions). Les cinq vrais positifs restants se sont relativement bien classés en s'étalant entre la 11^{ème} position et la 57^{ème} position. Nous avons également remarqué que les documents en rapport avec le risque alimentaire ont occupé les autres positions du classement créant un tampon séparant les vrais positifs des vrais négatifs (document sans aucun rapport avec le risque alimentaire). Partant de ce découpage, nous avons deux possibilités pour fixer le seuil de pertinence pour l'entrepôt d'eDot : le plus petit score des neuf premières positions (c'est-à-dire 0,14 pour privilégier la précision) ou le score du dernier vrai positif du classement (c'est-à-dire $\approx 0,08$ pour privilégier la couverture).

Pour le classement établi à partir de la requête " Q_{log} ", les résultats obtenus ont été médiocres et quasi-aléatoires. Cet échec s'explique par l'insuffisance de l'interprétation logique par un système de score binaire associée à " Q_{log} " pour distinguer finement les vrais positifs des autres documents.

La corrélation entre la pertinence par rapport au risque alimentaire et le classement établi par chaque requête de filtrage est représentée par les deux courbes ROC de la figure III.11 page 85.

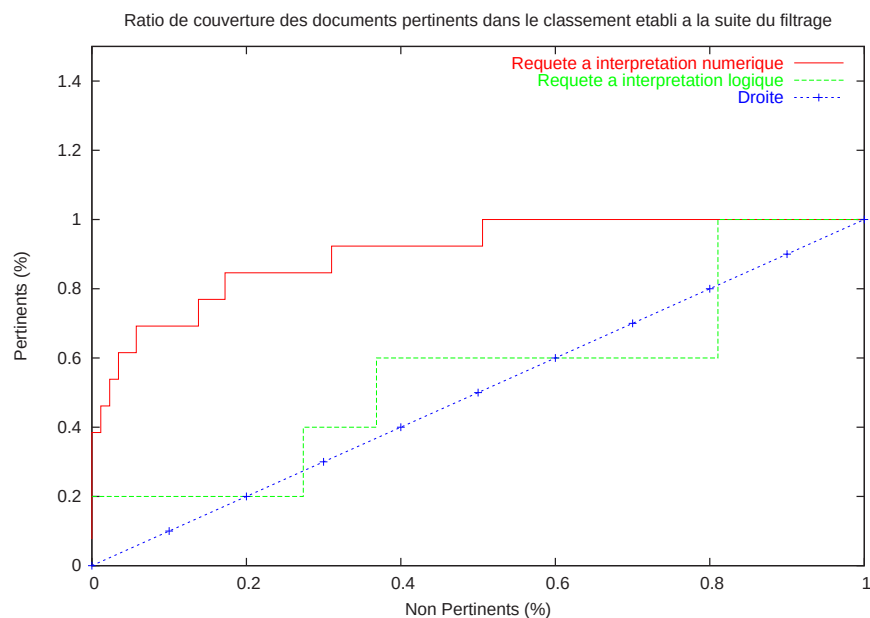


FIG. III.11 – Résultats expérimentaux du filtrage : Courbe ROC

En s'appuyant sur la formule de calcul de l'aire sous une courbe ROC (voir formule III.2 page 81), l'aire de la courbe ROC associée :

- à la requête " Q_{Num} " (courbe rouge) est égale à "0,9036" ;
- à la requête " Q_{Log} " (courbe verte) est égale à "0,5474" ;
- Au filtrage virtuel (courbe bleue) est égale à "0,5".

Conformément à la classification proposée dans le tableau III.30 page 82, le filtrage utilisant la requête " Q_{Num} " appartient à la classe A. Le filtrage utilisant la requête " Q_{Log} " appartient à la même classe que le filtrage virtuel c'est à dire à la classe F. Ceci montre le gain significatif en

efficacité et en précision d'une requête WeQueL produisant un score numérique.

La figure III.12 de la page 86 montre les deux courbes LIFT construites pour représenter l'évolution de la précision du module de filtrage `eDotHTMLFilter`.

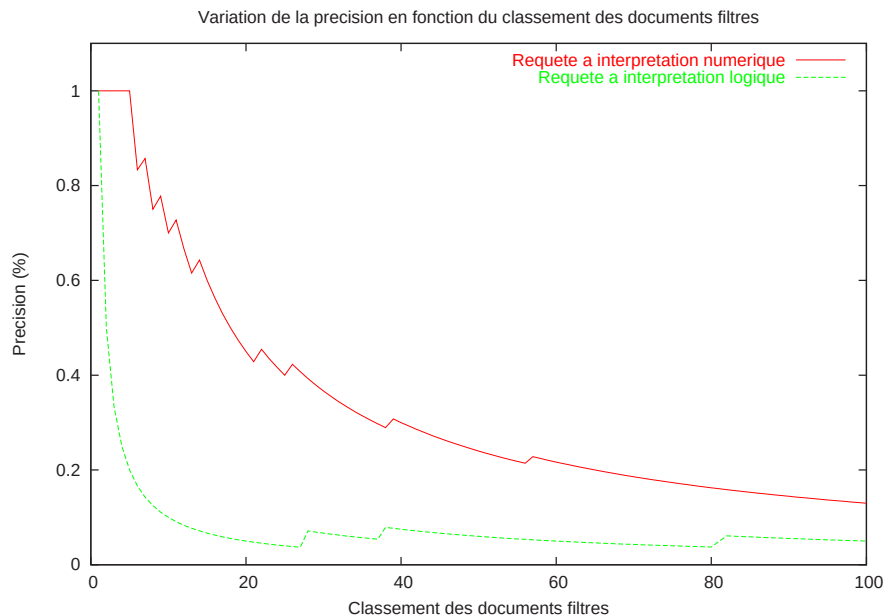


FIG. III.12 – Résultats expérimentaux du filtrage : Courbe LIFT

Les courbes LIFT associées aux deux requêtes de filtrage confirment les résultats obtenus à la suite de l'analyse des courbes ROC de chacune des deux requêtes précédentes. En effet, la courbe LIFT du filtrage par la requête " Q_{Num} " (courbe rouge) est constamment au-dessus de la courbe LIFT du filtrage par la requête " Q_{log} " (courbe verte). Ceci signifie simplement que la précision de la requête " Q_{Num} " est supérieure à la précision de la requête " Q_{Log} ".

6 Conclusion

Dans ce chapitre, nous avons présenté le langage WeQueL, un langage dont le principe consiste à combiner plusieurs critères de description à la fois où chaque critère permet de spécifier un ensemble de mots clés à cibler sur une section particulière des documents à évaluer. Par ce principe, nous avons contribué à enrichir et accroître le pouvoir expressif des requêtes mots clés. En effet, nous proposons dans le langage WeQueL un panel varié de requêtes atomiques et un système de combinaison de ces requêtes atomiques qui permettent aux utilisateurs de diversifier leur description des documents souhaités dans une même requête. En complément à cela, nous proposons également différents constructeurs sémantiques qui permettent aux utilisateurs de contrôler l'interprétation et l'évaluation de leurs requêtes de manière complètement transparente et déclarative.

Nous avons montré, dans la section expérimentale de ce chapitre, que pour rechercher des documents thématiques, les requêtes du langage WeQueL sont trois fois plus précises que des requêtes comparables sous Google. La couverture des requêtes WeQueL testées demeure cependant insuffisante pour inclure tous les documents pertinents contenus dans l'ensemble des documents évalués.

Dans le contexte du projet eDot, nous avons utilisé le langage WeQueL pour filtrer des documents rapatriés du web. Conformément à l'expertise établie par nos experts sur le risque alimentaire, nous avons conçu une requête de filtrage qui permet de cibler les termes de l'ontologie eDot sur les tables et listes HTML. Pour booster les performances de cette requête, nous avons choisi d'associer l'interprétation des valeurs de la requête de filtrage à des constructeurs numériques. Ceci nous a permis d'établir un classement des documents évalués qui s'est avéré fort utile pour déterminer le seuil de pertinence des documents ciblés par le projet eDot.

Dans l'expérimentation de validation de notre approche de filtrage, nous avons confirmé notre intuition sur la supériorité d'une requête produisant un score numérique sur une requête équivalente produisant un score booléen.

Deuxième partie

Exploration du web

Robots explorateurs du web

Dans ce chapitre, nous présentons le domaine de l'exploration du web à la recherche de documents ainsi que l'architecture et le fonctionnement d'un robot explorateur du web.

1 Définitions & Formalisation

1.1 Le web

Dans cette thèse, nous considérons le web comme étant une immense source de données en ligne qu'il est possible de consulter par navigation manuelle ou par exploration automatique. D'autre part, le web se caractérise par l'aspect hétérogène de ses données : texte, image, son, video, tables *etc.*

Les différentes données du web sont organisées sous forme d'unités d'informations appelées *pages ou documents web*. Ces documents sont repérés de manière unique et sans ambiguïté par un identifiant appelé *une url (Uniform Resource Locator)*. Une url [Berners-Lee et al. 1994] est un format de nommage universel, sous forme d'une chaîne de caractères ASCII, pour désigner un document sur le web. Dans la suite de ce chapitre, nous adoptons la notation $url(d)$ pour désigner l'url associée au document d .

1.2 Liens hypertextes

Les documents du web sont interconnectés par des liens appelés *liens hypertextes*. Ces liens permettent à un utilisateur ou à un programme automatique d'atteindre le contenu d'autres documents à partir d'un ou plusieurs documents donnés.

Un lien hypertexte est défini dans un document **HTML** par la balise `<a>` pour référencer un autre document. L'url de ce dernier document est mentionnée dans la valeur de l'attribut `href` de la balise `<a>`. Une description textuelle peut être ajoutée à un lien hypertexte en incluant une liste de mots entre la balise ouvrante et la balise fermante qui définissent ce lien. Cette liste de mots est appelée

ancre d'un lien.

En pratique, l'ancre d'un lien hypertexte constitue une information capitale dans la description du document pointé. En effet, la majorité des moteurs de recherche actuels utilisent l'ancre des liens pour ajouter une description supplémentaire aux contenus des documents qu'ils indexent.

Un lien hypertexte est à la fois entrant et sortant. Il est entrant par rapport au document vers lequel il pointe et est sortant par rapport au document depuis lequel il pointe. Dans l'exemple de la figure IV.1 page 92, le lien l est sortant par rapport au document « Doc_1 ». l est, par contre, entrant pour le document « Doc_2 ».

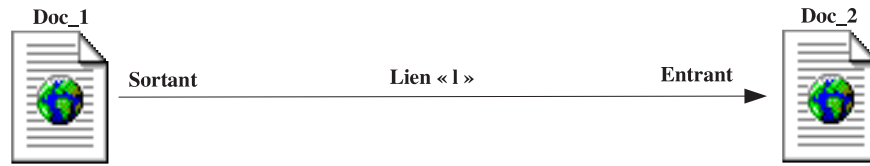


FIG. IV.1 – Lien entrant / sortant

Notation : Nous représentons un lien l sortant d'un document d et entrant vers un autre document d' par la notation suivante :

$$d \xrightarrow{l} d'$$

Remarque 1.1 *Tous les documents du web ne sont pas nécessairement atteignables par des liens hypertextes. En effet, un bon nombre de documents sont générés à la suite de la soumission d'un formulaire web ou d'une requête particulière à un serveur web. D'autres nécessitent tout simplement un mot de passe pour être consultés. Ces documents sont connus pour faire partie du web invisible (Deep Web). Dans le contexte de cette thèse, nous ne nous intéressons pas à cette partie du web.*

Définition 1 (Parents d'un document) *Le document d' est dit **parent** du document d si et seulement s'il existe un lien l sortant de d' pour pointer sur d . L'ensemble $A(d)$ des parents d'un document d est alors défini comme suit :*

$$A(d) = \{d' / \exists l \text{ un lien tel que : } d' \xrightarrow{l} d\}$$

Définition 2 (Frères d'un document) *Le document d' est dit **frère** du document d si et seulement s'il existe un parent d'' commun à d' et d . L'ensemble $S(d)$ des frères d'un document d est alors défini comme suit :*

$$S(d) = \{d' / \exists d'' \text{ un document tel que : } d'' \in A(d)\}$$

Définition 3 (Voisinage d'un document) *Le voisinage d'un document d , noté $V(d)$, est défini comme étant l'ensemble des frères et des parents de d .*

$$V(d) = A(d) \cup S(d)$$

La notion de voisinage (parents + frères) d'un document web est schématisée dans la figure IV.2 page 93.

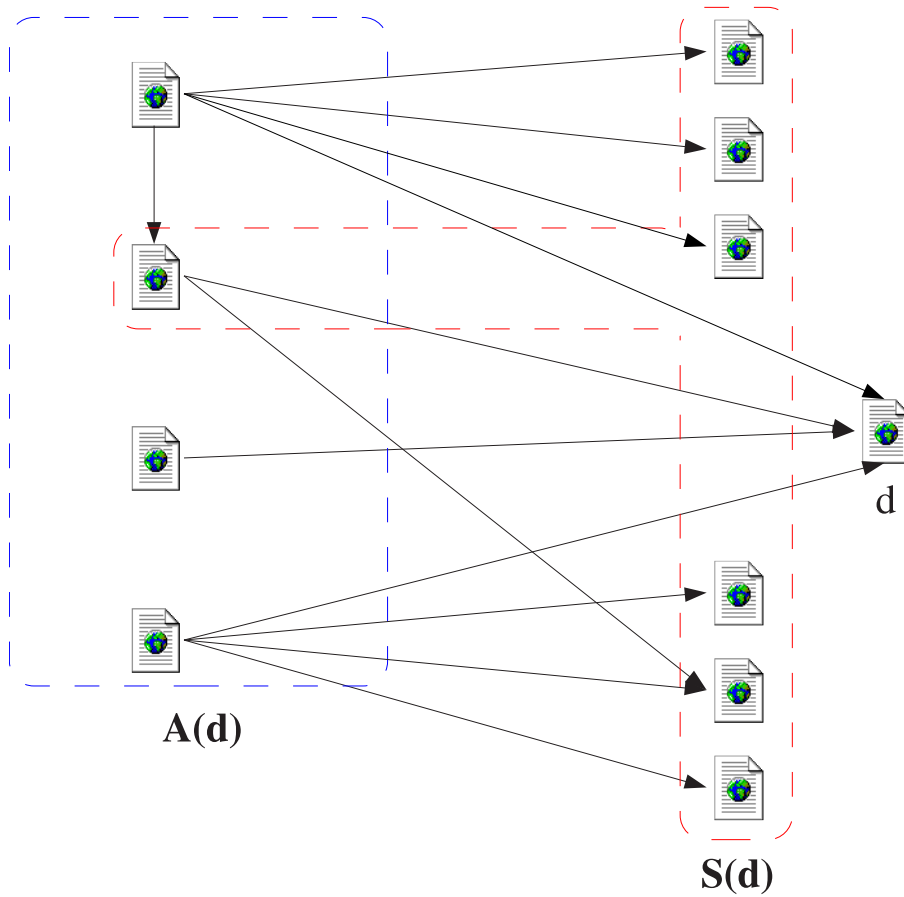


FIG. IV.2 – Voisinage d'un document

1.3 Graphe du web

La notion de navigation entre documents via les liens hypertextes permet de modéliser le web sous forme d'un graphe orienté que nous notons $\mathcal{G}(\mathcal{D}, \mathcal{L})$. L'ensemble $\mathcal{D}$ des sommets du graphe $\mathcal{G}(\mathcal{D}, \mathcal{L})$ correspond à l'ensemble de tous les documents contenus dans le web. L'ensemble $\mathcal{L}$ des arcs du graphe $\mathcal{G}(\mathcal{D}, \mathcal{L})$ correspond à l'ensemble de tous les liens hypertextes qui existent entre les documents de l'ensemble $\mathcal{D}$.

2 Exploration automatique du web

L'exploration automatique du web, par opposition à l'exploration humaine et manuelle du web, peut être définie comme étant un processus autonome capable de visiter et de consulter récursivement le contenu des documents du web dans un but précis. Ce but peut être trivial en consistant, par exemple, à récolter tout simplement le maximum de documents web à des fins d'indexation (moteurs de recherche généralistes). Il peut également être plus élaboré et donc plus compliqué à réaliser lorsqu'il s'agit, par exemple, de rechercher des documents spécifiques traitant d'un domaine particulier (moteurs de recherche thématiques).

Lorsque le but d'une exploration est réalisé, on parle de l'accomplissement d'un *cycle d'explora-*

tion. Dès lors, Il est possible de relancer d'autres cycles d'exploration afin de prendre en compte les éventuels changements survenus dans le web (mise à jour du contenu des documents déjà visités, découverte de nouveaux documents, suppression de documents, *etc.*).

L'accomplissement d'un seul cycle d'exploration du web est devenu une tâche délicate soumise à de fortes contraintes. En effet, les aspects techniques du web (trafic réseau et bande passante par exemple) constituent des facteurs pénalisants dans le rapatriement de documents. De plus, la taille actuelle du web (en nombre de documents et de liens) contrarie lourdement la progression du robot dans le web en prolongeant significativement le temps nécessaire à l'achèvement d'un cycle d'exploration. La croissance du web (ajout et suppression de documents) n'arrangent en rien la situation. Il est donc devenu difficile de mettre en œuvre des explorations ayant pour but de visiter la totalité des documents du web.

La mise en œuvre d'un processus d'exploration du web nécessite l'emploi de programmes spécifiques, spécialement conçus à cet effet, appelés *robots explorateurs* (*crawlers ou spiders en anglais*).

2.1 Robot explorateur du web

Un robot explorateur du web est un programme chargé de mettre en œuvre le processus automatique d'exploration du web. De plus en plus d'applications font appel à ce type de programmes pour satisfaire leur besoin en documents et données en provenance du web. La conception d'un robot explorateur est complexe et exige de ses concepteurs une parfaite maîtrise des aspects et technologies liés web, comme par exemple :

- les techniques de fouille des documents HTML pour rechercher et extraire par exemple les prochaines urls à suivre ;
- les communications client/serveur par le protocole HTTP [Fielding et al. 1997] pour le rapatriement de documents web ;
- le protocole TCP/IP [Socolofsky et Kale 1991] pour la gestion des aspects réseaux du web (résolution DNS des urls, *etc.*) ;
- les règles de bonne conduite du protocole SRE (*Standard Robot Exclusion protocol*) [Martijn Koster 1996] pour éviter la surcharge abusive de certains serveurs ou l'accès inutile à des documents sans intérêt ;

La qualité et l'efficacité d'un robot sont jugées par rapport à deux catégories de critères. La première catégorie de critères mesure les performances purement techniques du robot, à savoir : le temps moyen de rapatriement d'un document, le nombre moyen de documents rapatriés par unité de temps, la bande passante consommée, *etc.* La seconde catégorie de critères de qualité d'un robot s'intéresse à la mesure du taux de réalisation et d'accomplissement des objectifs assignés à l'exploration. Ces mesures permettent de savoir, par exemple, si le robot a rapatrié en quantité suffisante les documents souhaités (calcul de la précision et de la couverture) ou de quantifier la qualité des documents rapatriés (calcul de la popularité des documents rapatriés, calcul d'un score de pertinence de chaque document rapatrié, *etc.*).

2.2 Architecture d'un robot explorateur du web

Un robot explorateur se décompose en un ensemble de composants logiciels indépendants dont le fonctionnement est souvent parallélisé. Pour simplifier cette architecture, nous avons regroupé

ces composants en quatre modules fonctionnels distincts : la pile d'urls en attente de traitement, le module de d'application de la stratégie d'exploration, le module de rapatriement de documents à partir du web et le module de stockage et exploitation des ressources rapatriées (voir figure IV.3 page 95).

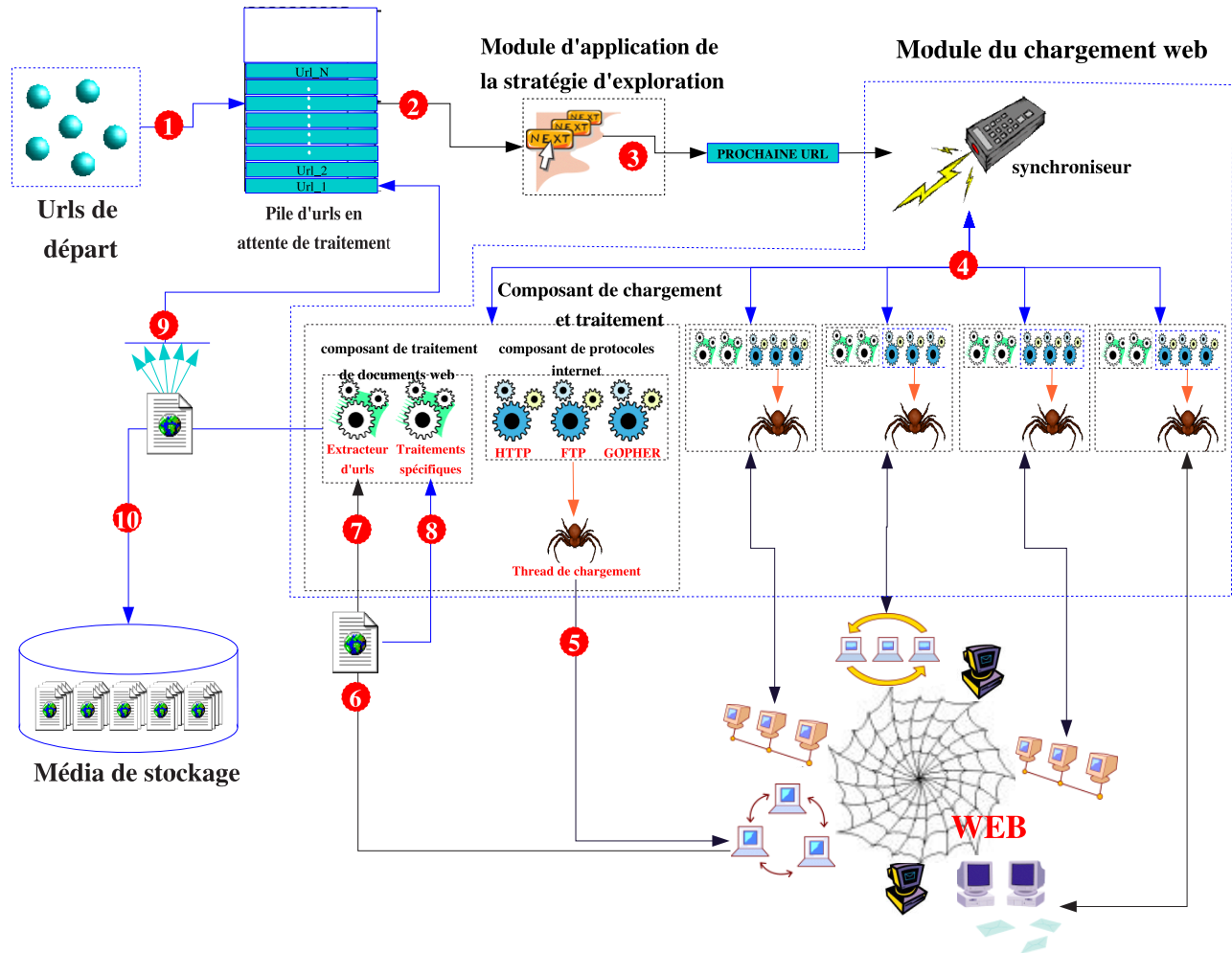


FIG. IV.3 – Architecture et fonctionnement général d'un robot explorateur du web

2.2.1 La pile d'urls en attente de traitement

Au cours de son exploration du web, un robot explorateur cumule, au fur et à mesure, un grand nombre d'urls classées en deux catégories. La première catégorie d'urls correspond aux urls déjà visitées et traitées par le robot. La seconde catégorie d'urls regroupe les urls qui identifient les documents atteignables par le robot avec un seul lien depuis un document déjà rapatrié. En attendant leur traitement (rapatriement et autres), ces urls sont stockées dans une structure en mémoire centrale que nous appelons «*la pile d'urls*». Cette pile constitue la structure centrale sur laquelle repose le fonctionnement des autres modules du robot explorateur du web.

La pile d'urls peut être vue comme étant une file d'attente dans laquelle le robot stocke tempo-

rairement les urls fraîchement découvertes dans les balises `<a href>` du contenu HTML des documents rapatriés. Les urls de cette file d'attente sont généralement classées par ordre de priorité de traitement. L'url jugée « importante et prioritaire » se retrouve en tête de la file d'attente. L'ordre décroissant de priorité est respecté dans l'insertion des autres urls en attente de traitement. Signalons, par ailleurs, que la capacité de stockage de la pile d'urls est limitée (principalement par la capacité de la mémoire centrale dédiée au robot). Pour optimiser l'espace mémoire utilisé, toute url traitée, c'est-à-dire rapatriée et analysée, est aussitôt retirée de la pile.

Documents candidats Les urls contenues dans la pile d'urls référencent les documents accessibles depuis un document déjà rapatrié par le robot et cela en suivant un seul lien. Nous appelons ces documents *documents candidats* car ils constituent des candidats potentiels à l'exploration. Plus formellement, un document candidat est un document non encore visité par le robot mais ayant un ou plusieurs liens entrants provenant immédiatement de documents déjà visités. À tout moment t de l'exploration d'un robot, le web peut être partitionné en trois ensembles disjoints : l'ensemble des documents visités jusqu'au moment t , l'ensemble des documents candidats également appelé *la frontière du robot* et le reste des documents web. Cette partition est schématisée dans la figure IV.4 page 96.

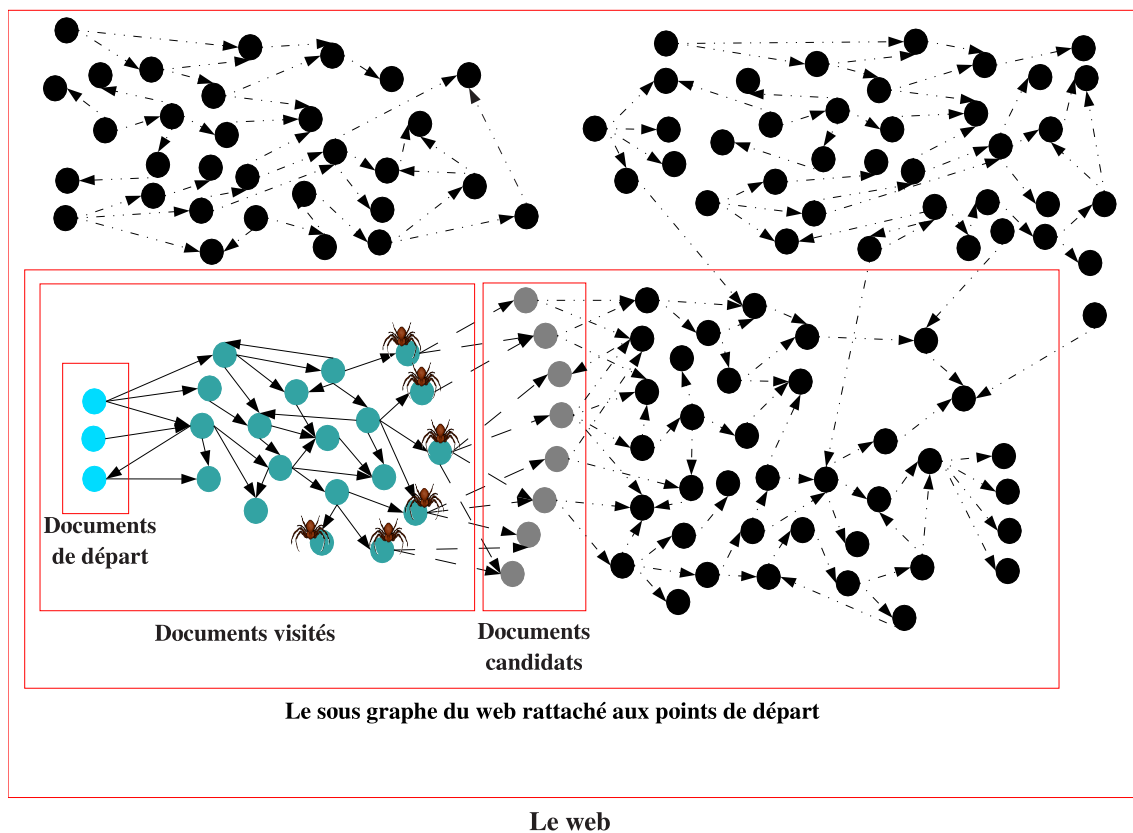


FIG. IV.4 – La partition du web engendrée par un robot explorateur

2.2.2 Le module d'application de la stratégie d'exploration

La taille de la pile d'urls fluctue au rythme des opérations d'ajouts d'urls extraites des balises HTML des documents rapatriés et des opérations de retraits d'urls traitées. La taille actuelle du web en documents HTML et le nombre important de liens hypertextes dans certains documents font que, lors d'une exploration web, les opérations d'ajouts de nouvelles urls sont nettement plus nombreuses que les opérations de retraits d'urls à traiter. Ce déséquilibre entre les urls en attente de traitement et les urls traitées conduit inéluctablement au dépassement de la capacité de la pile.

D'autre part, la désignation de la prochaine url à traiter parmi celles qui sont en attente est un problème délicat. En effet, il est difficile de juger a priori la pertinence d'un document non encore visité à partir seulement de son url dans la pile d'urls. Pour cette raison, un robot explorateur est couplé à une stratégie d'exploration qui lui permet de guider son parcours dans le web. Ce couplage est réalisé par l'intégration d'un module d'application de la stratégie choisie dans l'architecture logicielle du robot. Ce module fixe les priorités de traitement des urls de la pile du robot.

Pour garantir une exploration conforme à la stratégie adoptée, seul le module d'application de la stratégie d'exploration est autorisé à manipuler et à modifier l'ordonnancement des éléments de la pile d'urls du robot. De plus, ce module doit être l'unique point d'accès à la pile d'urls. Autrement dit, toute sollicitation de la pile d'urls doit être interceptée par le module d'application de la stratégie afin d'en vérifier la conformité avec la stratégie adoptée. Par exemple, le module d'application de la stratégie d'exploration est le seul module habilité à communiquer au robot les urls des documents à rapatrier.

La mise en place d'une stratégie d'exploration implique une réorganisation permanente des éléments de la file d'attente de la pile d'urls (classement décroissant des éléments de la pile). Cette réorganisation permet au robot de rapatrier les documents en observant l'ordre de priorité fixé par la stratégie adoptée.

2.2.3 Le module de rapatriement de documents

Le module de rapatriement de documents à partir du web est le module ayant pour objectif de charger et traiter le contenu des documents référencés par les urls communiquées par le module d'application de la stratégie d'exploration. Ce module se compose essentiellement d'un groupe d'agents (*threads*) de téléchargement web lancés généralement en parallèle dans un souci d'optimisation de performances.

Un agent de rapatriement (*web loader*) comporte deux composants fonctionnels : un composant de protocole et un composant d'analyse et de traitement des documents téléchargés.

Le composant de protocole d'un agent s'occupe de la sélection de la procédure appropriée au téléchargement d'un document référencé par une url spécifiant un protocole précis (*http*, *ftp*, *etc.*). Par exemple, l'agent responsable du téléchargement du document référencé par l'url suivante : <ftp://ftp.lri.fr/LRI/soft/cime/Release1/cime-1.05.tar.gz> sélectionne la procédure de téléchargement prévue pour une communication distante en *ftp* [Postel et Reynolds 1985].

Le second composant d'un agent de rapatriement s'occupe du traitement du contenu des documents rencontrés au cours de l'exploration. Pour les documents HTML, le traitement consiste à

analyser leur contenu afin d’extraire les références (urls) contenues dans les liens hypertextes. Ces références sont soumises au module d’application de la stratégie d’exploration pour qu’il se charge du calcul de leur priorité et de leur insertion dans la pile d’urls.

2.2.4 Le module de stockage des ressources rapatriées

Le module de stockage des ressources rapatriées permet de sauvegarder les documents web visités par un robot explorateur. Ce module est souvent commun à d’autres applications qui, pour leurs propres besoins, exploitent et réutilisent les résultats de l’exploration web d’un robot. Par exemple, les moteurs de recherche exploitent le contenu d’un ou de plusieurs index de documents web pour répondre aux requêtes mots clés de leurs utilisateurs. Ces index web sont alimentés régulièrement par les documents rapatriés par un ou plusieurs robots. Le format de stockage d’un document web est souvent propre aux applications web qui l’utilisent. Il existe, néanmoins, des informations primordiales dans le stockage d’un document web : l’url de ce document, son titre, sa date de rapatriement, son type, *etc.*

2.3 Le graphe web exploré par un robot

Nous définissons le graphe web exploré par un robot comme étant le graphe orienté formé par les documents visités par le robot explorateur et les liens entre ces documents. Nous notons ce graphe par $\mathcal{G}_{ex}(\mathcal{D}_{ex}, \mathcal{L}_{ex})$ où :

- $\mathcal{D}_{ex}$ est l’ensemble des sommets de ce graphe. Cet ensemble comprend la totalité des documents visités et rapatriés par le robot ;
- $\mathcal{L}_{ex}$ est l’ensemble des arcs du graphe exploré. Il comprend tous les liens reliant les documents de l’ensemble $\mathcal{D}_{ex}$.

2.4 Profondeur d’un document dans le graphe du web

Dans notre modélisation du web par un graphe orienté, la profondeur d’un document est définie par le nombre de liens contenus dans le plus petit chemin menant d’un point de départ du robot vers le document en question.

Exemple : Soient d_0 un document de départ pour une exploration et (d_1, d_2) deux documents tels que : $d_0 \xrightarrow{l_0} d_1$ et $d_1 \xrightarrow{l_1} d_2$. La profondeur de ces documents est :

document	profondeur
d_0	0
d_1	1
d_2	2

TAB. IV.1 – Exemple illustratif de la profondeur du document dans le graphe exploré par un robot

3 Déroulement d’une exploration du web

Comme tout processus, l’exploration du web débute par une phase d’initialisation durant laquelle le robot utilisé installe et initialise ses propres composants. Le robot vérifie également durant cette phase les paramètres indispensables à son fonctionnement tels que les connexions réseaux au

web, l'espace mémoire du système, *etc.* Une fois ces vérifications terminées, le robot initialise le contenu de sa pile d'urls avec une liste d'urls de départ appelées également les points de départ.

Les urls contenues dans la liste de départ sont manuellement choisies par l'administrateur du robot. Ces urls représentent les points d'entrée du robot dans le web pour démarrer son processus d'exploration. Pour certaines explorations, le choix des urls de départ peut se révéler crucial dans la réalisation des objectifs attendus. Par exemple, pour une exploration à la recherche de documents d'articles d'actualités, un robot lancé à partir de la page d'accueil de Yahoo ! [Yahoo ! site] possède théoriquement plus de chances de récolter rapidement ces documents qu'un robot lancé à partir d'une page d'accueil d'un site de recettes de cuisine ! La phase d'initialisation qui consiste à charger des urls de départ est schématisée par l'étape ❶ de la figure IV.3 page 95.

Le module d'application de la stratégie d'exploration entre ensuite en œuvre pour déterminer, si cela est possible, les priorités de rapatriement des urls de départ. Les éléments de la file d'attente de la pile d'urls sont classés et ordonnés conformément à la stratégie adoptée. Cette étape de réorganisation de l'ordre des éléments de la file d'attente de la pile d'urls est schématisée par l'étape ❷ de la figure IV.3 page 95.

À ce stade, l'exploration du web peut enfin démarrer. Pour cela, le robot crée un certain nombre d'agents de rapatriement dédiés au téléchargement des candidats référencés par les urls de la pile du robot. Les agents créés se mettent en situation de concurrence pour l'obtention de l'url du prochain document à télécharger. Afin de contrôler et synchroniser cette concurrence, le composant de synchronisation joue le rôle d'intermédiaire entre les agents de rapatriement et le module de stratégie. Il applique pour cela la politique du « *premier venu, premier servi (FIFO)* » pour satisfaire les demandes des agents de rapatriement. Avec cette politique, le composant de synchronisation garantit qu'un même candidat n'est rapatrié que par un seul agent et par conséquent plusieurs candidats différents peuvent être rapatriés en parallèle. Les demandes d'urls à traiter faites par chaque agent sont schématisées par les étapes ❸-❹ de la figure IV.3 page 95.

Ayant obtenu une url à traiter, un agent de rapatriement entame immédiatement le téléchargement du document candidat correspondant à cette url. Cette url est analysée pour extraire : le protocole (la procédure) de communication à utiliser, le serveur web à contacter, le chemin d'accès au document sollicité. Le téléchargement des documents candidats est schématisé par les étapes ❺-❻ de la figure IV.3 page 95.

Tout document rapatrié par un agent est ensuite traité, analysé et stocké par un composant spécialement dédié à cette tâche. Le traitement d'un document HTML consiste à extraire les urls référencées dans les attributs `href` du code HTML du document. Les nouvelles urls trouvées sont immédiatement réinjectées dans la file d'attente de pile pour être à leur tour traitées ultérieurement. Les opérations d'analyse, de traitement et de stockage de documents sont représentées par les étapes ❼-❽-❾ et ❿ de la figure IV.3 page 95.

La boucle d'étapes allant de ❷ à ❿ est répétée par le robot jusqu'à la fin du processus d'exploration. Cette répétition engendre une propagation progressive des agents du robot dans les nœuds du graphe du web. Nous appellerons cette boucle d'étapes une *itération du robot*.

La fin d'un processus d'exploration est déterminée par la satisfaction d'une ou plusieurs condi-

tions appelées «*conditions d'arrêt d'un robot*». Une condition d'arrêt d'un robot correspond à un événement donné dont l'occurrence empêche ou entrave la propagation des agents du robot dans le graphe du web. Nous avons classé les conditions d'arrêt d'un robot en deux catégories : les conditions inhérentes à la stratégie d'exploration et les conditions engendrées par la topologie du graphe $\mathcal{G}$ du web.

La première catégorie regroupe les conditions d'arrêts prédéfinies dans le module d'application de la stratégie d'exploration du robot. Par exemple, pour une exploration web avec une stratégie qui consiste à borner le nombre de documents visités par une limite donnée, l'atteinte de cette limite correspond à la condition d'arrêt.

Les conditions d'arrêt de la seconde catégorie se produisent lorsque tous les agents d'exploration du robot ont atteint une partie de la limite du graphe du web. Cette limite est symbolisée par des documents n'ayant aucun successeur dans le graphe du web (pas de liens sortants). Au niveau interne du robot, cette situation se traduit par une file d'attente d'urls vide, ce qui n'est pas forcément le cas dans la première catégorie d'arrêt du robot.

En réponse à une ou plusieurs conditions d'arrêts satisfaites, le module d'application de stratégie du robot refuse de communiquer les urls des candidats restants. Le module de synchronisation signifie aux agents de rapatriement la fin de leur mission en mettant un terme à leur existence.

4 Conclusion

Dans ce chapitre, nous avons introduit la technologie de l'exploration du web à la recherche de documents. Pour cela, nous avons présenté l'architecture et le fonctionnement d'un robot explorateur du web. Nous avons également présenté les formalismes et les algorithmes en rapport avec le déroulement d'un processus d'exploration web.

La recherche de documents par exploration du web est une tâche relativement lourde à mettre en place mais techniquement réalisable. Le marché informatique actuel propose des machines suffisamment puissantes pour satisfaire les exigences technologiques d'une telle tâche (vitesse de téléchargement, répartition de la bande passante, puissance de calcul du processeur pour traiter un document, gestion du parallélisme, *etc.*).

Cependant, la puissance purement technique du matériel informatique utilisé ne suffit pas, à elle seule, à assurer au robot des performances acceptables pour faire face à l'énorme quantité d'informations contenues dans le web. En effet, le traitement de plus de 8 milliards de documents web, recensés par Google [Google site], nécessite forcément une stratégie élaborée. Le rendement d'un processus d'exploration est étroitement lié à l'efficacité et la pertinence de la stratégie adoptée.

En conclusion, les stratégies d'exploration du web constituent des éléments significatifs dans la distinction entre les différentes techniques d'exploration web. Dans le chapitre suivant, nous présentons les stratégies d'exploration web abordées dans la littérature scientifique.

État de l'art

Dans ce chapitre, nous passons en revue les différentes techniques d'exploration du web abordées dans la littérature scientifique. Nous avons montré dans le précédent chapitre (voir chapitre IV page 101) que la stratégie d'exploration permettait de différencier les techniques d'exploration du web. Partant de ce constat, nous avons divisé les techniques d'exploration du web en trois catégories : les techniques d'exploration systématique du web, les techniques d'exploration reposant sur une stratégie de type « le meilleur d'abord » et les techniques d'exploration sélectives du web guidées par des critères thématiques.

1 Exploration systématique du web

Une exploration systématique, également appelée exploration exhaustive ou *general-purpose crawling*, est une exploration ayant pour but de rapatrier le plus possible de documents du web sans aucune distinction particulière. Idéalement, le graphe web rattaché aux points de départ est entièrement visité à la fin du processus d'exploration.

Pour mettre en place une telle approche, le robot explorateur utilise une stratégie dite «*stratégie non informée*». Une stratégie non informée est une stratégie qui s'appuie exclusivement sur la structure et la topologie du graphe du web pour déterminer l'ordre de parcours des documents candidats de la file d'attente du robot. L'aspect sémantique provenant du contenu des documents déjà rapatriés ou des liens hypertextes pointant sur les candidats est ignoré dans l'orientation des agents de rapatriement du robot.

Il existe quatre stratégies non informées pour une exploration systématique du web : stratégie aléatoire, stratégie en largeur d'abord, stratégie en profondeur d'abord et stratégie fondée sur la notion de popularité. Ces quatre stratégies sont essentiellement utilisées dans les robots explorateurs des moteurs de recherche actuels. Les moteurs de recherche communiquent très peu sur les stratégies d'exploration de leurs robots explorateurs.

1.1 Exploration systématique en largeur d'abord

Une exploration systématique en largeur d'abord consiste à visiter en priorité tous les candidats d'une profondeur donnée avant de visiter ceux de la profondeur suivante : un candidat de profon-

leur i ne peut être rapatrié après un candidat de profondeur $i + 1$.

La mise en place d'une exploration en largeur d'abord nécessite une file d'attente d'urls gérée suivant la politique FIFO (*First In First Out*). Une gestion FIFO consiste tout simplement à mettre chaque url extraite d'un document rapatrié en queue de file d'attente. L'url du candidat le plus ancien dans la file d'attente est présentée en premier aux agents de rapatriement. Le schéma V.1 page 104 illustre un exemple de la gestion FIFO d'une pile d'urls au cours d'une exploration en largeur d'abord avec un seul agent de rapatriement.

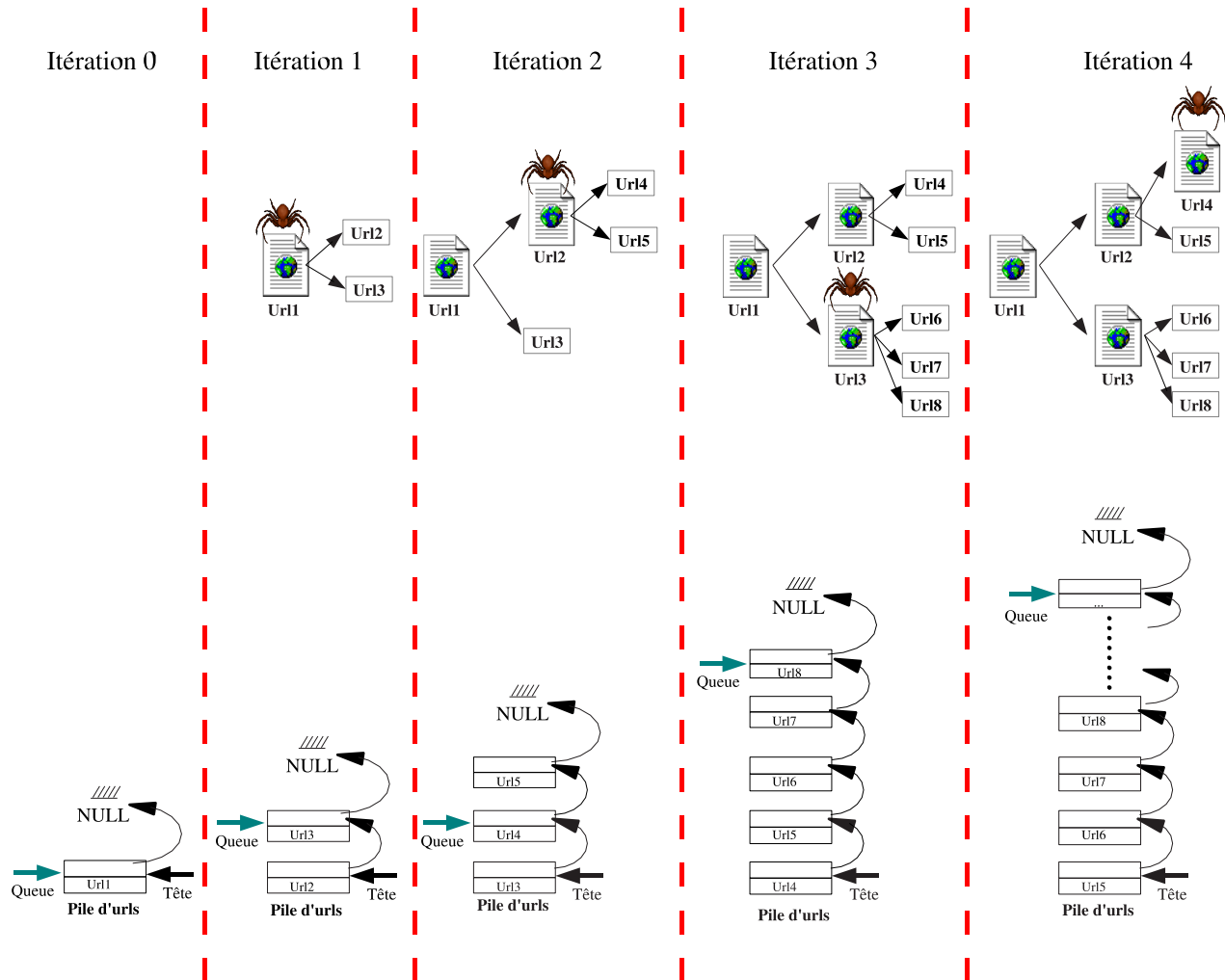


FIG. V.1 – Exploration en largeur d'abord

L'exhaustivité étant l'objectif principal d'une exploration systématique, une approche en largeur d'abord entraîne un accroissement important de la taille de la file d'attente. Cet accroissement devient critique lorsqu'il s'agit d'explorer une grande partie ou la totalité du graphe du web. Le dépassement de capacité de la file d'attente menace alors à tout moment de perturber le bon déroulement de l'exploration du robot.

Supposons que le facteur de branchement (c'est-à-dire le nombre de liens sortants) moyen d'une

page web est b . Supposons également que le robot a débuté son exploration à partir de k points de départ. La taille de la pile d'urls pour une telle exploration à une itération i donnée s'exprime comme suit :

$$size(queue) = k \times b^i \quad (V.1)$$

Cette formule montre clairement que, pour une exploration en largeur d'abord, la taille de la pile d'urls associée croît de manière exponentielle. Cette croissance exponentielle engendre de sérieux problèmes de passage à l'échelle du web tel qu'il est actuellement (plus de huit milliards de documents recensés par Google [Google site]).

Le problème du passage à l'échelle du web dans la conception des robots a été abordé dans les travaux de *Allan Heydon & Mark Najork* [Heydon et Najork 1999]. Les auteurs proposent Mercator, un robot explorateur doté de structures de données optimisées capables de contenir et gérer de grandes quantités d'informations. Ce robot possède également une architecture ouverte lui permettant d'intégrer différentes stratégies d'exploration sous forme de modules logiciels indépendants nécessitant un faible coût de développement. Mercator est, par ailleurs, l'outil proposé par Altavista [AltaVista site] dans le cadre de sa solution de collecte de documents dans les intranets volumineux.

Mercator a été expérimenté avec une stratégie en largeur d'abord pour collecter des statistiques sur le web. Lancée en mai 1999, cette exploration a permis de récolter plus de 65 millions de documents en 8 jours. Cette quantité représentait, à cette époque, un taux de couverture non négligeable du web (44% par rapport à l'index total d'Altavista en mai 1999). Il suffisait alors d'une dizaine de jours supplémentaires pour espérer couvrir la totalité du web. Avec la taille actuelle du web, il faut compter un délai 100 fois supérieur au délai précédent pour atteindre une couverture totale du web.

1.2 Exploration systématique en profondeur d'abord

Une exploration systématique en profondeur d'abord consiste à visiter en priorité le candidat ayant la plus grande profondeur dans la pile d'urls. Ce candidat correspond systématiquement au plus récent candidat découvert par le robot.

Une gestion de la file d'attente d'urls conforme à une politique LIFO (*Last In First Out*) permet de mettre en place un processus d'exploration web en profondeur d'abord. Une gestion LIFO consiste à mettre à chaque fois la dernière url extraite en tête de la file d'attente. Le schéma V.2 page 106 illustre un exemple de la gestion LIFO d'une pile d'urls au cours d'une exploration en profondeur d'abord avec un seul agent de rapatriement.

Contrairement à une exploration en largeur d'abord, une exploration en profondeur engendre une croissance linéaire de la taille de la file d'attente. Le dépassement de la capacité de la pile d'urls une stratégie en profondeur d'abord est beaucoup moins probable que pour une stratégie en largeur d'abord.

En reprenant les hypothèses précédentes (facteur de branchement égal à b et k points de départ), à l'itération i la taille de la file d'attente gérée en LIFO est donnée par la formule suivante :

$$size(queue) = k + i \times (b - 1) \quad (V.2)$$

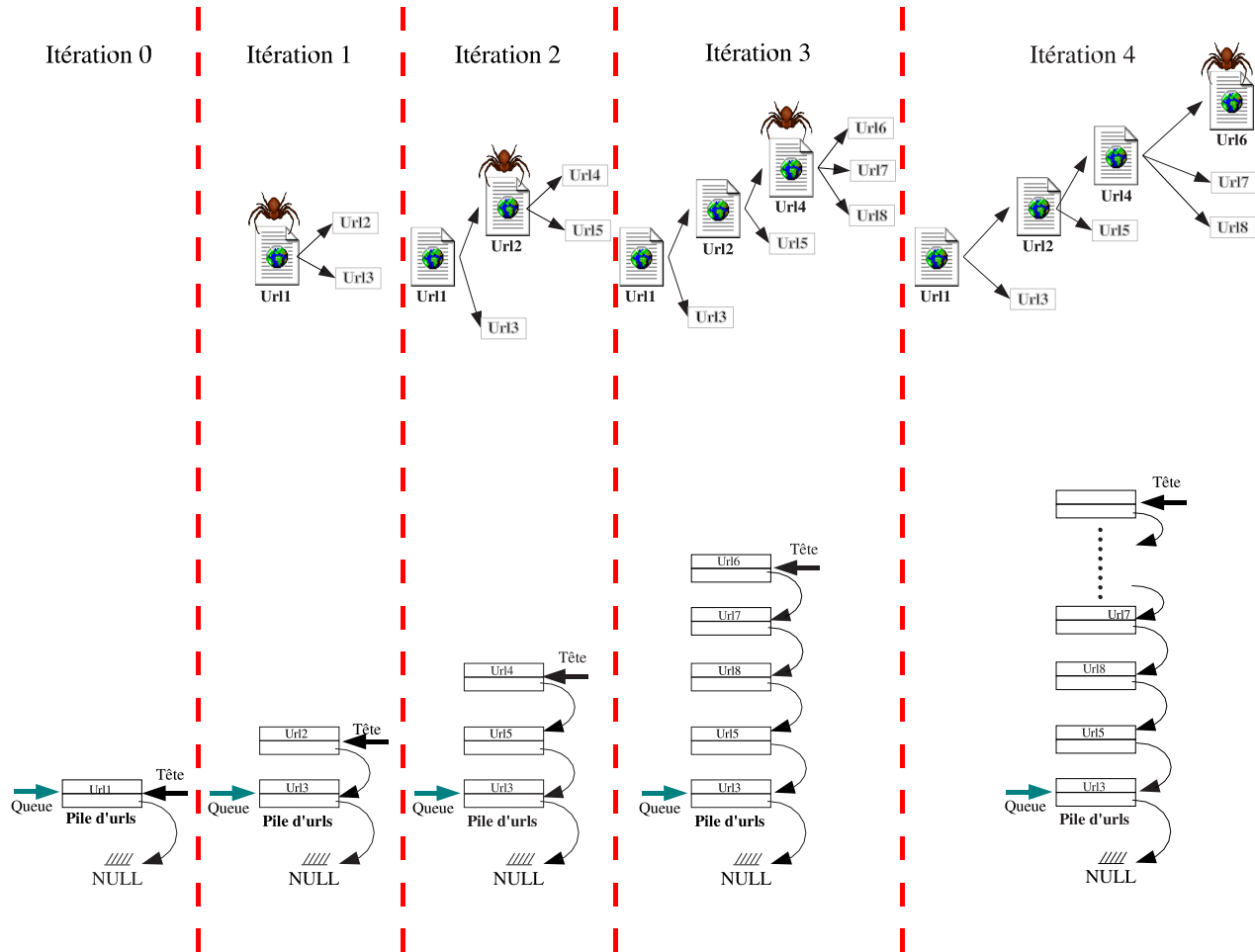


FIG. V.2 – Exploration en profondeur d'abord

1.3 Exploration systématique aléatoire

Une exploration systématique aléatoire, appelée également *Random Crawl*, consiste à choisir le prochain candidat à traiter de manière totalement aléatoire. La mise en place d'une telle approche nécessite un repositionnement arbitraire de la tête de la file d'attente du robot sur une url quelconque dans cette même file. Cette opération est effectuée après chaque itération pour désigner aléatoirement le prochain candidat à proposer aux agents de rapatriement.

À l'exception de sa tête, la file d'attente d'un robot utilisant une stratégie aléatoire possède un fonctionnement similaire à une pile LIFO. La croissance de la taille d'une file aléatoire est donc linéaire (voir la formule V.2 page 105).

L'exploration aléatoire du web est souvent utilisée par les robots pour effectuer un échantillonnage du web ou une analyse de la distribution des documents dans le graphe du web.

1.4 Exploration systématique par popularité d’abord

Une exploration web par popularité d’abord consiste à privilégier le parcours des documents populaires et importants à la suite du calcul d’un score de popularité. Il existe différentes métriques de popularité pour définir le score d’importance d’un document web : le PageRank [Brin et Page 1998], Hubs and Authority [Kleinberg 1999], le nombre de liens entrants [Cho et al. 1998], *etc.*

À partir du graphe exploré et à la suite de chaque itération du robot, la stratégie par popularité d’abord procède au calcul du score de popularité de chaque candidat en attente de traitement. La popularité d’un document étant, dans la plupart des cas, récursivement dépendante de la popularité des documents voisins (parents et/ou frères), le précédent calcul est étendu aux documents du graphe exploré pour répercuter la découverte de nouveaux liens entre les documents fraîchement rapatriés et les documents du graphe exploré dans la mise à jour des scores de popularités.

La file d’attente est réorganisée à chaque itération par ordre décroissant de popularité de sorte à présenter en priorité les candidats populaires aux agents de rapatriement.

Le calcul de la popularité des documents visités et des candidats d’une itération donnée repose généralement sur des opérations matricielles qui dépendent de la structure et de la taille du graphe exploré. Par exemple, le PageRank des documents visités et des candidats lors d’une exploration s’obtiennent par le calcul du vecteur propre principal de la matrice d’adjacence normalisée du graphe exploré par le robot [Brin et Page 1998].

La croissance de la taille du graphe exploré entraîne inévitablement un important surcoût dans la réalisation des opérations matricielles nécessaires au calcul de la popularité des candidats. Ce surcoût se traduit par une consommation plus importante des ressources matérielles et un allongement progressif du temps de calcul. Rappelons que, pendant ce temps de calcul, aucun rapatriement n’est possible. Cette situation conduit alors les agents du robot à stagner sur la frontière qui sépare les documents visités des documents candidats (voir figure IV.4 page 96). Le rapatriement reprend aussitôt que le candidat le plus populaire est connu. Ainsi, le déroulement du processus d’exploration par popularité est régulièrement retardé par les stagnations successives des agents de rapatriement.

La problématique du calcul de la mesure de popularité pendant le déroulement d’un processus d’exploration par popularité d’abord a été abordée dans les travaux de *Serge Abiteboul, Mihai Preda & Grégory Cobena* [Abiteboul et al. 2003]. Les auteurs proposent l’algorithme OPIC pour éviter les gros calculs matriciels et pour minimiser l’effet de stagnation du robot. L’algorithme OPIC permet le calcul en-ligne et incrémental de la popularité d’un document par distribution et propagation d’une quantité symbolique d’argent comptant appelée *cash*.

Au commencement de l’exploration, l’algorithme OPIC associe une même somme initiale de *cash* aux documents de départ. Le principe consiste alors à distribuer cette somme sur les prochains voisins de ces points de départ. Cette distribution est récursivement répétée sur les documents rapatriés au fur et à mesure de l’avancement de l’exploration.

Un document d rapatrié perçoit de son père (document pointant sur d) la même somme que chacun de ses frères (documents ayant le même père que d). En d’autres mots, si le père du document d possède un *cash* de c et n liens sortants, le document d perçoit une somme égale à $\frac{c}{n}$.

Le document d peut également percevoir d'autres sommes différentes de la part d'autres parents pointant sur lui. Les autres parents de d n'étant pas tous découverts en même temps, la perception des différentes sommes dues au document d s'étale progressivement sur toute la période de l'exploration. Plus le document d est pointé par plusieurs parents, plus le robot a de chance de le visiter souvent et donc lui attribuer rapidement ses gains. Un document populaire est alors défini comme étant un document percevant beaucoup d'argents de ses parents.

Les expérimentations, effectuées sur différentes explorations d'un graphe web prédéfini avec le robot Xyro [Mignet et al. 2001, Sédrati-Dinet 2003] de l'entreprise Xyleme [Xyleme site], ont montré une relative rapidité dans la convergence de l'algorithme OPIC vers les valeurs définitives de popularité des documents. Cette convergence nécessite néanmoins des visites répétées sur les documents du graphe exploré pour réajuster les gains (la popularité) de chaque document. En d'autres mots, la fiabilité de la mesure ne se confirme qu'à la suite de plusieurs cycles d'explorations sur le même graphe. Une exploration par popularité d'abord fondée sur l'algorithme OPIC peut se révéler utile dans le rafraîchissement et la mise à jour du contenu des documents d'un graphe visité au préalable. Dans ce cas de figure, la mise à jour peut être guidée par la popularité en focalisant le robot sur le rapatriement prioritaire des documents jugés populaires lors des précédentes explorations.

Dans leur étude expérimentale [Cho et al. 1998], *Junghoo Cho, Hector Garcia Molina et Lawrence Page* ont utilisé la mesure du PageRank [Brin et Page 1998] pour évaluer et comparer les résultats d'exploration de trois stratégies : exploration systématique par popularité d'abord, exploration systématique en largeur d'abord et exploration systématique aléatoire. Les expérimentations menées sur le site web de l'université de *Stanford* (<http://www.stanford.edu/>) montrent qu'une stratégie par popularité d'abord rapatrie rapidement des documents populaires et en quantité supérieure aux autres approches systématiques. Précisons que les plus mauvais résultats sont enregistrés par l'exploration aléatoire.

1.5 Évaluation des approches systématiques

Par définition, l'orientation dans le web d'une exploration systématique s'appuie uniquement sur les propriétés de la structure du graphe du web. En l'absence de critères « thématiques » de pertinence et outre les mesures quantitatives classiques (taux de documents rapatriés par unité de temps, nombre de connexions à la fois, *etc.*), les mesures de popularité représentent les seuls critères d'évaluation de la qualité d'une exploration systématique.

En utilisant le PageRank [Brin et Page 1998] comme mesure de qualité, *Mark Najork & Janet L. Wiener* [Najork et Wiener 2001] et *Junghoo Cho & Hector Garcia Molina & Lawrence Page* [Cho et al. 1998] ont montré que la stratégie en largeur d'abord et la stratégie par popularité d'abord rapatriaient plus de documents populaires (ayant un PageRank élevé) que les autres approches systématiques. Ce résultat s'explique par le fait qu'un candidat populaire possède un nombre important de liens entrants provenant de documents eux aussi populaires. Cette propriété rend ce candidat rapidement atteignable par le robot. D'autre part et contrairement à la stratégie en profondeur d'abord et à la stratégie aléatoire, la stratégie en largeur d'abord entraîne une croissance exponentielle de la pile d'urls (voir formule V.1 page 105). Cette croissance augmente donc considérablement les chances d'inclure et de couvrir, dès les premières itérations, les candidats populaires dans la file d'attente d'urls d'une stratégie en largeur d'abord.

2 Exploration par stratégie de type « le meilleur d’abord »

Une exploration par le meilleur d’abord repose sur une stratégie informée pour guider le parcours du robot dans le graphe du web. Contrairement à une exploration systématique, une exploration de cette catégorie utilise une fonction fournie par avance pour évaluer les candidats et déterminer leurs priorités de rapatriement. Cette fonction d’évaluation calcule un score pour chaque candidat de l’exploration à partir de certaines propriétés du voisinage de chaque candidat considéré. « Le meilleur » candidat, à rapatrier en priorité, est le candidat totalisant le score le plus élevé.

Cette approche peut être envisageable pour initier des explorations à la recherche de documents satisfaisant des critères simples : rechercher par exemple des documents satisfaisant une requête mots clés simple. Pour cet exemple, il est possible d’envisager une fonction d’évaluation qui privilégie les candidats pointés par des documents totalisant un score élevé de similarité avec la requête mots clés (heuristique naïve).

Pour une recherche plus complexe (recherche de documents thématiques par exemple), le choix de la fonction d’évaluation des candidats n’est pas facile. En effet, en l’absence du contenu textuel du candidat, il est difficile d’anticiper, avec précision, la pertinence d’un candidat non rapatrié par de simples heuristiques. Pour accroître la précision et la fiabilité de l’évaluation des candidats, le recours aux techniques classiques d’apprentissage automatique (apprentissage bayésien par exemple) s’avère nécessaire.

2.1 L’algorithme Fish-Search

L’algorithme Fish-Search [De Bra et Post 1994] a été conçu pour rechercher des documents satisfaisant une requête mots clés donnée, appelée *requête d’exploration*. La satisfaisabilité de la requête d’exploration est déterminée par une fonction booléenne de pertinence. Cette fonction vérifie la présence des mots clés de la requête d’exploration dans le contenu de chaque document rapatrié. Elle renvoie 1 lorsque ces mots sont présents dans le document évalué, 0 sinon.

Le fonctionnement de l’algorithme Fish-Search est comparable au comportement d’un poisson évoluant dans un environnement inconnu. L’objectif est de survivre le plus longtemps possible en retrouvant le maximum de nourriture. En cas de rupture de nourriture, le poisson est dans l’impossibilité de continuer sa recherche et meurt de famine.

Dans le web, les documents évalués à 1 par la fonction de pertinence représentent la nourriture des agents de l’algorithme Fish-Search. Chaque agent créé (un poisson) est initialisé avec trois paramètres : la requête d’exploration, des urls de départ et une quantité initiale D de vitalité. Cette quantité est décrémentée à chaque fois que l’agent rapatrie un document ne satisfaisant pas la requête d’exploration (évalué à 0 par la fonction de pertinence). Lorsque la quantité de vitalité devient nulle, l’agent se retrouve dans une situation de famine et est, par conséquent, arrêté.

La stratégie du « meilleur d’abord », utilisée par l’algorithme Fish-Search, repose sur une heuristique naïve qui consiste à privilégier le suivi des liens provenant de documents déjà rapatriés et évalués à 1 par la fonction de pertinence. Pour mettre en œuvre cette heuristique, l’algorithme Fish-Search utilise une fonction d’évaluation de liens vers des candidats plutôt que les candidats

eux-mêmes. Cette fonction associe un score maximal aux $k\%$ liens sortants¹ d’un document rapatrié et évalué à 1 par la fonction de pertinence. Les $(1 - k\%)$ liens restants de ce document sont associés à un score nul. Pour un document évalué à 0 par la fonction de pertinence, la fonction d’évaluation des liens associe un score nul à k' ($k' > k$) des liens sortants provenant de ce document. Les $(1 - k'\%)$ liens restants sont associés à un score équivalent à 50% du score maximal précédent. Cette dernière association permet de donner une chance aux liens provenant de documents évalués à 0 par la fonction de pertinence. En effet, dans certains cas, ces documents peuvent pointer vers les documents recherchés.

Remarque 2.1 *Les proportions k et k' ont été choisies par les auteurs de sorte à limiter le nombre de liens à suivre dans la pile d’urls.*

Précisons toutefois qu’avec trois valeurs discrètes possibles retournées (0, 50% du score maximal et le score maximal), la fonction d’estimation de l’intérêt du suivi des liens vers les candidats est peu discriminante en cas d’égalité de scores.

2.2 L’algorithme Shark-Search

L’algorithme Shark-Search [Hersovici et al. 1998], développé dans les laboratoires d’IBM-Israël à Haïfa, permet de mettre en œuvre une stratégie de type le « meilleur d’abord » pour explorer le web à la recherche de documents satisfaisant une requête d’exploration donnée. Cet algorithme est une amélioration de l’algorithme Fish-Search (voir section précédente). Le fonctionnement global de Shark-Search est très similaire au fonctionnement de l’algorithme Fish-Search. Il existe cependant deux différences majeures :

1. Contrairement à l’algorithme Fish-Search, l’algorithme Shark-Search utilise une fonction de pertinence numérique plus fiable que la vérification booléenne de la présence de mots clés. Cette fonction calcule la similarité entre la requête d’exploration et le contenu de chaque document rapatrié. Plus précisément, la fonction de pertinence de l’algorithme Shark-Search calcule le cosinus de l’angle qui sépare le vecteur de mots du document du vecteur de mots de la requête d’exploration.
2. Pour être plus discriminante, la fonction d’évaluation des liens vers les candidats de l’algorithme Shark-Search renvoie des estimations numériques. Avec plus de valeurs retournées possibles, les égalités de scores entre les liens pointant vers des candidats sont peu probables.

Soit l un lien sortant d’un document rapatrié d pour pointer vers un candidat donné. Le calcul de l’intérêt du suivi du lien l dans l’algorithme Shark-Search est effectué par une combinaison linéaire entre deux paramètres :

1. le score renvoyé par la fonction de pertinence sur le document d’origine du lien (c’est-à-dire d) ;
2. le score de similarité (cosinus) entre le vecteur de mots de l’ancre du lien l et le vecteur de mots de la requête d’exploration.

L’heuristique proposée par l’algorithme Shark-Search est plus fine et plus discriminante que celle proposée par l’algorithme Fish-Search. Grâce à la combinaison linéaire décrite plus haut, la distinction entre les liens pointant vers des candidats s’effectue par un compromis entre la pertinence du document d’où provient le lien et la pertinence de sa description (ancree) par rapport à la requête d’exploration. Contrairement à l’algorithme Fish-Search, ce compromis permet au robot de départager les liens provenant d’un même document.

¹choisis aléatoirement

3 Exploration sélective du web guidée par des critères thématiques

Une exploration sélective du web guidée par des critères thématiques est définie comme étant un processus de parcours du web ayant pour objectif de rapatrier rapidement le plus possible de documents satisfaisant des critères thématiques définis par avance. Nous appelons ces documents : « *les documents cibles* ». Par exemple, il est possible d'initier une exploration sélective à la recherche de documents de cours informatiques en ligne. L'objectif d'une telle exploration est de trouver le maximum de documents de cours informatiques (contrainte de précision), si possible parmi les premiers documents rapatriés (contrainte de rapidité).

Afin d'atteindre les objectifs de précision et de rapidité, une approche sélective doit minimiser le parcours inutile de documents ne correspondant pas aux documents cibles. Pour cela, une fonction d'évaluation de l'intérêt du rapatriement des candidats est utilisée pour suggérer, à chaque itération, le candidat susceptible de correspondre le plus aux critères thématiques guidant l'exploration. Ce candidat est dit « *le plus prometteur à rapatrier* ».

Remarque 3.1 *Dans la littérature scientifique, certaines approches utilisent une fonction d'évaluation de l'intérêt de suivre un lien pointant vers un candidat plutôt que l'intérêt du rapatriement du candidat lui-même. Ces deux fonctions sont équivalentes car, indépendamment du lien suivi, elles suggèrent dans les deux cas le candidat le plus intéressant à rapatrier.*

La fonction d'évaluation de l'intérêt des candidats calcule un score pour chaque candidat d'une itération donnée. En général, ce calcul s'appuie sur les caractéristiques du voisinage du candidat considéré (pertinence de ses parents et de ses frères, la présence de certains mots clés dans ses liens entrants, *etc.*). Les scores calculés indiquent au robot l'intérêt du rapatriement de chaque candidat dans la pile d'urls par rapport aux critères thématiques de l'exploration. En classant ces candidats par ordre décroissant de scores, le robot est capable d'optimiser son parcours dans le web en ciblant, à chaque itération, le rapatriement du document le plus prometteur.

Remarque 3.2 *Les scores calculés par la fonction d'évaluation de l'intérêt du rapatriement des candidats sont, en général, des estimations qui anticipent l'intérêt et la pertinence du candidat par rapport aux critères thématiques guidant l'exploration. Ces scores possèdent donc une valeur indicative seulement. Autrement dit, une fois rapatrié, le document jugé le plus prometteur peut ne pas correspondre aux documents cibles souhaités. Ainsi, la difficulté de la mise en place d'une approche sélective d'exploration du web réside dans l'élaboration de la fonction d'évaluation la plus fiable possible.*

La fonction d'évaluation de l'intérêt du rapatriement des candidats d'une exploration sélective emploie des mécanismes plus fiables que les heuristiques utilisées dans les approches de la catégorie précédente (voir section précédente page 109). Ces mécanismes sont mis en œuvre par apprentissage automatique, ce qui les rend mieux adaptés aux exigences d'une recherche guidée par des critères thématiques.

Dans la littérature scientifique, nous avons recensé deux manières différentes de construire la fonction d'évaluation de l'intérêt de rapatriement d'un lien : par apprentissage automatique initial sur un graphe d'entraînement ou par apprentissage automatique en-ligne sur le graphe exploré.

3.1 Approches sélectives avec apprentissage initial

Cette catégorie regroupe les approches mettant en œuvre des fonctions avancées d'évaluation de l'intérêt d'un candidat pour rechercher des documents cibles (documents thématiques par exemple). Ces fonctions reposent sur un processus d'apprentissage automatique effectué avant le lancement de l'exploration. L'objectif de cet apprentissage est de définir une fonction capable de reconnaître un candidat prometteur en maximisant son score d'intérêt de rapatriement.

L'apprentissage de la fonction d'évaluation de l'intérêt de rapatriement des candidats s'effectue sur un graphe appelé « graphe d'entraînement ». Le graphe d'entraînement doit reproduire une configuration réelle des occurrences des documents cibles et des documents de leur voisinage tels qu'ils pourraient être trouvés dans le web. La construction d'un tel graphe nécessite une connaissance préalable d'un minimum de documents représentatifs des documents ciblés par l'exploration. Pour certaines thématiques, cette connaissance n'est pas évidente car elle renvoie la phase de construction du graphe d'entraînement à la même problématique qui a motivé le recours à l'exploration sélective : comment et où trouver dans le web des documents en rapport avec la thématique ciblée par l'exploration et en quantité suffisante ?

Dans le cas où la construction du graphe d'entraînement est possible, la distinction des documents cibles du graphe d'entraînement est indispensable. Un processus d'étiquetage est nécessaire. Il n'est toutefois pas toujours possible d'automatiser ce processus. En effet, pour certains documents cibles complexes, l'automatisation comporte un risque important d'erreurs et d'imprécision dans l'étiquetage des documents du graphe d'entraînement. Cette imprécision entraîne une détérioration non négligeable de la qualité de la fonction apprise. Une intervention manuelle s'impose alors comme l'alternative la plus fiable pour distinguer les documents cibles du graphe d'entraînement des autres documents. Ce travail manuel est souvent coûteux en temps et fastidieux à réaliser.

Dans ce qui suit, nous vous présentons les deux approches majeures d'exploration sélective de cette catégorie : l'approche de *Jason Rennie & Andrew Katchies McCallum* [Rennie et McCallum 1999, McCallum et al. 1999] et l'approche de *Michelangelo Diligenti et al.* [Diligenti et al. 2000].

3.1.1 Focused Crawling de Jason Rennie & Andrew Katchies McCallum

Dans leurs articles [Rennie et McCallum 1999, McCallum et al. 1999], *Jason Rennie & Andrew Katchies McCallum* se sont intéressés à la construction d'un moteur de recherche spécialisé dans les publications scientifiques en informatique au format postscript. Le moteur en question est CORA [Cora site]. Pour alimenter ce moteur en publications informatiques en-ligne, les auteurs proposent une approche sélective d'exploration du web qui utilise une fonction d'évaluation du bénéfice à espérer du suivi de chaque lien pointant sur les candidats de l'exploration. Cette fonction exploite les résultats issus de la combinaison de deux techniques d'apprentissage automatique appliquées à un graphe d'entraînement construit avant le lancement de l'exploration. Les deux techniques utilisées par les auteurs sont : l'apprentissage par renforcement [Cornuéjols et al. 2002a, Sutton et Barto 1998, Mitchell 1997a, Kaelbling et al. 1996] et la classification bayésienne [Cornuéjols et al. 2002b, Mezaour 2001, Lewis 1998, Mitchell 1997b].

La phase d'entraînement de l'approche proposée par *Jason Rennie & Andrew Katchies McCallum* repose sur un graphe d'entraînement construit à partir des sites web des départements informatiques de trois universités américaines : Brown University [Brown site], Cornell University [Cornell site]

et University of Pittsburgh [Pittsburgh site]. Le graphe résultant représente 53.012 documents et 592.216 liens hypertextes. En parallèle, un vocabulaire de référence est construit à partir des mots contenus dans les 53.012 documents du graphe d'entraînement. Nous notons ce vocabulaire par $\mathcal{V}$ et nous supposons que sa cardinalité est de K mots différents.

L'étiquetage du graphe d'entraînement a pu être automatisé grâce à une heuristique dont la fiabilité a été estimée à 95% par les auteurs. Cette heuristique considère qu'un document d est pertinent par rapport aux besoins du moteur CORA si d est au format postscript (son extension = 'ps') et si la tranformation de d en fichier texte contient le mot 'abstract' (section contenue dans la plupart des articles scientifiques). 2.263 documents postscript du graphe d'entraînement ont pu ainsi être étiquetés comme étant pertinents. Il est clair que l'approche automatique proposée par les auteurs pour construire le graphe d'entraînement n'est pas reproductible pour les cas où les documents cibles ne sont pas aussi facilement identifiables.

Deux classifieurs successifs sont ensuite entraînés sur le graphe d'entraînement pour reconnaître les propriétés qui caractérisent les liens prometteurs. Le premier classifieur utilise la technique d'apprentissage par renforcement pour associer à tout lien du graphe d'entraînement un score de pertinence. Ce score représente la récompense ou le bénéfice à attribuer au robot s'il parcourt ce lien dans le graphe d'entraînement. Les 592.216 liens du graphe d'entraînement sont alors mis en correspondance avec leurs récompenses respectives grâce à une fonction de calcul que nous notons par f_r . Les valeurs obtenues sont discrétisées en intervalles afin de créer n classes $C_i = [\gamma_1^i, \gamma_2^i]$ d'équivalence de récompense entre les liens. Les bornes γ_1^i, γ_2^i sont choisies de sorte à refléter la distance qui sépare les liens de la classe C_i des documents pertinents. Cette distance permet également de définir l'ordre de préférence (priorité) entre les liens des différentes classes d'équivalence :

$$(\gamma_1^x < \gamma_2^x < \gamma_1^y < \gamma_2^y) \Rightarrow (C_x < C_y) \quad (V.3)$$

En résumé, le processus d'apprentissage du classifieur par renforcement, noté « *Reinforcement* », peut être vu comme étant la composition de deux étapes : l'étape du calcul de la pertinence d'un lien (symbolisé par la fonction « *Reward* ») et l'étape d'association des classes d'équivalences C_i (symbolisé par la fonction « *Discretize* »). Cette composition s'exprime comme suit :

$$\begin{array}{ccccc} \mathcal{L}_T & \xrightarrow{\text{Reward}} & [0, +\infty[& \xrightarrow{\text{Discretize}} & \{C_1, \dots, C_n\} \\ l & \longrightarrow & \text{Reward}(l) & \longrightarrow & C_i \text{ tel que : } \gamma_1^i \leq \text{Reward}(l) \leq \gamma_2^i \end{array} \quad (V.4)$$

$$\text{Reinforcement} = \text{Discretize} \circ \text{Reward}$$

Une fois la fonction « *Reinforcement* » établie, chaque lien l du graphe d'entraînement est représenté par un vecteur de dimension K traduisant les occurrences des mots du vocabulaire $\mathcal{V}$ dans la description textuelle associée à l . La description textuelle retenue par les auteurs pour décrire un lien se compose de trois parties : les mots provenant de l'ancre d'un lien, les mots contenus dans l'url référencée et les mots provenant du contenu du document d'origine du lien. La fonction Ψ de caractérisation d'un lien s'exprime alors comme suit :

$$\begin{array}{rcl} \Psi & : & \mathcal{L} \longrightarrow \mathbb{N}^K \\ & & l \longrightarrow \Psi(l) = \langle \lambda_1, \dots, \lambda_K \rangle \end{array} \quad (V.5)$$

La fonction «*Reinforcement*» apprise par le premier classifieur est partielle car elle ne met en correspondance que les liens du graphe d'entraînement avec la classe d'équivalence qui leur est associée. Cette fonction est donc spécifique au graphe d'entraînement et est inutilisable pour des liens provenant d'autres graphes web. Ainsi, le second classifieur utilise les principes de la classification bayésienne pour généraliser la fonction «*Reinforcement*» à n'importe quel lien web. L'objectif visé consiste à déterminer la classe d'équivalence C_i d'un lien ($l \in \mathcal{L}$) quelconque à partir seulement du vecteur caractéristique de ce lien ($\Psi(l)$). Pour cela, le classifieur bayésien calcule l'influence de l'occurrence de chaque mots w_j du vocabulaire $\mathcal{V}$ sur l'appartenance d'un lien aux différentes classes d'équivalences. Cette influence est représentée par la probabilité qu'un lien $l \in \mathcal{L}$ ait la caractéristique w_j (*i.e.* $\lambda_j \neq 0$) sachant que la classe d'équivalence de ce lien est C_i . Le calcul de cette probabilité, notée $P(w_j/C_i)$, est approximée par l'observation de l'influence du mot w_j sur la classe C_i dans le graphe d'entraînement. La fréquence de chaque classe d'équivalence, notée $P(C_i)$, est également calculée à partir du graphe d'entraînement. Le déroulement du classifieur bayésien est résumé comme suit :

$$\{\langle \Psi(l), \text{Reinforcement}(l) \rangle \mid l \in \mathcal{L}_T\} \xrightarrow{\text{Bayesian}} \begin{cases} \{P(C_i) \mid i = 1..n\} \\ \{P(w_j/C_i) \mid (i = 1..n) \wedge (j = 1..K)\} \end{cases} \quad (\text{V.6})$$

La fonction $\mathcal{F}_{link}$ d'estimation de l'utilité de suivre un lien est alors construite sur la base des sorties du classifieur bayésien. À partir de la caractérisation $\Psi(l)$ d'un lien l et de l'hypothèse d'indépendance des occurrences des mots du vocabulaire $\mathcal{V}$, la fonction $\mathcal{F}_{link}$ calcule la probabilité pour que le lien l appartienne à chaque classe d'équivalence C_i . La classe d'équivalence retenue pour le lien l est la classe C_x pour laquelle le lien a le plus de chances d'appartenir. La classe C_x est la classe considérée comme la plus probable selon les connaissances apprises du graphe d'entraînement. Les formules suivantes résument le fonctionnement de la fonction $\mathcal{F}_{link}$:

$$\begin{aligned} P(C_i/\Psi(l)) &= \mathbf{P}(\mathbf{C}_i) \times \frac{P(\Psi(l)/C_i)}{P(\Psi(l))} \\ &= \mathbf{P}(\mathbf{C}_i) \times \frac{P(\langle \lambda_1, \dots, \lambda_K \rangle / C_i)}{P(\langle \lambda_1, \dots, \lambda_K \rangle)} \\ &= \mathbf{P}(\mathbf{C}_i) \times \frac{\prod_{j=1}^K \mathbf{P}(\mathbf{w}_j / \mathbf{C}_i)}{P(\langle \lambda_1, \dots, \lambda_K \rangle)} \quad \text{hypothèse d'indépendance} \\ &\propto \mathbf{P}(\mathbf{C}_i) \times \prod_{j=1}^K \mathbf{P}(\mathbf{w}_j / \mathbf{C}_i) \quad \text{où : } \propto \text{ est le symbole du prédicat « proportionnel »} \end{aligned} \quad (\text{V.7})$$

Remarque 3.3 Les termes mis en gras dans la formule V.7 sont remplacés par les approximations apprises par le classifieur bayésien.

$$\begin{aligned} \mathcal{F}_{link} &: \mathcal{L} \longrightarrow \{C_1, \dots, C_n\} \\ l &\longrightarrow \mathcal{F}_{link}(l) = C_x \quad \text{tel que } C_x = \arg \max_{C_i, i=1..n} P(C_i/\Psi(l)) \end{aligned} \quad (\text{V.8})$$

Après ce processus d'apprentissage, l'exploration sélective proposée par Jason Rennie & Andrew Katchies McCallum consiste à parcourir le web en privilégiant les liens appartenant à la plus grande

classe d'équivalence (voir l'ordre défini dans les classes d'équivalence des lien dans la formule V.3 page 113).

Dans leurs expérimentations, les auteurs ont comparé leur approche avec une exploration systématique en largeur d'abord sur le site du département informatique de l'université du Texas [Austin site]. Les résultats montrent clairement qu'une exploration sélective avec apprentissage initial retrouve plus rapidement les publications informatiques qu'une recherche systématique en parcourant moins de documents.

3.1.2 Focused Crawling de Michelangelo Diligenti et al.

Michelangelo Diligenti et al. proposent un robot sélectif, appelé CFC (*Context Graph Crawler*) pour cibler l'exploration du web sur la recherche de documents thématiques [Diligenti et al. 2000]. Le fonctionnement du robot CFC repose sur la notion du *graphe contextuel*.

Un graphe contextuel est un graphe où les nœuds sont récursivement hiérarchisés par niveaux de profondeur. Les nœuds du niveau (i) d'un graphe contextuel représentent les successeurs immédiats des nœuds appartenant au niveau suivant, *i.e.*, le niveau ($i + 1$).

Comme pour le robot développé par *Jason Rennie et Andrew Katchies McCallum*, le robot CFC nécessite une phase d'apprentissage sur un graphe d'entraînement. Pour contourner la contrainte de l'étiquetage d'un graphe d'entraînement, *Michelangelo Diligenti et al.* proposent de construire le graphe d'entraînement en partant de documents pertinents connus par avance. Le principe consiste alors à recréer l'environnement et le contexte web qui entoure ces documents pertinents. Le graphe ainsi construit est appelé « *graphe contextuel* ».

Un graphe contextuel est un graphe de documents web hiérarchisé en plusieurs niveaux. Chaque niveau regroupe les parents des documents du niveau inférieur. Ainsi, le premier niveau (niveau 0) contient un échantillon de documents cibles. Le deuxième niveau contient les parents des documents pertinents, le troisième niveau contient les parents des documents du deuxième niveau, *etc.* Pour cela, le recours au service des liens inverses proposés par la plupart des moteurs de recherche est nécessaire. Un service de liens inverses, *Reverse linkage or Backward linking*, est un service qui reçoit en entrée une url valide d'un document et renvoie en sortie la liste des parents du document correspondant à l'url soumise. Par exemple, sous Google, la soumission de la requête "`link :www.lri.fr`" permet d'obtenir tous les parents de la page d'accueil du site du LRI.

La structure d'un graphe contextuel d'entraînement est représentée dans la schéma V.3 page 116. Le déroulement d'un processus de construction d'un graphe contextuel d'entraînement $\mathcal{G}_T^C(\mathcal{D}_T^C, \mathcal{L}_T^C)$ est montré dans l'algorithme 1 page 117.

Contrairement au graphe d'entraînement construit dans l'approche proposée par *Jason Rennie et Andrew Katchies McCallum* [Rennie et McCallum 1999], la construction du graphe contextuel d'entraînement du robot CFC est entièrement automatique. Toutefois, la recherche de documents cibles pour initialiser cette construction demeure encore manuelle. De ce fait, il n'est pas évident de trouver manuellement un nombre suffisant de documents pertinents garantissant un apprentissage efficace pour certaines thématiques complexes.

Algorithme 1 Construction d'un graphe contextuel d'entraînement

Inputs : $U_0 = \{url_1^*, \dots, url_n^*\} : n$ urls de documents pertinents de départ
 N : la profondeur du graphe contextuel

Output : $\mathcal{G}_T^C(\mathcal{D}_T^C, \mathcal{L}_T^C)$ un graphe contextuel d'entraînement

```

1:  $\mathcal{D}_T^C \leftarrow \emptyset$ 
2:  $\mathcal{L}_T^C \leftarrow \emptyset$ 
3:  $i \leftarrow 0$ 
4:  $\mathcal{D}_i \leftarrow \emptyset$ 
5:  $\mathcal{L}_i \leftarrow \emptyset$ 
6:  $\mathcal{U}_i \leftarrow U_0$ 
7: Foreach ( $i < N$ ) Do
8:    $\mathcal{U}_{i+1} \leftarrow \emptyset$ 
9:    $\mathcal{L}_i \leftarrow \emptyset$ 
10:  Foreach ( $u \in \mathcal{U}_i$ ) Do
11:    charger le document  $d$  correspondant à l'url  $u$ 
12:     $\mathcal{D}_i \leftarrow \mathcal{D}_i \cup \{d\}$ 
13:    If ( $i \geq 1$ ) Then
14:       $\mathcal{L}_i \leftarrow \mathcal{L}_i \cup \{1 / \exists d' \in \mathcal{D}_{i-1} : d \xrightarrow{l} d'\}$ 
15:    End If
16:    interroger Google pour obtenir l'ensemble  $U_{google}$  des urls pointant sur le document  $d$ 
17:     $\mathcal{U}_{i+1} \leftarrow \mathcal{U}_{i+1} \cup U_{google}$ 
18:  End For
19:   $\mathcal{D}_T^C \leftarrow \mathcal{D}_T^C \cup \mathcal{D}_i$ 
20:  If ( $i \geq 1$ ) Then
21:     $\mathcal{L}_T^C \leftarrow \mathcal{L}_T^C \cup \mathcal{L}_i$ 
22:  End If
23:   $i \leftarrow i + 1$ 
24: End For
25: Return ( $\mathcal{D}_T^C, \mathcal{L}_T^C$ )

```

cela, *Michelangelo Diligenti et al.* proposent d'entraîner un classifieur bayésien à savoir associer à n'importe quel document du web le niveau du graphe contextuel qui lui correspond. Pour cela, le classifieur bayésien évalue l'influence des occurrences des mots de chaque document du graphe d'entraînement sur l'appartenance à un niveau donné. Ces mots forment le vocabulaire $\mathcal{V}$ de référence

du graphe contextuel ($|\mathcal{V}| = K$). Les documents du graphe contextuel sont remplacés par le vecteur d'occurrences de chaque mot $w_{j,j=1..K}$ du vocabulaire $\mathcal{V}$ dans leur contenu. Le classifieur bayésien fournit alors en sortie les estimations suivantes :

$$\{\langle d, \mathcal{D}_i \rangle / d \in \mathcal{D}_i\} \xrightarrow{\text{Bayesian}} \begin{cases} \{P(D_i), i = 0..N-1\} \\ \{P(w_j/D_i), (i = 0..N-1) \wedge (j = 1..K)\} \end{cases} \quad (\text{V.10})$$

En s'appuyant sur l'hypothèse naïve d'indépendance des occurrences des mots du vocabulaire, la probabilité qu'un document quelconque $d = \langle w_{j,j=1..K} \rangle$ du web appartienne à un niveau donné du graphe contextuel est calculée par la formule suivante :

$$P(D_i/d) \propto P(D_i) \times \prod_{j=1}^K P(w_j/D_i) \quad (\text{V.11})$$

Cette formule est par la suite utilisée par le robot CFC lors de son exploration du web pour placer chaque document d rapatrié dans le niveau $\mathcal{D}_x$ qui lui convient dans le graphe contextuel :

$$\mathcal{D}_x \leftarrow \mathcal{D}_x \cup \{d\} \text{ tel que } \mathcal{D}_x = \arg \max_{\mathcal{D}_i, i=0..N-1} P(\mathcal{D}_i/d) \quad (\text{V.12})$$

La stratégie d'exploration du robot CFC consiste alors à suivre prioritairement les liens provenant de documents proches du niveau 0 :

$$\begin{aligned} \mathcal{F}_{link} &: \mathcal{L} \longrightarrow [-N, 0] \\ l &\longrightarrow \mathcal{F}_{link}(l) = -i \text{ tel que } (d \xrightarrow{l} d') \wedge (d \in \mathcal{D}_i) \end{aligned} \quad (\text{V.13})$$

Les auteurs ont lancé deux explorations pour tester le robot CFC sur deux recherches thématiques différentes : la recherche d'articles scientifiques en ligne (exploration ❶) et la recherche de documents sur le vélo (exploration ❷).

Les documents obtenus à la suite de l'exploration ❶ ont été comparés aux documents rapatriés parallèlement par une exploration systématique en largeur d'abord. Les résultats montrent une nette efficacité de l'approche sélective à retrouver des documents scientifiques. Pour une période de temps donnée, le robot CFC retrouve en moyenne 2 fois plus de documents pertinents qu'une exploration en largeur d'abord.

Pour les documents sur le vélo, les auteurs ont comparé leur approche (l'exploration ❷) avec une exploration sélective simple (notée ❸) reposant sur une fonction naïve d'estimation de liens. Cette fonction associe à un lien le score d'appartenance de son document d'origine à la thématique du vélo. Pour déterminer ce score, les auteurs ont réutilisé les estimateurs du niveau 0 issus de l'entraînement sur le graphe CFC construit pour la thématique des vélos. Ainsi, si l est un lien tel que $d \xrightarrow{l} d'$, $d = \langle w_{j,j=1..K} \rangle$ est déjà rapatrié et que d' est encore un candidat alors $\mathcal{F}_{link}(l) = P(D_0) \times \prod_{j=1}^K P(w_j/D_0)$ (voir formule V.11 page 118).

La stratégie mise en œuvre dans l'exploration ❸ revient à privilégier le rapatriement du candidat pointé par le lien provenant du document le plus « pertinent » par rapport à la thématique du vélo.

La comparaison des résultats de l'exploration ❷ aux résultats de l'exploration ❸ a montré qu'un entraînement sur un graphe contextuel à plusieurs niveaux permet de mieux localiser les documents pertinents qu'une stratégie naïve reconnaissant uniquement les documents cibles (niveau 0). Les auteurs ont toutefois souligné l'insuffisance de la précision de leur approche sur cette recherche thématique (vélo). En effet, les documents traitant sur le vélo sont complexes et emploient une terminologie variée mais également communes avec d'autres thématiques. Les classifieurs bayésiens employés se sont montrés imprécis pour reconnaître les documents ciblés par l'exploration ❷.

3.2 Exploration sélective du web sans apprentissage initial

Dans la section précédente, nous avons montré que les approches sélectives reposant sur un apprentissage initial nécessitaient un graphe d'entraînement relativement difficile à construire pour certaines recherches documentaires. Nous avons également montré que le nombre de documents pertinents contenus dans ce graphe d'entraînement était étroitement lié aux performances de l'apprentissage et donc aux performances de l'exploration sélective. C'est pour ces deux raisons que la mise en œuvre d'une telle exploration exige, dans certains cas, une intervention humaine fastidieuse pouvant engendrer un surcoût important.

Dans un souci de réduction des coûts de déploiement d'un processus d'exploration sélective, différentes approches alternatives ont vu le jour dans le domaine de l'exploration sélective du web. Ces approches préconisent la suppression de la phase d'apprentissage initial pour la construction de la fonction d'évaluation de l'intérêt des candidats. En remplacement, les approches de cette catégorie proposent de construire la précédente fonction de manière incrémentale au fur et à mesure de l'avancement du processus de l'exploration (*i.e.* en-ligne). Pour cela, ces approches présentent un ensemble de techniques, souvent statistiques, pour permettre au robot de reconnaître progressivement les bons candidats des mauvais.

Le principe de la construction en-ligne de la fonction d'évaluation de l'intérêt des candidats repose sur le cumul incrémental d'un ensemble d'informations modélisées, en général, sous forme de mesures statistiques. À un instant donné de l'exploration, ces mesures traduisent l'incidence de certaines propriétés observées dans le graphe exploré sur la qualité du parcours effectué par le robot depuis son lancement. Cette qualité est souvent résumée par l'ensemble des échecs et des réussites successives du robot dans ses rapatriements. Précisons que, dans ce contexte, un échec correspondant au rapatriement d'un document non ciblé par l'exploration et une réussite correspond au rapatriement d'un document cible. La détermination des échecs et des succès de rapatriements du robot ne peut se faire manuellement (de manière interactive par exemple) car cela entraînerait un retard important dans l'avancement du robot. Il est donc nécessaire de disposer d'une seconde fonction, qui soit fiable et précise, pour évaluer la pertinence des documents rapatriés.

Dans la littérature scientifique, nous avons identifié deux techniques majeures d'exploration sélective sans apprentissage initial : InfoSpiders [Menczer et al. 2004, Menczer 2003, Menczer 1997] et Intelligent crawling [Aggarwal et al. 2001].

3.2.1 InfoSpiders

Filippo Menczer et al. ont mis au point un robot sélectif appelé **InfoSpiders** [Menczer et al. 2004, Menczer 2003] pour explorer le web à la recherche de documents thématiques. La thématique

est décrite par un vecteur de mots clés pertinents à fournir au commencement de l'exploration.

InfoSpiders est un robot composé d'un ensemble d'agents de rapatriement. Contrairement à l'architecture centralisée montrée dans la figure IV.3 page 95 du chapitre précédent, chaque agent d'**InfoSpiders** est indépendant des autres et utilise sa propre pile d'urls. Les priorités de rapatriement de candidats sont donc locales et propres à chaque agent.

La stratégie d'exploration du web de chaque agent du robot **InfoSpiders** repose sur un algorithme évolutionnaire appelé ARACHNID [Menczer 1997] : ***A**ddaptive **R**etrieval **A**gents **C**hoosing **H**euristic **N**eighborhoods for **I**nformation **D**iscovery*.

L'algorithme ARACHNID permet à chaque agent d'estimer l'intérêt de suivre chaque lien du document que l'agent vient de rapatrier. L'intérêt d'un lien est calculé par une combinaison linéaire de deux scores. Le premier score est obtenu par le calcul de la similarité du contenu du document contenant le lien à estimer avec le vecteur de mots clés décrivant la thématique visée. Signalons que ce score est identique pour tous les liens sortant d'un même document. Lorsqu'il est élevé, ce score augmente la valeur de l'intérêt du lien ce qui incite l'agent à privilégier le suivi des liens provenant de documents très similaires à la liste de mots clés fournie en début d'exploration. Le second score est, quant à lui, calculé par un réseau de neurones monocouche. Ce réseau reçoit en entrée les fréquences des mots apparaissant dans une fenêtre fixe du voisinage du lien à estimer (k mots avant et après le lien ainsi que les mots de son ancre). La particularité de ce réseau réside dans la pondération de ses entrées. En effet, le poids associé à une entrée du réseau (fréquence d'un mot) est inversement proportionnelle à la distance qui sépare le mot correspondant à cette entrée de l'ancre du lien. Autrement dit, plus le mot est éloigné de l'ancre, plus le poids de l'entrée de sa fréquence dans le réseau de neurones est faible. En sortie, le réseau de neurones propose une valeur numérique représentant la pertinence du lien.

Chaque agent entraîne en-ligne son réseau de neurones à construire une fonction linéaire de séparation entre l'espace des liens prometteurs et l'espace des liens inintéressants. Au commencement de l'exploration, la stratégie du robot est aléatoire et le réseau de neurones d'estimation des liens est vide (poids des entrées nuls). La pertinence de chaque document rapatrié est déterminée par le calcul de similarité du contenu textuel du document avec le vecteur de mots décrivant la thématique de l'exploration. Au delà d'un seuil fixé par avance, le document est considéré comme pertinent. Ainsi, dès le rapatriement du premier document pertinent, l'agent mémorise les mots du voisinage du lien l'ayant conduit à ce document. Par un système d'apprentissage, le réseau de neurones est initialisé, puis affiné à mesure que d'autres documents pertinents sont trouvés.

Par ailleurs, **InfoSpiders** ne garde pas en service tous ses agents de départ. En effet, par un système de bonus, le robot attribue une quantité vitale à chaque agent trouvant un document pertinent. La valeur du bonus attribué à un agent est égale au score de similarité entre le contenu du document qu'il vient de rapatrier et le contenu du vecteur décrivant la thématique d'exploration. Les agents cumulant peu de bonus sont voués à disparaître. Par contre, les agents totalisant un seuil donné de cumul de bonus sont automatiquement reproduits. La reproduction consiste à partager la pile d'urls du père en deux. L'agent enfanté hérite de la moitié des candidats du père et se voit attribué une énergie (bonus) nulle. De plus, cet agent nouvellement créé reçoit un nouveau vecteur descriptif de la thématique de l'exploration. Initialisé avec les mots de la requête d'exploration, ce vecteur est enrichi avec les mots du meilleur document trouvé par le père de l'agent enfant. Précisons

que le meilleur document trouvé par le père correspond au document rapatrié ayant totalisé le plus grand score de similarité.

Les expérimentations menées par les auteurs sur trois thématiques (documents sur les compétitions de bucherons, sites sur le management et sites d'associations de consommateurs) ont montré que **InfoSpiders** est plus précis qu'un robot utilisant une approche systématique (la largeur d'abord dans ce cas). Dans leurs expérimentations, la précision est calculée par le rapport entre le nombre de documents pertinents et le nombre total de documents rapatriés. Rappelons que la pertinence correspond aux documents ayant une similarité avec le vecteur décrivant la thématique supérieure au seuil de pertinence fixé au départ de l'exploration. Cependant, **InfoSpiders** s'est montré moins pertinent qu'une exploration sélective naïve (type Fish-Search). Cette exploration repose sur une stratégie naïve qui consiste à suivre en priorité les liens provenant du document ayant le score de similarité le plus grand. La différence de performances est particulièrement marquée au début de l'exploration. Les auteurs attribuent cela au temps nécessaire aux agents d'**InfoSpiders** de mettre en place leur stratégie évolutive.

D'un point de vue général et contrairement aux approches sélectives présentées jusqu'ici (ou celle qui sera présentée à la suite), le robot **InfoSpiders** ne dispose d'aucune vision globale sur les candidats rencontrés. L'indépendance des agents de rapatriement et l'indépendance de leurs piles d'urls respectives entraînent le robot à cibler la recherche d'optimums locaux. Chaque optimum est propre aux candidats de la pile d'urls d'un agent donné. Par ailleurs, le fonctionnement du réseau neuronal de chaque agent ainsi que le fonctionnement de la méthode de calcul de l'intérêt des liens ont recours à des paramètres numériques à fixer en début d'exploration. Il n'est pas évident de choisir les meilleures valeurs pour ces paramètres, ce qui rend l'approche peu robuste.

3.2.2 Exploration « intelligente » du web (Intelligent crawling)

Dans leur article [Aggarwal et al. 2001], *Charu C. Aggarwal, Fatima Al-Garawi et Philip S. Yu* proposent une approche alternative à l'exploration ciblée du web pour rechercher à moindre coût des documents particuliers sur le web. Cette approche, appelée *exploration « intelligente » du web* ou « *intelligent crawling* », ne nécessite aucun apprentissage automatique au préalable ni aucune intervention manuelle. Le robot associé, appelé *robot « intelligent »*, est directement lancé à la recherche de documents pertinents.

Au cours d'une exploration « intelligente », chaque document rapatrié est immédiatement évalué par le robot pour déterminer sa pertinence. Le robot utilise pour cela une fonction booléenne d'évaluation notée f_D . La fonction f_D renvoie la valeur 'vrai (1)' si le contenu du document rapatrié correspond à la thématique visée. Dans le cas contraire, elle renvoie la valeur 'faux(0)'. Par exemple, pour rechercher des documents de formule 1 sur «*Michael Schumacher*», la fonction f_D peut simplement consister à tester la présence des mots «*Michael*», «*Schumacher*» et «*F1*» dans le titre du document à évaluer.

À la suite du rapatriement de chaque candidat, un robot « intelligent » s'auto-évalue en tenant compte de la valeur renvoyée par la fonction f_D sur le contenu du document qui vient d'être rapatrié. Cette auto-évaluation consiste à calculer la précision de l'exploration atteinte à la suite du dernier rapatriement. Dans ce contexte, la précision représente la proportion de documents pertinents rapatriés. Précisons que, dans le contexte d'une exploration « intelligente », les auteurs considèrent

qu'un document est pertinent lorsque son contenu est évalué à *vrai* par la fonction $f_{\mathcal{D}}$. Ainsi, la précision p_t d'un robot « intelligent » à un moment t de son exploration s'exprime comme suit :

$$p_t = \frac{|\{d \in \mathcal{D}_{ex} / f_{\mathcal{D}}(d) = 1\}|}{|\mathcal{D}_{ex}|} \quad (\text{V.14})$$

où $\mathcal{D}_{ex}$ est l'ensemble des documents rapatriés depuis le départ de l'exploration jusqu'à l'instant t .

Remarque 3.4 *Dans la suite, nous noterons l'ensemble des documents pertinents rapatriés $\{d \in \mathcal{D}_{ex} / f_{\mathcal{D}}(d) = 1\}$ par $\mathcal{D}_{ex}^C$*

La précision p_t calculée à un instant t de l'exploration d'un robot intelligent peut être interprétée comme étant la capacité de ce robot à rapatrier un document pertinent à l'instant $t + 1$. Autrement dit, le robot possède $p_t\%$ de chances de rapatrier un document pertinent à l'instant $t + 1$. L'objectif visé consiste donc à rapatrier, à chaque fois, le candidat susceptible d'améliorer la précision du robot intelligent.

Pour atteindre cet objectif, l'approche « intelligente » de *Charu C. Aggarwal et al.* propose de mettre en œuvre une technique de prédiction de l'intérêt du rapatriement de chaque candidat rencontré par le robot. Le principe de cette technique consiste à prédire l'intérêt du rapatriement d'un candidat donné à partir des propriétés observées dans son voisinage. Les auteurs proposent de s'intéresser à quatre propriétés : les mots contenus dans les parents rapatriés du candidat (pr_1), les mots composant l'url du candidat (pr_2), la pertinence des parents rapatriés du candidat (pr_3) et la pertinence des frères rapatriés du candidat (pr_4).

Remarque 3.5 *Pour des besoins thématiques particuliers, d'autres propriétés peuvent être rajoutées aux quatre propriétés précédentes si leur observation s'avère être déterminante dans la prédiction de la pertinence des candidats. Ceci n'affecte en rien la validité de l'approche des auteurs que nous présentons ci-dessous.*

L'intérêt de rapatriement d'un candidat est calculée par une mesure appelée *le ratio d'intérêt (lift)*.

Définition 4 (ratio d'intérêt) *Le ratio d'intérêt d'un événement E dans la détermination de l'événement C , noté $I(C, E)$, est défini par la formule suivante :*

$$I(C, E) = \frac{P(C/E)}{P(C)} \quad (\text{V.15})$$

En considérant que l'événement E soit un événement composé de n événements élémentaires indépendants, i.e. $E = \bigcap_{i=1}^n E_i$, le ratio précédent s'exprime comme suit :

$$\begin{aligned}
I(C, \bigcap_{i=1}^n E_i) &= \frac{P(C / \bigcap_{i=1}^n E_i)}{P(C)} \\
&= \frac{\frac{P(C \cap (\bigcap_{i=1}^n E_i))}{P(\bigcap_{i=1}^n E_i)}}{P(C)} \\
&= \frac{P(\bigcap_{i=1}^n (C \cap E_i))}{P(\bigcap_{i=1}^n E_i) \times P(C)} && \text{distributivité de l'intersection sur elle même} \\
&= \frac{\prod_{i=1}^n P(C \cap E_i)}{\prod_{i=1}^n P(E_i) \times P(C)} && \text{hypothèse d'indépendance} \\
&= \frac{\prod_{i=1}^n P(C \cap E_i) \times P(C)^{n-1}}{\prod_{i=1}^n P(E_i) \times P(C) \times P(C)^{n-1}} \\
&= \prod_{i=1}^n \frac{P(C \cap E_i)}{P(C)} \times P(C)^{n-1} \\
&= \prod_{i=1}^n I(C, E_i) \times P(C)^{n-1} \\
I(C, \bigcap_{i=1}^n E_i) &\propto \prod_{i=1}^n I(C, E_i) \tag{V.16}
\end{aligned}$$

Le ratio d'intérêt de deux événements C et E permet d'apprécier l'apport de l'occurrence de l'événement E dans la réalisation de l'événement C . Autrement dit, le ratio peut être vu comme un coefficient multiplicateur qui permet de déduire les chances de réalisation de l'événement C à partir des chances d'observation de la propriété E . Par exemple, pour un ratio d'intérêt $I(C, E) = 2$, on dit que les individus ayant la propriété E possèdent deux fois plus de chances de satisfaire C que les autres individus [Azé 2003].

Notations Dans le contexte d'une exploration « intelligente », l'événement C correspond au fait que «le robot rapatrie un document pertinent» et l'événement $E_{i,i=1..4}$ correspond au fait que «la propriété ' pr_i ' est observée chez un candidat donné». Le ratio d'intérêt d'une propriété ' pr_i ' du voisinage d'un candidat d dans la réalisation de l'événement C est noté $I(C, E_i) \equiv I_{pr_i}[d]$. L'intérêt d'un candidat d est noté $I[d]$.

L'estimation de l'intérêt $I[d]$ du rapatriement d'un candidat donné d consiste à combiner le calcul des ratios d'intérêt $I_{pr_i,i=1..4}[d]$ de l'observation de chaque propriété ' $pr_{i,i=1..4}$ ' du voisinage de d dans la réalisation de l'objectif du robot C (rapatrier un document pertinent). Ceci revient à mesurer l'influence de chaque valeur de la propriété ' pr_i ' du voisinage de d dans la réalisation de l'événement C et cela à partir des succès du robot dans ses rapatriements antérieurs (depuis le départ de l'exploration). Rappelons que, dans le contexte d'une exploration « intelligente », un succès correspond au rapatriement d'un document d tel que $f_D(d) = 1$. Ainsi, en considérant que Φ est une fonction permettant de combiner ses paramètres, l'intérêt $I[d]$ s'exprime comme suit :

$$I[d] = \Phi(I_{pr_1}(d), I_{pr_2}(d), I_{pr_3}(d), I_{pr_4}(d)) \tag{V.17}$$

Exemple : Les auteurs ont choisi dans leur article la combinaison suivante :

$$I[d] = \lambda_1 \cdot \log(I_{pr_1}(d)) + \lambda_2 \cdot \log(I_{pr_2}(d)) + \lambda_3 \cdot \log(I_{pr_3}(d)) + \lambda_4 \cdot \log(I_{pr_4}(d)) \quad (\text{V.18})$$

Calcul du ratio d'intérêt du contenu textuel des parents d'un candidat (I_{pr_1}) Pour toute évaluation à 'vrai' (1) d'un candidat d téléchargé à l'itération i , le robot constitue un ensemble de mots à partir du contenu textuel des parents de ce candidat. Ces mots forment l'ensemble $\mathcal{W}_d^i$ défini comme suit :

$$\mathcal{W}_d^i = \begin{cases} \{w_y \in d' / (\exists l \in \mathcal{L} : d' \xrightarrow{l} d) \wedge (f_{\mathcal{D}}(d) = 1)\} \\ \emptyset \text{ si } f_{\mathcal{D}} = 0 \end{cases} \quad (\text{V.19})$$

À chaque itération i du robot débouchant sur un succès, un vocabulaire de référence $\mathcal{V}^i$ est mis à jour. Au commencement de l'exploration, ce vocabulaire est vide. Son initialisation s'effectue au moment du premier succès de rapatriement du robot. Le vocabulaire se voit attribué les mots du document pertinent rapatrié. À chaque document pertinent d' supplémentaire, le vocabulaire s'enrichit des mots du contenu textuel d' . Ce processus est résumé par l'équation ci-dessous :

$$\mathcal{V}^i = \bigcup_{x=1}^i \mathcal{W}_d^x = \mathcal{V}^{i-1} \cup \mathcal{W}_d^i \quad (\text{V.20})$$

Remarque 3.6 Lors d'un échec de rapatriement (téléchargement d'un document non pertinent), aucun cumul de mots n'est effectué pour le vocabulaire $\mathcal{V}^i$ car $\mathcal{W}_d^i = \emptyset$.

Le vocabulaire $\mathcal{V}^i$ peut être vu comme un moyen pour le robot de mémoriser les mots contenus dans les documents ayant pointé sur un candidat pertinent. Ainsi, à l'avenir (itération $i+1$ et plus), la présence d'un certain nombre des mots du vocabulaire $\mathcal{V}^i$ dans le contenu des parents d'un candidat donné peut être interprété comme un indicateur sûr de la pertinence du candidat. Autrement dit, le robot s'appuie sur son expérience pour privilégier logiquement le rapatriement du candidat pointé par des parents ayant un contenu riche en mots déjà vus dans les parents d'autres candidats pertinents (*i.e.* le vocabulaire $\mathcal{V}^i$).

Soit Q_y l'événement : «le mot w_y du vocabulaire $\mathcal{V}^i$ est présent dans l'un des parents du candidat d ». Soit également M l'ensemble des indices y pour lesquels l'événement Q_y est vrai, *i.e.* $M = \{y / Q_y \text{ est vrai}\}$. Le ratio d'intérêt I_{pr_1} pour un candidat d donné est calculé comme suit :

$$\begin{aligned} I_{pr_1}(d) &= I(C, \bigcap_{y \in M} Q_y) \\ &= \frac{P(C / [Q_1 \cap Q_2 \dots \cap Q_{|M|}])}{P(C)} \\ &= \frac{P(C \cap [Q_1 \cap Q_2 \dots \cap Q_{|M|}])}{P([Q_1 \cap Q_2 \dots \cap Q_{|M|}])} \times \frac{1}{P(C)} \end{aligned}$$

En supposant l'indépendance des occurrences des événements Q_y , la formule précédente se réécrit de la manière suivante :

$$\begin{aligned}
I_{pr_1}(d) &= \frac{P((C \cap Q_1) \cap (C \cap Q_2) \dots (C \cap Q_{|M|}))}{P(Q_1) \cdot P(Q_2) \dots P(Q_{|M|})} \times \frac{P(C)^{|M|-1}}{P(C) \cdot P(C)^{|M|-1}} \\
&= \frac{P(C \cap Q_1)}{P(Q_1) \times P(C)} \times \frac{P(C \cap Q_2)}{P(Q_2) \cdot P(C)} \times \dots \times \frac{P(C \cap Q_{|M|})}{P(Q_{|M|}) \cdot P(C)} \times P(C)^{|M|-1} \\
&= \prod_{y \in M} I(C, Q_y) \times P(C)^{|M|-1}
\end{aligned}$$

La valeur du terme $P(C)^{|M|-1}$ étant la même pour tous les calculs d'intérêt des autres candidats, le ratio I_{pr_1} est proportionnel au produit des ratios d'intérêts de chaque événement Q_y pris séparément, c'est-à-dire :

$$I_{pr_1}(d) \propto \prod_{y \in M} I(C, Q_y) \quad (\text{V.21})$$

Le ratio d'intérêt de l'événement atomique Q_y par rapport à la détermination de l'événement C est donné par la formule V.15 page 122 :

$$\begin{aligned}
I(C, Q_y) &= \frac{P(C/Q_y)}{P(C)} \\
&= \frac{P(C \cap Q_y)}{P(Q_y) \times P(C)}
\end{aligned} \quad (\text{V.22})$$

Pour calculer la valeur de $I(C, Q_y)$, un robot « intelligent » procède à une approximation des probabilités $P(C \cap Q_y)$, $P(Q_y)$ et $P(C)$ par les formules suivantes :

$$\begin{aligned}
P(C \cap Q_y) &\simeq \frac{\# \text{ de documents } \in \mathcal{D}_{ex}^C \text{ ayant le mot } w_y \text{ dans le contenu de l'un de leurs parents}}{\# \text{ de documents } \in \mathcal{D}_{ex}} \\
P(Q_y) &\simeq \frac{\# \text{ de documents } \in \mathcal{D}_{ex} \text{ ayant le mot } w_y \text{ dans le contenu de l'un de leurs parents}}{\# \text{ de documents } \in \mathcal{D}_{ex}} \\
P(C) &\simeq p_i = \frac{\# \text{ de documents } \in \mathcal{D}_{ex}^C}{\# \text{ de documents } \in \mathcal{D}_{ex}}
\end{aligned} \quad (\text{V.23})$$

Pour mettre en œuvre les approximations précédentes, un robot « intelligent » dédie 4 variables de comptage :

1. $n_{w_y}^C$: variable de comptabilisation du nombre de documents pertinents rapatriés contenant le mot w_y dans le contenu de l'un de leurs parents ;
2. n_{w_y} : variable de comptage du nombre de documents contenant chez leurs parents le mot w_y ;
3. n_i : variable de comptage du nombre de documents rapatriés depuis le début de l'exploration ;
4. n_i^C : variable de comptage du nombre de documents pertinents rapatriés depuis le début de l'exploration.

Ces variables sont initialisées à zéro au début de l'exploration et sont incrémentées par pas de 1 pour refléter l'état courant des valeurs des différentes approximations précédentes. Ainsi, le ratio d'intérêt d'un candidat d par rapport à la propriété pr_1 se calcule par la formule suivante :

$$\begin{aligned}
I(C, Q_y) &\simeq \frac{\frac{n_{w_y}^C}{n_i^C}}{\frac{n_{w_y}}{n_i} \times \frac{n_i^C}{n_i}} \\
&\simeq \frac{n_{w_y}^C}{n_{w_y}} \times \frac{n_i}{n_i^C}
\end{aligned}$$

Le terme $\frac{n_i}{n_i^C}$ étant commun à tous $I(C, Q_{y,y \in M})$, le ratio d'intérêt $I(C, Q_y)$ est déterminé par la formule suivante :

$$I(C, Q_y) \propto \frac{n_{w_y}^C}{n_{w_y}} \quad (\text{V.24})$$

Calcul du ratio d'intérêt du contenu de l'url d'un candidat (I_{pr_2}) Ce ratio se calcule de la même manière que le ratio d'intérêt précédent (I_{pr_1}) mais avec un vocabulaire de référence différent. En effet, le robot constitue, en parallèle au vocabulaire $\mathcal{V}^i$, un vocabulaire spécifique aux termes contenus dans les urls des candidats ayant été évalués à 'vrai' à la suite de leur rapatriement. Nous notons ce vocabulaire par $\mathcal{V}_{url}^i$.

Soit U_y l'événement : «le mot w_y du vocabulaire $\mathcal{V}_{url}^i$ est présent dans l'url du candidat d ». Soit également M l'ensemble des indices y pour lesquels l'événement U_y est vrai, *i.e.* $M = \{y / U_y \text{ est vrai}\}$. Le ratio d'intérêt de la présence des mots du vocabulaire $\mathcal{V}_{url}^i$ dans l'url d'un candidat d est :

$$I_{pr_2}(d) = I(C, \bigcap_{y \in M} U_y)$$

D'après la formule V.21 page 125, ce ratio est proportionnel au produit des ratios séparés des événements U_y :

$$I_{pr_2}(d) \propto \prod_{y \in M} I(C, U_y) \quad (\text{V.25})$$

Pour un événement U_y , la formule V.22 page 125 s'écrit :

$$I(C, U_y) = \frac{P(C \cap U_y)}{P(U_y) \times P(C)}$$

Deux variables supplémentaires sont également requises pour estimer la probabilité conjointe des événements C et U_y ainsi que pour estimer la probabilité $P(U_y)$:

1. $n_{u_y}^C$: variable de comptage du nombre de documents pertinents contenant le mot u_y dans leur url ;
2. n_{u_y} : variable de comptage du nombre de documents rapatriés contenant le mot u_y dans leur url.

En s'appuyant sur la formule V.24 page 126, le ratio d'intérêt de l'événement U_y dans la détermination de l'événement C s'écrit alors comme suit :

$$I(C, U_y) \propto \frac{n_{u_y}^C}{n_{u_y}} \quad (\text{V.26})$$

Calcul du ratio d'intérêt de la pertinence des parents d'un candidat (I_{pr_3}) L'analyse des valeurs de la propriété pr_3 chez les documents pertinents rapatriés permet au robot de prédire la pertinence de ses futurs candidats à partir de la pertinence (ou de la non pertinence) de leurs parents. Cette analyse permet également de découvrir l'existence (ou l'absence) d'une proximité «géographique» des documents recherchés par le robot dans le graphe du web.

Exemple : Supposons qu'à l'itération i le robot ait rapatrié 1000 documents dont 100 documents sont pertinents. La précision du robot à cet instant est : $P(C) = \frac{100}{1000} = 0.1$.

Si la répartition des documents web était complètement aléatoire, la probabilité pour qu'un lien quelconque du graphe exploré relie deux documents pertinents serait de : $P(C) \times P(C) = 0.01$. De même, la probabilité qu'un lien quelconque du graphe exploré relie un parent non pertinent à un document pertinent est de : $P(C) \times (1 - P(C)) = 0.09$.

Supposons que dans le graphe exploré, la proportion constatée de liens reliant deux documents pertinents est de : $prop_1 = 0.07$ et la proportion de liens reliant un parent non pertinent à un document pertinent est de : $prop_2 = 0.02$.

La proportion $prop_1$ étant plus importante que la proportion $prop_2$, le robot constate une relative proximité entre les documents pertinents de cette exploration dans le graphe web. Ceci lui permet de conclure que la pertinence de tous les parents d'un candidat donné augmente de $\frac{0.07}{0.01} = 7$ fois les chances du robot d'enregistrer un succès en rapatriant le candidat en question.

Soit L^p l'événement : «le candidat d est pointé par un parent pertinent». Soit également l'événement L^{np} : «le candidat d est pointé par un parent non pertinent». Supposons que le candidat d est pointé par k documents pertinents et par m documents non pertinents. Le calcul du ratio d'intérêt des parents du candidat d dans la détermination de la pertinence du candidat à rapatrier est effectué comme suit :

$$\begin{aligned} I_{pr_3}(d) &= I(C, \left[\left(\bigcap_{t=1}^k L_t^p \right) \cap \left(\bigcap_{t=1}^m L_t^{np} \right) \right]) \\ &\propto \prod_{t=1}^k I(C, L_t^p) \times \prod_{t=1}^m I(C, L_t^{np}) \end{aligned}$$

Le ratio d'intérêt de deux parents pertinents différents est le même car cette quantité dépend de la pertinence du parent et non du parent lui-même. Ainsi, la précédente équation devient :

$$I_{pr_3}(d) \propto I(C, L^p)^k \times I(C, L^{np})^m$$

En considérant $prop_1$ la proportion constatée de liens reliant deux documents pertinents du graphe exploré et $prop_2$ la proportion constatée de liens reliant un document non pertinent à un document pertinent du graphe exploré, les ratios $I(C, L^p)$ et $I(C, L^{np})$ sont calculés comme suit :

$$\begin{aligned} I(C, L^p) &= \frac{prop_1}{P(C) \cdot P(C)} \\ I(C, L^{np}) &= \frac{prop_2}{P(C) \cdot (1 - P(C))} \end{aligned}$$

Pour le calcul des deux proportions $prop_1$ et $prop_2$, le robot utilise trois variables de comptage :

1. N_l : variable de comptage du nombre de liens dans le graphe exploré ;
2. N_l^{pp} : variable de comptage du nombre de liens dans le graphe exploré reliant deux documents pertinents entre eux ;
3. N_l^{np} : variable de comptage du nombre de liens dans le graphe exploré reliant un document non pertinent à un document pertinent.

$$I_{pr3}(d) \simeq \left(\frac{N_l^{pp}}{N_l \cdot \frac{n_i^C}{n_i} \cdot \frac{n_i^C}{n_i}} \right)^k \times \left(\frac{N_l^{np}}{N_l \cdot \frac{n_i^C}{n_i} \cdot (1 - \frac{n_i^C}{n_i})} \right)^m \quad (\text{V.27})$$

Calcul du ratio d'intérêt de la pertinence des frères d'un candidat (I_{pr4}) L'analyse de la pertinence des frères des candidats pertinents antérieurs permet au robot d'estimer la pertinence du futur candidat à rapatrier à partir seulement de la pertinence de ses frères déjà rapatriés.

Soit f_d^p le nombre de frères pertinents du document d parmi f_d frères au total. Le ratio I_{pr4} est alors défini par le rapport entre le nombre constaté de frères pertinents dans le graphe déjà exploré et le nombre attendu de frères pertinents :

$$I_{pr4} \simeq \frac{f_d^p}{f_d \times P(C)} \quad (\text{V.28})$$

Exemple : Soit la figure V.4 page 129 illustrant 8 documents rapatriés par un robot « intelligent » et deux candidats non encore rapatriés reliés à ces documents rapatriés.

Le candidat ⑦ possède 5 frères déjà rapatriés dont 4 sont évalués à 'vrai' (pertinents). Supposons que la précision $P(C)$ du robot à cet instant est de 0.25.

Si le web était complètement aléatoire, le nombre attendu de frères pertinents pour le candidat ⑦ serait de : $0.2 \times 5 = 1$. Or, sur le graphe exploré, ce candidat possède déjà 4 frères pertinents. Ceci peut s'expliquer par une connexion du candidat ⑦ à un parent *hub* [Kleinberg 1999]. Il est donc fort probable que le candidat ⑦ soit pertinent au vu des bons résultats provenant de son père. Le ratio d'intérêt de cette connaissance dans la détermination de la pertinence du candidat rapatrié est : $\frac{4}{1} = 4$. Autrement dit, la chance de rapatrier un candidat pertinent à partir de candidats présentant au minimum 4 frères pertinents est multipliée par au moins 4.

La stratégie d'exploration Comme pour les approches précédentes, la fonction d'estimation de l'intérêt des candidats d'une exploration « intelligente » permet au robot intelligent de rapatrier le candidat ayant le ratio d'intérêt le plus élevé. Pour le ce candidat, l'importance de son ratio traduit une présence importante des propriétés ayant favorisé le rapatriement de documents cibles depuis le début de l'exploration. A posteriori, l'observation de ces mêmes propriétés chez le candidat en question devrait déboucher sur un succès de rapatriement.

En notant par $\mathcal{D}_{candidat}^i$ l'ensemble des candidats se présentant à l'itération i , la fonction d'estimation de l'intérêt de rapatriement des candidats, notée $\mathcal{F}_{candidat}$, s'exprime comme suit :

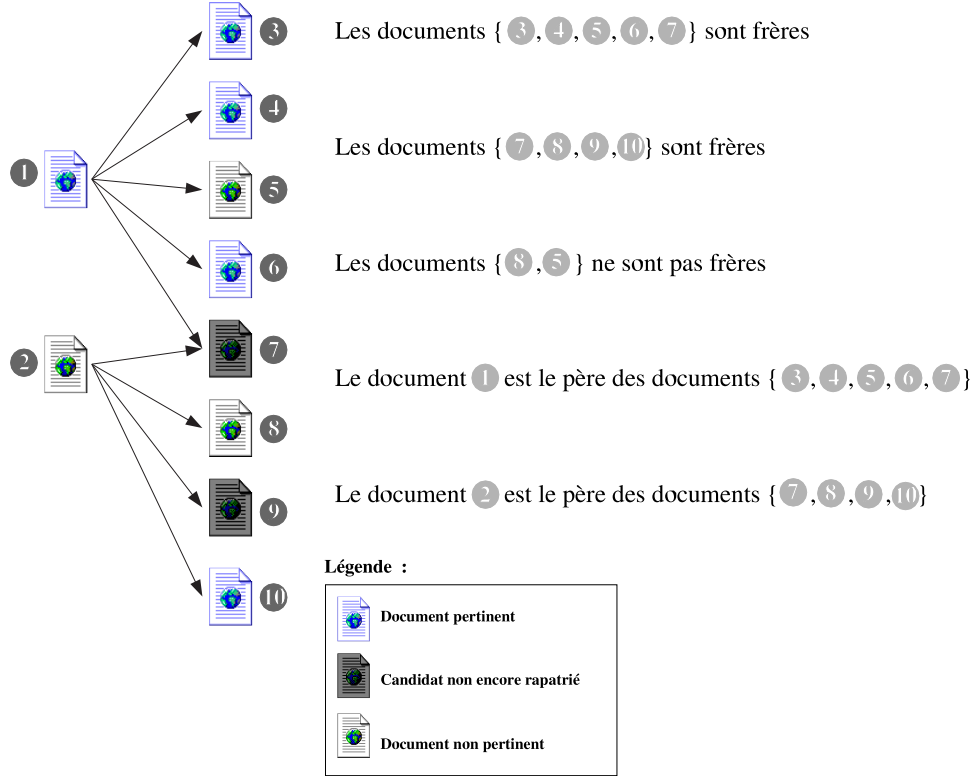


FIG. V.4 – Parents et frères d'un candidat

$$\mathcal{F}_{candidat} : \begin{array}{l} \mathcal{D}_{candidat}^i \longrightarrow \mathbb{R} \\ d \longrightarrow \mathcal{F}_{candidat}(d) = I[d] \end{array} \quad (\text{V.29})$$

Déroulement d'une exploration « intelligente » Partant d'un ensemble d'urls de départ, un robot « intelligent » débute son parcours par une exploration systématique (aléatoire ou en largeur d'abord par exemple) car à ce stade la stratégie sélective du robot n'est pas encore opérationnelle. Ceci s'explique par l'impossibilité d'utiliser la fonction $\mathcal{F}_{candidat}$ tant que les variables de l'ensemble de statistiques $\mathcal{K}_i$ ne sont pas mises à jour. Autrement dit, tant que le premier candidat pertinent n'ait pas été trouvé, le robot est incapable de discriminer « objectivement » les candidats qui se présentent à lui. Pouvant durer longtemps, ce contretemps constitue un handicap majeur de l'exploration « intelligente » du web. Il met également en évidence l'importance du choix des points de départ pour éviter une telle situation.

Dès la découverte des premiers candidats pertinents, le robot incrémente progressivement les variables de son ensemble de statistiques $\mathcal{K}_i$. La fonction $\mathcal{F}_{candidat}$ devient alors opérationnelle en permettant de choisir pour la prochaine itération ($i + 1$) du robot le candidat d^* tel que :

$$d^* = \arg \max_{d \in \mathcal{D}_{candidat}^{i+1}} \mathcal{F}_{candidat}(d, \mathcal{K}_i) \quad (\text{V.30})$$

Discussion L'approche « intelligente » [Aggarwal et al. 2001] proposée par *Charu C. Aggarwal, Fatima Al-Garawi et Philip S. Yu* nous semble parfaitement adaptée à la mise en œuvre de recherches thématiques de documents sur le web à moindre frais (sans apprentissage initial). Son avantage majeur réside dans l'apprentissage fait en-ligne pour mesurer l'incidence de certains facteurs et propriétés sur la pertinence des candidats. De plus, grâce aux propriétés ' pr_3 ' et ' pr_4 ', le robot est capable de déterminer si les documents recherchés sont « géographiquement » proches les uns des autres dans le graphe du web.

Par ailleurs, la souplesse de la combinaison des différentes propriétés dans le calcul de l'intérêt de rapatriement d'un candidat permet à l'administrateur d'un robot de paramétrer l'exploration de sorte à privilégier une ou plusieurs propriétés par rapport aux autres propriétés restantes. Ceci s'effectue, par exemple, par renforcement des coefficients des intérêts des propriétés à privilégier dans le calcul de l'intérêt global d'un candidat (voir la formule V.18 page 124). La souplesse de la combinaison de calcul de l'intérêt est particulièrement utile lorsqu'il s'avère que la ou les propriétés privilégiées sont déterminantes dans l'estimation de la pertinence d'un document.

Toutefois, nous estimons que la qualité du calcul d'intérêt d'un candidat donné dépend en grande partie de l'exactitude de la fonction d'évaluation f_D à déterminer la pertinence d'un document. Dans leurs expérimentations, les auteurs ont testé leur approche sur des fonctions d'évaluations très simples. Ces fonctions se résument à des tests de présence de mots prédéfinis dans le contenu des documents rapatriés. Pour des besoins de recherche plus complexes, il est difficile de formuler une fonction f_D aussi précise qu'un simple test de présence de mots dans le contenu. Les estimations de la fonction $\mathcal{F}_{candidat}$ peuvent ainsi être affectées par l'imprécision de la fonction d'évaluation de documents.

4 Conclusion

Nous avons présenté dans cet état de l'art les différentes techniques d'exploration du web à la recherche de documents. Ces différentes techniques sont classées et montrées dans la figure V.5 page 132. Dans cette classification, nous avons distingué trois approches principales en fonction de leur manière de parcourir le graphe du web : les explorations systématiques, les explorations reposant sur une stratégie de type « le meilleur d'abord » et les explorations sélectives guidées par des critères thématiques.

Une exploration systématique du web s'appuie sur des stratégies non informée (largeur d'abord, profondeur, ...) pour parcourir le plus possible de documents du graphe du web issu des points de départ. N'ayant aucun critère thématique particulier pour choisir les candidats à rapatrier, cette approche est appropriée pour alimenter des index de moteurs de recherche généralistes (non spécialisés dans une thématique précise).

Une exploration par stratégie informée de type « le meilleur d'abord » utilise une fonction d'évaluation de candidats fournie par avance. Grâce à cette fonction, le robot explorateur est capable de guider son parcours dans le web en privilégiant le rapatriement des « meilleurs » candidats. Cette approche peut être envisageable pour initier des explorations à la recherche de documents satisfaisant des critères simples. Pour des critères complexes (la recherche de documents thématiques par exemple), nous avons montré que le choix de la fonction d'évaluation des candidats est problématique. Le recours à l'apprentissage automatique (apprentissage bayésien par exemple) est nécessaire pour construire une fonction d'évaluation fiable et adaptée à la recherche de documents thématiques complexes.

Une exploration sélective focalise le parcours du robot sur le rapatriement de documents pertinents par rapport à une thématique donnée, souvent complexe à caractériser. L'objectif attendu est de rapatrier le plus de documents pertinents en minimisant le parcours inutile de documents non pertinents. Une exploration sélective utilise à cet effet une fonction d'évaluation de l'intérêt du rapatriement de chaque candidat. Contrairement à la fonction d'évaluation des candidats de l'approche « le meilleur d'abord », la fonction d'évaluation des candidats de l'approche sélective repose sur un processus d'apprentissage automatique qu'il est possible d'effectuer de deux manières différentes : sur un graphe d'entraînement avant le lancement de l'exploration ou en-ligne en s'appuyant sur des mesures statistiques recueillies depuis le graphe exploré.

Les approches sélectives avec apprentissage initial traduisent l'incidence de la présence de certaines propriétés (des mots clés dans les liens entrants par exemple) dans le voisinage des documents cibles du graphe d'entraînement par des probabilités. Ces probabilités sont par la suite utilisées par le robot pour estimer l'intérêt des candidats à partir de leur voisinage. La fiabilité de ces approches est étroitement liée aux caractéristiques du graphe d'entraînement utilisé. Pour une fiabilité et une précision optimales, ce graphe doit contenir suffisamment d'exemples représentatifs des documents ciblés par l'exploration. Nous avons montré que la construction d'un tel graphe n'est pas toujours évidente. Pour certaines recherches, cette phase nécessite une intervention manuelle fastidieuse et coûteuse en temps.

Les approches sélectives sans apprentissage initial permettent de s'affranchir de la phase de construction d'un graphe d'entraînement. Ces approches proposent de construire la fonction d'esti-

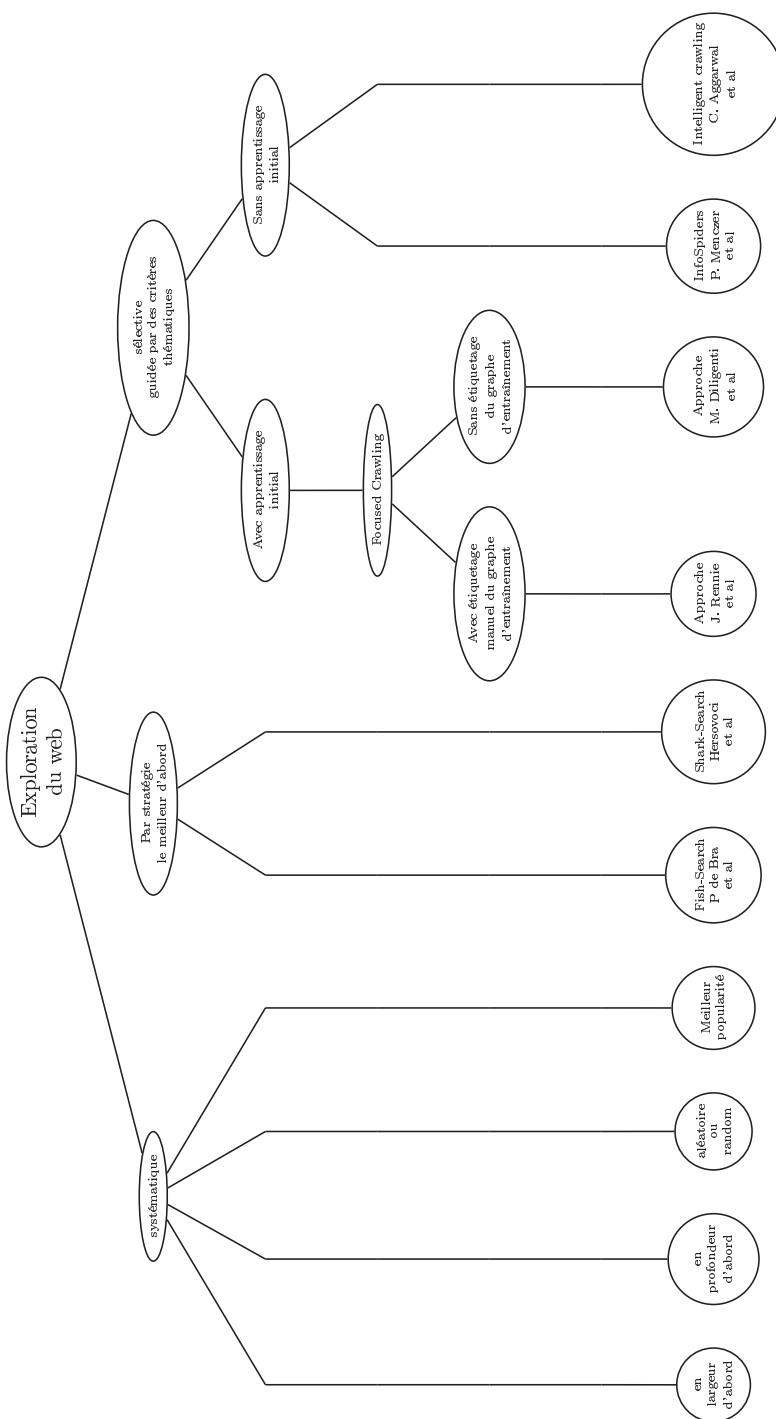


FIG. V.5 – Classification des différentes techniques d'exploration du web

mation de l'intérêt des candidats en-ligne à partir d'un ensemble de mesures statistiques recueillies depuis le graphe exploré. Contrairement à une fonction apprise à partir d'un graphe d'entraînement (approche sélective avec apprentissage initial), la fonction d'estimation des candidats de cette approche évolue au cours de l'exploration car les mesures statistiques précédentes sont régulièrement mises à jour pour répercuter les changements survenus sur le graphe exploré à la suite de chaque nouveau rapatriement de document.

Pour l'exploration « intelligente » par exemple, la construction de la fonction d'estimation des candidats se fait pendant l'exploration par cumul progressif de statistiques. Ces statistiques traduisent la fréquence d'occurrences de certaines propriétés depuis le début de l'exploration. Utilisées dans un calcul de ratio d'intérêt, ces mesures permettent d'établir le lien entre les chances de rapatrier un document cible et l'observation des propriétés précédentes dans le voisinage d'un candidat. Nous avons montré la forte dépendance entre la précision de cette approche et la fiabilité de la fonction booléenne utilisée pour évaluer les documents rapatriés. Pour des thématiques complexes, il n'est pas facile de caractériser, de manière exacte, les documents cibles par une formule booléenne. Nous avons, par ailleurs, montré qu'une exploration « intelligente » n'est pas immédiatement opérationnelle lors de son commencement. En effet, le calcul du ratio d'intérêt n'est effectif qu'une fois le premier document cible rapatrié.

Dans le prochain chapitre, nous présentons notre approche d'exploration du web intitulée *exploration sélective guidée par une requête WeQueL*. Notre approche utilise une requête WeQueL pour guider son exploration. L'objectif visé est de rapatrier le plus possible de documents totalisant un score élevé de la requête d'exploration. Comme l'approche « intelligente », notre approche ne nécessite aucun apprentissage initial.

Exploration sélective du web guidée par une requête WeQueL

Dans ce chapitre, nous présentons notre approche d'exploration sélective du web intitulée *exploration sélective guidée par une requête WeQueL*. L'objectif de notre approche est de rechercher rapidement le plus possible de documents thématiques décrits par une requête WeQueL. Autrement dit, l'approche vise à rapatrier les documents totalisant un score d'évaluation élevé sur la requête WeQueL décrivant la thématique à cibler. Nous appelons cette requête *la requête d'exploration*.

Notre approche consiste à apprendre progressivement à estimer le score d'évaluation de chaque candidat sur la requête d'exploration. Précisons que cette estimation porte sur le score d'évaluation du contenu du candidat sur une requête WeQueL alors que ce contenu n'est pas encore rapatrié. Ceci est rendu possible grâce à un système de calcul de probabilités qui repose sur le voisinage du candidat à estimer et sur un ensemble de mesures statistiques recueillies pendant l'exploration. Ainsi, l'estimation calculée peut être vue comme étant une approximation vraisemblable du score effectif du candidat sur la requête d'exploration.

Partant des estimations faites sur chaque candidat, notre stratégie privilégiée, à chaque itération du robot, le rapatriement du candidat dont le score a été approximé avec la plus forte valeur (*i.e* susceptible de correspondre au mieux aux spécifications de la requête d'exploration).

Le score effectif du document venant d'être rapatrié est obtenu par évaluation de son contenu sur la requête d'exploration. La comparaison de ce score avec le score estimé, lorsque ce document était un candidat, permet à notre approche de mettre à jour un ensemble de mesures statistiques pour affiner progressivement les futures estimations de candidats.

Nous avons organisé ce chapitre en trois sections. Dans la première section, nous motivons le développement de notre approche en la situant par rapport aux approches sélectives existantes. Nous décrivons par la suite le principe d'une exploration guidée par une requête WeQueL. La dernière

section est dédiée aux résultats expérimentaux de notre approche sur différentes requêtes WeQueL. Le protocole expérimental suivi y est décrit en détail.

1 Présentation générale de notre approche

Dans l'état de l'art sur les différentes techniques d'exploration du web présentées dans le chapitre précédent (voir chapitre V page 103), nous avons distingué deux approches majeures : les techniques d'exploration systématique du web utilisées par la plupart des moteurs de recherche généralistes (Google par exemple) et les techniques sélectives.

Nous avons montré dans le chapitre précédent que les approches systématiques étaient faciles à mettre en œuvre mais qu'elles étaient en contre partie très coûteuses en temps d'exploration et en espace mémoire/stockage. N'étant pas spécifiquement conçues pour rechercher des documents thématiques, ces techniques sont inadaptées aux besoins des nouvelles applications émergentes autour du web : moteurs de recherche spécialisés, entrepôts thématiques, *etc.* Pour ce type d'applications, le recours aux approches de type le « meilleur d'abord » ou aux approches sélectives guidées par des critères thématiques permet d'obtenir un meilleur compromis entre le volume de documents à parcourir/traiter et le nombre de documents cibles retrouvés. Nous avons cependant souligné, pour ce type d'approches, la difficulté à élaborer et mettre en place une fonction fiable et objective pour évaluer l'intérêt du rapatriement d'un candidat. En effet, pour les approches de type « le meilleur d'abord », l'utilisation d'une fonction heuristique prédéfinie pour l'évaluation de l'intérêt d'un candidat ne permet pas d'atteindre une précision satisfaisante de l'exploration. Pour les approches sélectives nécessitant un apprentissage initial, la difficulté majeure réside dans la construction d'un graphe d'entraînement suffisamment représentatif des documents cibles pour assurer une précision satisfaisante de l'exploration. Pour l'approche « intelligent crawling » [Aggarwal et al. 2001] sans apprentissage initial, les mesures statistiques cumulées tout au long du processus d'exploration permettent de mettre en place une fonction précise et efficace pour évaluer l'intérêt de rapatrier un candidat donné. Cependant, la fiabilité de cette fonction est étroitement liée à la précision de la fonction booléenne utilisée pour évaluer la pertinence des documents rapatriés. Or, pour les thématiques complexes, il n'est pas évident de définir une fonction booléenne suffisamment précise pour caractériser les documents cibles.

Afin de contourner ces difficultés, nous avons opté pour une approche hybride sans apprentissage initial combinant le principe de la mise à jour de mesures statistiques (comme dans « l'intelligent crawling ») mais exploitées à la manière de *Diligenti et al* [Diligenti et al. 2000] pour estimer le score d'évaluation de chaque candidat par une approche bayésienne (*Naive Bayes*).

À la différence de l'approche de *Diligenti et al* [Diligenti et al. 2000], nous ne construisons pas de graphe d'entraînement. Ainsi, au départ de l'exploration, notre approche ne dispose d'aucun élément pour distinguer les candidats prometteurs des autres. Pour compenser cette absence, nous récoltons, en-ligne, un certain nombre de mesures statistiques dans le but de compter les fréquences d'occurrences de certaines propriétés dans le graphe exploré. Par un calcul bayésien de probabilités, nous générons à partir des mesures statistiques précédentes un ensemble d'estimations de probabilités traduisant l'influence des propriétés observées dans le graphe exploré sur la détermination des scores d'évaluation des documents. Nous exploitons ces probabilités pour estimer le score d'évaluation de chaque candidat de l'exploration à partir des propriétés de son voisinage.

À la différence de l'approche « intelligente » [Aggarwal et al. 2001], nous ne calculons pas la pertinence d'un document rapatrié par une fonction booléenne mais par une évaluation plus souple produisant un score numérique. Plus précisément, nous évaluons le contenu de chaque document rapatrié sur la requête WeQueL d'exploration. Un document rapatrié est considéré comme pertinent si son score est supérieur à un seuil fixé. L'étude expérimentale réalisée sur le langage WeQueL (cf. section 5 du chapitre III page 68) nous permet de fixer ce seuil. Par exemple, pour la thématique du risque alimentaire, nous avons expérimentalement vérifié que 90% des documents ayant obtenu un score supérieur à 0,14 étaient effectivement pertinents. Un seuil égal à 0,14 peut alors être envisagé pour déterminer la pertinence des documents rapatriés par une exploration guidée par la requête conçue pour le risque alimentaire.

L'objectif de notre approche consiste à parcourir le web à la recherche du maximum possible de documents satisfaisant un critère de pertinence défini par : une requête WeQueL décrivant la thématique ciblée et un seuil de pertinence fixé par avance. Il est également prévu de minimiser le parcours inutile de documents non pertinents (dont le score est inférieur au seuil de pertinence). Pour atteindre ces deux objectifs, notre approche repose sur une fonction d'estimation du score d'évaluation des candidats de l'exploration, que nous appelons *WEst : WeQueL Estimator*.

Notre méthode d'estimation du score d'évaluation d'un candidat donné d sur une requête WeQueL $\mathcal{Q}$ consiste à attribuer une valeur de score à d sous forme d'un intervalle de scores ($]s_i, s_{i+1}]$) et cela sans que le robot ne dispose encore du contenu de d . Cet intervalle d'approximation est choisi parmi un ensemble d'intervalles $\{[0, s_1], [s_1, s_2], \dots, [s_n, 1]\}$ fixé au début de l'exploration et couvrant tous les scores possibles sur la requête $\mathcal{Q}$ (discrétisation des scores possibles sur $\mathcal{Q}$). L'estimateur *WEst* considère que l'intervalle $]s_i, s_{i+1}]$ d'approximation du score de d est le plus vraisemblable compte tenu des caractéristiques observées dans le voisinage du candidat considéré. Bien que la méthode d'estimation du score autorise l'utilisation de différentes caractéristiques, nous avons fait le choix de restreindre *WEst* à l'observation de deux caractéristiques dans le voisinage d'un candidat :

1. le score de chaque document rapatrié du voisinage de ce candidat ;
2. la nature des liens reliant ce candidat à ses voisins rapatriés (lien de type père-fils ou de type frère-frère) ;

L'estimateur *WEst* calcule pour chaque candidat la probabilité que son score appartienne à chaque intervalle de discrétisation sachant les scores des documents déjà rapatriés dans le voisinage de ce candidat. L'estimation attribuée à ce candidat est égale au centre de l'intervalle de discrétisation pour lequel la probabilité précédente est la plus forte. Les estimations établies par *WEst* pour chaque candidat évoluent au fur et à mesure de l'exploration. Chaque nouveau rapatriement permet le calcul du score exact d'un document et entraîne le réajustement des estimations des anciens candidats.

Pour résumer, notre approche nécessite à son lancement 4 paramètres en entrée : les urls de départ, une requête thématique $\mathcal{Q}$ écrite dans le langage WeQueL, un seuil de pertinence au delà duquel les documents sont considérés comme pertinents, une discrétisation des scores possibles sur $\mathcal{Q}$ sous forme d'intervalles. Elle met en place un ensemble de mesures statistiques exploitées dans l'estimation du score de chaque candidat de l'exploration. Ces estimations sont « dynamiques » et sont réajustées à chaque nouveau rapatriement.

Dans la suite, nous détaillons l'estimateur *WEst* et le modèle de probabilité à partir duquel

les estimations des scores des candidats de l'exploration sont calculées. Dans la section 2.4, nous décrivons la stratégie d'exploration suivie par notre approche. Nous expliquons par la suite le procédé de réajustement des estimations des candidats à partir des mises à jour des mesures statistiques.

2 L'estimateur *WEst*

Les prédictions de scores de l'estimateur *WEst* reposent en premier lieu sur l'ensemble $\mathcal{I}_{[0,1]}$ des intervalles issus de la discrétisation des scores possibles de la requête WeQueL guidant l'exploration. Cette discrétisation permet de répartir les documents en classes suivant leurs scores d'évaluation sur la requête d'exploration. Les intervalles de discrétisation constituent donc un paramètre à part entière du processus d'apprentissage de la fonction de prédiction des scores des candidats.

Nous estimons le score d'un candidat d donné comme étant le centre de l'intervalle $I^* \in \mathcal{I}_{[0,1]}$ tel que I^* est l'intervalle de score le plus probable pour les documents déjà rapatriés ayant un voisinage similaire au voisinage du candidat d . Dans notre caractérisation du voisinage, le voisinage d'un document d est similaire au voisinage d'un autre document d' si :

- d et d' possèdent le même nombre de pères et de frères ;
- les scores des parents du document d appartiennent aux mêmes intervalles de discrétisation des parents du document d' ;
- les scores des frères du document d appartiennent aux mêmes intervalles de discrétisation des frères du document d' .

Ainsi, l'approximation du score d'évaluation $score(d, \mathcal{Q})$ d'un candidat d donné de l'exploration sur la requête $\mathcal{Q}$ consiste à déterminer l'intervalle I^* de la partition $\mathcal{I}_{[0,1]}$ pour lequel l'événement suivant a le plus de chances de se réaliser : « avoir un score appartenant à $I \in \mathcal{I}_{[0,1]}$ sachant qu'on a un voisinage similaire au voisinage du document d ». Pour cela, l'estimateur *WEst* applique le modèle de probabilités conditionnelles décrit dans ce qui suit.

2.1 Modèle de probabilité

Soient :

- d un candidat quelconque à un moment donné de l'exploration ;
- $\mathcal{I}_{[0,1]} = \{I_0 : [0, s_1], I_1 : [s_1, s_2], \dots, I_{k-1} : [s_{k-1}, 1]\}$ une partition de k intervalles sur les scores de la requête d'exploration ;
- $A(d) = \{d_1^A, \dots, d_n^A\}$ l'ensemble des parents rapatriés du candidat d ;
- $S(d) = \{d_1^S, \dots, d_m^S\}$ l'ensemble des frères rapatriés du candidat d ;
- $\{I_1^A, \dots, I_n^A\}$ les intervalles de la partition $\mathcal{I}_{[0,1]}$ contenant les scores calculés des parents du document d , i.e., $\forall i = 1..n, score(d_i^A, \mathcal{Q}) \in I_i^A$;
- $\{I_1^S, \dots, I_m^S\}$ les intervalles de la partition $\mathcal{I}_{[0,1]}$ contenant les scores calculés des frères du document d , i.e., $\forall i = 1..m, score(d_i^S, \mathcal{Q}) \in I_i^S$;
- $E[I]$ l'événement : « le score d'un document rapatrié appartient à l'intervalle I » ;
- C l'événement : « les caractéristiques du voisinage du document rapatrié sont identiques à celles du candidat d » $\equiv \left[\left(\bigwedge_{i=1}^n C_i^A \right) \wedge \left(\bigwedge_{j=1}^m C_j^S \right) \right]$ tel que :
 - $\forall i = 1..n, C_i^A \equiv$ « avoir un père ayant un score appartenant à l'intervalle I_i^A » ;
 - $\forall j = 1..m, C_j^S \equiv$ « avoir un frère ayant un score appartenant à l'intervalle I_j^S ».

La probabilité de la réalisation de l'événement $E[I]$ sachant la réalisation de l'événement C s'écrit :

$$\begin{aligned}
\forall I \in \mathcal{I}_{[0,1]}, \quad P(E[I] / C) &= \frac{P(E[I] \cap C)}{P(C)} \\
&= \frac{P(E[I] \cap [(\bigcap_{i=1}^n C_i^A) \cap (\bigcap_{j=1}^m C_j^S)])}{P([\bigcap_{i=1}^n C_i^A) \cap (\bigcap_{j=1}^m C_j^S)])}
\end{aligned} \tag{VI.1}$$

Hypothèse d'indépendance : Nous supposons que les occurrences des événements C_i^A et C_j^S sont indépendantes les unes des autres (deux à deux).

La formule précédente s'écrit :

$$\forall I \in \mathcal{I}_{[0,1]}, \quad P(E[I] / C) = \prod_{i=1}^n \frac{P(E[I] \cap C_i^A)}{P(C_i^A)} \times \prod_{j=1}^m \frac{P(E[I] \cap C_j^S)}{P(C_j^S)} \tag{VI.2}$$

L'intervalle I^* d'approximation du score du candidat d correspond alors à :

$$I^* = \arg \max_{I \in \mathcal{I}_{[0,1]}} (P(E[I] / C)) \tag{VI.3}$$

2.2 Les matrices statistiques : support d'implantation de l'estimateur *WEst*

Le calcul de la valeur de la probabilité conditionnelle $P(E[I] / C)$ de l'équation VI.2 nécessite la connaissance des valeurs des probabilités : $P(E[I] \cap C_i^A)$, $P(C_i^A)$, $P(E[I] \cap C_j^S)$ et $P(C_j^S)$. À partir du graphe exploré $\mathcal{G}_{ex}(\mathcal{D}_{ex}, \mathcal{L}_{ex})$ par le robot, *WEst* estime les probabilités précédentes de la manière suivante :

$$P(E[I] \cap C_i^S) = \frac{\# \text{ de documents } \in \mathcal{G}_{ex} \text{ ayant un score calculé } \in I \text{ et ayant un frère dont le score est } \in I_i^S}{\# \text{ de documents } \in \mathcal{G}_{ex} \text{ ayant un frère}} \tag{VI.4}$$

$$P(E[I] \cap C_i^A) = \frac{\# \text{ de documents } \in \mathcal{G}_{ex} \text{ ayant un score calculé } \in I \text{ et ayant un père dont le score est } \in I_i^A}{\# \text{ de documents } \in \mathcal{G}_{ex} \text{ ayant un père}} \tag{VI.5}$$

$$P(C_i^S) = \frac{\# \text{ de documents } \in \mathcal{G}_{ex} \text{ ayant un frère dont le score est } \in I_i^S}{\# \text{ de documents } \in \mathcal{G}_{ex} \text{ ayant un frère}} \tag{VI.6}$$

$$P(C_i^A) = \frac{\# \text{ de documents } \in \mathcal{G}_{ex} \text{ ayant un père dont le score est } \in I_i^A}{\# \text{ de documents } \in \mathcal{G}_{ex} \text{ ayant un père}} \tag{VI.7}$$

Remarque 2.1 *Nous recommandons l'utilisation d'un nombre important d'intervalles d'approximation (minimum 10) et cela dans le but d'offrir à l'estimateur *WEst* un choix plus fin pour circonscrire avec la plus grande précision possible l'approximation du score d'un candidat.*

Le calcul des différents nombres intervenant dans les 4 dernières équations à partir du graphe exploré s'effectue de manière incrémentale grâce à des variables de comptage. Ces variables se présentent sous forme de deux matrices de dimensions $k \times k$: la matrice statistique de parenté ($\mathcal{M}_{\mathcal{I}_{[0,1]}}^A$) et la matrice statistique de fratrie ($\mathcal{M}_{\mathcal{I}_{[0,1]}}^S$).

2.2.1 La matrice statistique de parenté

La matrice $\mathcal{M}_{\mathcal{I}_{[0,1]}}^A$ permet au robot de représenter, à tout instant de son exploration, la répartition statistique des intervalles de scores entre documents pères et documents fils dans le graphe exploré. Chaque élément $m^A(i, j)$ de cette matrice est chargé de comptabiliser le nombre de documents ayant un score appartenant à l'intervalle $I_i \in \mathcal{I}_{[0,1]}$ et pointant vers un fils ayant un score appartenant à l'intervalle I_j . De manière formelle, les éléments de la matrice $\mathcal{M}_{\mathcal{I}_{[0,1]}}^A$ sont définis comme suit :

$$\forall i, j = 1..k, \quad m_{ij}^A = \left| \{d' \in \mathcal{D}_{ex} / (\exists (d, l) \in \mathcal{D}_{ex} \times \mathcal{L}_{ex} : d \xrightarrow{l} d') \wedge (score(d, \mathcal{Q}) \in I_i) \wedge (score(d', \mathcal{Q}) \in I_j)\} \right| \quad (\text{VI.8})$$

2.2.2 La matrice statistique de fratrie

La matrice $\mathcal{M}_{\mathcal{I}_{[0,1]}}^S$ permet au robot de modéliser la répartition statistique des intervalles de scores entre documents frères dans le graphe exploré. Chaque élément $m^S(i, j)$ de cette matrice est chargé de comptabiliser le nombre de documents ayant un score appartenant à l'intervalle $I_i \in \mathcal{I}_{[0,1]}$ et ayant un frère dont le score appartient à l'intervalle I_j . De manière formelle, les éléments de la matrice $\mathcal{M}_{\mathcal{I}_{[0,1]}}^S$ sont définis comme suit :

$$\forall i, j = 1..k, \quad m_{ij}^S = \left| \{d \in \mathcal{D}_{ex} / \left(\exists (d', d'') \in \mathcal{D}_{ex} \times \mathcal{D}_{ex} \exists l, l' \in \mathcal{L}_{ex} : (d' \xrightarrow{l} d) \wedge (d'' \xrightarrow{l'} d) \wedge (score(d, \mathcal{Q}) \in I_i) \wedge (score(d', \mathcal{Q}) \in I_j) \right) \} \right| \quad (\text{VI.9})$$

Remarque 2.2 *La relation de fratrie entre documents est réflexive. En effet, si un document d est frère d'un autre document d' alors d' est également frère de d . Par conséquent, la matrice $\mathcal{M}_{\mathcal{I}_{[0,1]}}^S$ est symétrique, i.e., $m^S(i, j) = m^S(j, i)$*

2.2.3 Implantation du modèle de probabilité de l'estimateur *WEst*

En utilisant les définitions VI.8 et VI.9 des éléments des matrices $\mathcal{M}_{\mathcal{I}_{[0,1]}}^A$ et $\mathcal{M}_{\mathcal{I}_{[0,1]}}^S$, les équations VI.4, VI.5, VI.6 et VI.7 se réécrivent comme suit (nous considérons que l'intervalle I possède le rang j dans la partition $\mathcal{I}_{[0,1]}$) :

$$\begin{aligned} P(E[I] \cap C_i^S) &= \frac{m_{ji}^S}{\sum_{j=1}^k \sum_{i=1}^k m_{ji}^S} \\ P(E[I] \cap C_i^A) &= \frac{m_{ij}^A}{\sum_{j=1}^k \sum_{i=1}^k m_{ij}^A} \\ P(C_i^S) &= \frac{\sum_{j=1}^k m_{ji}^S}{\sum_{i=1}^k \sum_{j=1}^k m_{ji}^S} \\ P(C_i^A) &= \frac{\sum_{j=1}^k m_{ij}^A}{\sum_{i=1}^k \sum_{j=1}^k m_{ij}^A} \end{aligned} \quad (\text{VI.10})$$

2.3 Estimation du score d'un candidat

À la suite de la détermination de l'intervalle $(I^* =]s_i, s_{i+1}] \in \mathcal{I}_{[0,1]})$ d'approximation du score d'un candidat donné d' , l'estimateur *WEst* calcule le score estimé de d' comme étant le centre de

l’intervalle I^* , *i.e.* :

$$\text{score_estimé}(d, \mathcal{Q}) = s_i + \frac{s_{i+1} - s_i}{2} \quad (\text{VI.11})$$

Le score calculé par cette formule est interprété par le robot comme étant l’intérêt à accorder au rapatriement du candidat considéré. Plus ce score est grand, plus l’intérêt de rapatrier ce candidat est important.

Remarque 2.3 *Dans nos expérimentations, nous avons utilisé la formule précédente pour attribuer aux candidats une approximation numérique de leur score plus fine qu’une approximation par un intervalle de score. Ceci nous a permis d’apprécier l’erreur moyenne commise par l’estimateur *WEst* dans ses approximations de scores. Pour un document rapatrié donné, l’erreur d’approximation correspond à la différence entre le score effectif de ce document après évaluation de son contenu sur la requête d’exploration et l’approximation de ce score faite par l’estimateur *WEst* lorsque ce document était candidat.*

2.4 Stratégie d’exploration

Au lancement de l’exploration, le robot ne dispose d’aucun élément pour distinguer les candidats « prometteurs » parmi les candidats définis par les points (urls) de départ. En effet, l’amorçage des prédictions de l’estimateur *WEst* nécessite la connaissance du voisinage entourant chaque point de départ ainsi que des valeurs non nulles dans les éléments des matrices statistiques : matrice de parenté et matrice de fratrie. Pour ces raisons, nous avons choisi de rapatrier, sans distinction, tous les documents désignés par les urls de départ au commencement d’une exploration utilisant notre approche. Ce choix peut être interprété comme une application restreinte de la stratégie « largeur d’abord » à la profondeur 0 (profondeur des points de départ dans le graphe exploré).

La stratégie de type « largeur d’abord », adoptée pour débiter une exploration utilisant notre approche, permet au robot de se constituer un graphe initial comprenant des documents rapatriés et évalués et éventuellement des liens entre ces documents initiaux. Ce graphe du début offre au robot une répartition initiale favorisant l’amorçage des calculs du système *WEst*. En effet, les éléments des matrices $\mathcal{M}_{\mathcal{I}_{[0,1]}}^A$ et $\mathcal{M}_{\mathcal{I}_{[0,1]}}^S$ sont initialisés à partir des valeurs constatées sur ce graphe de départ. L’estimateur *WEst* est donc rapidement opérationnel pour suggérer les « meilleurs » candidats à suivre. Dans les mêmes conditions, une approche « intelligente » [Aggarwal et al. 2001] ne peut amorcer ses calculs de ratios d’intérêts pour ses candidats que si le robot a rapatrié dans ce graphe des documents pertinents.

Après cette phase d’initialisation, le robot adopte une stratégie sélective qui consiste à privilégier le rapatriement des candidats estimés avec les scores les plus élevés.

2.5 Ajustement des estimations de *WEst*

À la suite de chaque itération du robot, l’estimateur *WEst* procède à l’appréciation de ses propres prédictions en comparant le score estimé de chaque candidat rapatrié au score réel obtenu par l’évaluation de son contenu. Cette appréciation se traduit par une confirmation ou une infirmation de la prédiction du dernier document rapatrié.

2.5.1 Confirmation d'une prédiction de l'estimateur *WEst*

La confirmation de la prédiction « $score(d, \mathcal{Q}) \in I_i$ » se produit lorsque le score calculé du contenu rapatrié du candidat d appartient effectivement à l'intervalle $I_i \in \mathcal{I}_{[0,1]}$. Dans ce cas, l'estimateur *WEst* incrémente de 1 les éléments de la colonne i de la matrice d'ancêtres correspondant aux indices des intervalles contenant le score des parents déjà rapatriés, *i.e.* :

$$\forall j \in \{k / \exists I_k \in \mathcal{I}_{[0,1]}, \exists d' \in A(d), score(d', \mathcal{Q}) \in I_k\}, m_{ji}^A \leftarrow m_{ji}^A + 1$$

Dans sa confirmation de la prédiction « $score(d, \mathcal{Q}) \in I_i$ », l'estimateur *WEst* incrémente également de 1 les éléments de la ligne i et de la colonne i (par symétrie) de la matrice de fratrie correspondant aux indices des intervalles contenant les scores réels des frères déjà rapatriés du document d , *i.e.* :

$$\forall j \in \{k / \exists I_k \in \mathcal{I}_{[0,1]} \exists d' \in S(d), score(d', \mathcal{Q}) \in I_k\}, (m_{ji}^S \leftarrow m_{ji}^S + 1) \wedge (m_{ij}^S \leftarrow m_{ij}^S + 1)$$

Les différentes incrémentations, opérées lors de la confirmation de la prédiction « $score(d, \mathcal{Q}) \in I_i$ », permettent à l'estimateur *WEst* de renforcer les valeurs des probabilités vues dans l'équation VI.10. Autrement dit, pour un candidat futur d' ayant un voisinage semblable à celui du document d , l'estimateur *WEst* suggérera la prédiction « $score(d', \mathcal{Q}) \in I_i$ » avec plus de chances de réalisation.

2.5.2 Infirmer d'une prédiction de l'estimateur *WEst*

L'infirmer de la prédiction « $score(d, \mathcal{Q}) \in I_i$ » se produit lorsque le score calculé du contenu rapatrié du candidat d n'appartient pas à l'intervalle $I_i \in \mathcal{I}_{[0,1]}$. En supposons que le score réel calculé après rapatriement du candidat d soit compris dans l'intervalle $I_{i'}$, l'estimateur infirme sa prédiction « $score(d, \mathcal{Q}) \in I_i$ » en incrémentant les éléments qu'il aurait fallu incrémenter si la prédiction était « $score(d, \mathcal{Q}) \in I_{i'}$ ». Ainsi, l'approximation par l'intervalle $I_{i'}$ du score d'un futur candidat présentant un voisinage similaire à celui du document d sera progressivement privilégiée à l'approximation par l'intervalle I_i .

Remarque 2.4 *WEst* procède à l'auto-critique de ses prédictions en confirmant ou en infirmant toute estimation ayant conduit à un rapatriement de document. Cette auto-critique « en-ligne » permet à l'estimateur *WEst* de réajuster progressivement l'intérêt des candidats restants en fonction de l'évolution de la répartition statistique entre les intervalles de scores dans le graphe exploré. En effet, les éléments des matrices statistiques de parenté et de fratrie sont continuellement modifiés après chaque rapatriement rendant les anciennes approximations des scores des candidats non encore rapatriés caduques. Ainsi, l'estimateur *WEst* met à jour ses approximations sur tous les candidats restants dans la pile de candidats du robot (conformément aux calculs de probabilités montrés dans l'équation VI.10 page 140). Dans nos expérimentations et dès lors que le nombre de candidats dans la pile devenait important, l'opération de rafraîchissement des estimations sur tous les candidats de la pile a généré un accroissement pénalisant du temps nécessaire au robot pour choisir son prochain candidat à rapatrier. Sachant qu'une incrémentation de 1 sur les éléments des matrices statistiques n'influe que très peu sur les calculs de probabilités de l'équation VI.10 page 140, nous avons fixé un taux de rafraîchissement ($\approx$ tous les 50 rapatriements) des estimations qui permet d'atténuer les effets du réajustement des estimations des candidats restants.

2.6 Exemple illustratif de l'approche

Soit $\mathcal{G}_{ex}(\mathcal{D}_{ex}, \mathcal{L}_{ex})$ le graphe exploré par un robot à un instant donné (voir figure VI.1 page 143). Soit également $\mathcal{I}_{[0,1]} = \{I_1 : [0, \frac{1}{3}], I_2 :]\frac{1}{3}, \frac{2}{3}], I_3 :]\frac{2}{3}, 1]\}$ une partition équilibrée sur l'intervalle $[0, 1]$ où les intervalles I_1 , I_2 et I_3 sont respectivement représentés par les couleurs **bleu**, **rouge** et **vert** dans la figure VI.1 page 143.

Les documents explorés sont numérotés de 1 à 9 et représentés dans la figure VI.1 par des cercles en trait continu. Avant leur exploration, les scores de ces 9 documents ont été approximés par des intervalles de scores. Une fois rapatriés, le robot procède à l'évaluation de leur contenu pour déterminer leur score exact. Dans la figure VI.1, le score de chaque document rapatrié est représenté par la couleur de l'intervalle qui contient ce score. Par exemple, le document 1 a été évalué avec un score=0.8. Il est donc représenté par la couleur verte symbolisant l'appartenance de ce score calculé à l'intervalle I_3 .

Le document 10, symbolisé par un cercle en trait discontinu, représente un candidat accessible depuis le document 1. Dans cet exemple, nous montrons comment l'estimateur *WEst* approxime le score d'évaluation du candidat 10 en connaissant son voisinage et en ayant la répartition statistique entre les trois intervalles de la partition $\mathcal{I}_{[0,1]}$ dans le graphe $\mathcal{G}_{ex}(\mathcal{D}_{ex}, \mathcal{L}_{ex})$.

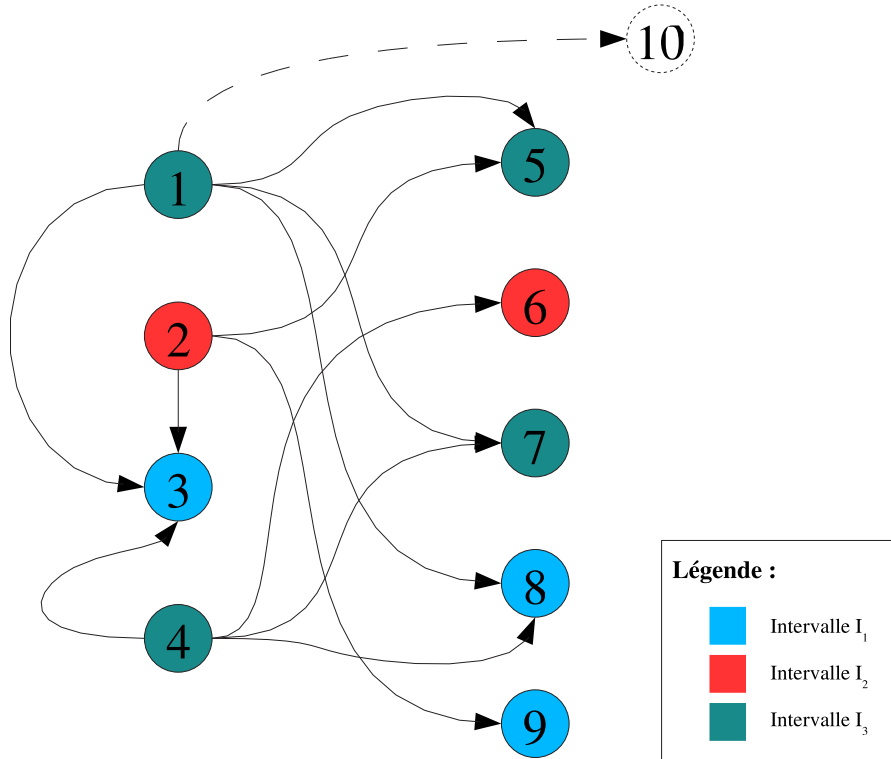


FIG. VI.1 – Exemple d'estimation du score d'un candidat

L'ensemble des parents et des frères rapatriés du document 10 sont :

$$\begin{aligned} A(10) &= \{1\} \\ S(10) &= \{5, 7, 8, 3\} \end{aligned}$$

Le voisinage du document 10 peut être défini par l'événement suivant : $C \equiv (score(1) \in I_3) \wedge (score(5) \in I_3) \wedge (score(7) \in I_3) \wedge (score(8) \in I_1) \wedge (score(3) \in I_1)$. Pour un document quelconque présentant un tel voisinage, la probabilité pour que le robot rapatrie un document ayant un score appartenant à l'intervalle I_1 est :

$$\begin{aligned} P(E[I_1]/C) &= \left(\frac{P(E[I_1] \cap C_3^A)}{P(C_3^A)} \right) \times \left(\frac{P(E[I_1] \cap C_3^S)}{P(C_3^S)} \right) \times \frac{P(E[I_1] \cap C_3^S)}{P(C_3^S)} \times \frac{P(E[I_1] \cap C_1^S)}{P(C_1^S)} \times \frac{P(E[I_1] \cap C_1^S)}{P(C_1^S)} \\ &= \frac{P(E[I_1] \cap C_3^A)}{P(C_3^A)} \times \left(\frac{P(E[I_1] \cap C_3^S)}{P(C_3^S)} \right)^2 \times \left(\frac{P(E[I_1] \cap C_1^S)}{P(C_1^S)} \right)^2 \end{aligned}$$

En s'appuyant sur les formules de l'équation VI.10 page 140, la probabilité précédente s'exprime comme suit :

$$P(E[I_1]/C) = \frac{m_{31}^A}{\sum_{j=1}^3 m_{3j}^A} \times \left(\frac{m_{31}^S}{\sum_{j=1}^3 m_{3j}^S} \right)^2 \times \left(\frac{m_{11}^S}{\sum_{j=1}^3 m_{1j}^S} \right)^2$$

De même, la probabilité pour que le robot rapatrie un document ayant un score appartenant à l'intervalle I_2 est :

$$P(E[I_2]/C) = \frac{m_{32}^A}{\sum_{j=1}^3 m_{3j}^A} \times \left(\frac{m_{32}^S}{\sum_{j=1}^3 m_{3j}^S} \right)^2 \times \left(\frac{m_{21}^S}{\sum_{j=1}^3 m_{2j}^S} \right)^2$$

Et pour l'intervalle I_3 :

$$P(E[I_3]/C) = \frac{m_{33}^A}{\sum_{j=1}^3 m_{3j}^A} \times \left(\frac{m_{33}^S}{\sum_{j=1}^3 m_{3j}^S} \right)^2 \times \left(\frac{m_{31}^S}{\sum_{j=1}^3 m_{3j}^S} \right)^2$$

L'état des matrices $\mathcal{M}_{\mathcal{I}_{[0,1]}}^A$ et $\mathcal{M}_{\mathcal{I}_{[0,1]}}^S$ après le rapatriement des 9 documents du graphe est :

$$\mathcal{M}_{\mathcal{I}_{[0,1]}}^A = \begin{bmatrix} 0 & 0 & 0 \\ 2 & 0 & 1 \\ 2 & 1 & 2 \end{bmatrix} \quad \mathcal{M}_{\mathcal{I}_{[0,1]}}^S = \begin{bmatrix} 3 & 1 & 3 \\ 1 & 0 & 1 \\ 3 & 1 & 2 \end{bmatrix}$$

Calcul numérique :

$$\begin{aligned} P(E[I_1]/C) &= \frac{2}{5} \times \left(\frac{3}{6}\right)^2 \times \left(\frac{3}{7}\right)^2 \\ &= 0,018 \end{aligned}$$

$$\begin{aligned} P(E[I_2]/C) &= \frac{1}{5} \times \left(\frac{1}{6}\right)^2 \times \left(\frac{1}{2}\right)^2 \\ &= 0,001 \end{aligned} \tag{VI.12}$$

$$\begin{aligned} P(E[I_3]/C) &= \frac{2}{5} \times \left(\frac{2}{6}\right)^2 \times \left(\frac{3}{6}\right)^2 \\ &= 0,011 \end{aligned}$$

Sachant le voisinage du document 10, le robot a plus de chances de rapatrier un document avec un score d'évaluation appartenant à l'intervalle I_1 . Par conséquent, le candidat 10 est approximé avec un score compris dans l'intervalle $[0, \frac{1}{3}]$ et l'intérêt de son rapatriement est égal au centre de cet intervalle, *i.e.*, $\frac{1}{6}$.

Étant le seul candidat, le document 10 est rapatrié puis évalué pour déterminer son score réel. Deux cas possibles sont envisageables :

Cas 1 : le score du contenu rapatrié du document 10 appartient à l'intervalle $[0, \frac{1}{3}]$. L'estimateur *WEst* confirme sa prédiction en mettant à jour ses matrices statistiques de la manière suivante :

$$\mathcal{M}_{\mathcal{I}_{[0,1]}}^A = \begin{bmatrix} 0 & 0 & 0 \\ 2 & 0 & 1 \\ \boxed{3} & 1 & 2 \end{bmatrix} \quad \mathcal{M}_{\mathcal{I}_{[0,1]}}^S = \begin{bmatrix} \boxed{4} & 1 & \boxed{4} \\ 1 & 0 & 1 \\ \boxed{4} & 1 & 2 \end{bmatrix}$$

Cas 2 : le score du contenu rapatrié du document 10 n'appartient pas à l'intervalle $[0, \frac{1}{3}]$. Dans ce cas, deux possibilités d'infirmer la prédiction sur l'intervalle I_1 :

Cas 2.1 : le score du contenu rapatrié du document 10 appartient à l'intervalle $]\frac{1}{3}, \frac{2}{3}]$. La mise à jour des matrices statistiques donne :

$$\mathcal{M}_{\mathcal{I}_{[0,1]}}^A = \begin{bmatrix} 0 & 0 & 0 \\ 2 & 0 & 1 \\ 2 & \boxed{2} & 2 \end{bmatrix} \quad \mathcal{M}_{\mathcal{I}_{[0,1]}}^S = \begin{bmatrix} 3 & 1 & 3 \\ 1 & \boxed{1} & \boxed{2} \\ 3 & \boxed{2} & 2 \end{bmatrix}$$

Cas 2.2 : le score du contenu rapatrié du document 10 appartient à l'intervalle $]\frac{2}{3}, 1]$. La mise à jour des matrices statistiques donne :

$$\mathcal{M}_{\mathcal{I}_{[0,1]}}^A = \begin{bmatrix} 0 & 0 & 0 \\ 2 & 0 & 1 \\ 2 & 1 & \boxed{3} \end{bmatrix} \quad \mathcal{M}_{\mathcal{I}_{[0,1]}}^S = \begin{bmatrix} 3 & 1 & \boxed{4} \\ 1 & 0 & 1 \\ \boxed{4} & 1 & \boxed{3} \end{bmatrix}$$

3 Évaluation expérimentale de l'approche

Pour les besoins de nos expérimentations, nous avons conçu « **webcrawler** », un robot explorateur sous forme d'un paquetage « **.jar** » entièrement écrit en JAVA. La mise au point de ce robot représente plus de 10.000 lignes de code réparties en une trentaine de fichiers sources.

Le robot **webcrawler** se compose de différents modules génériques conformément à l'architecture présentée dans le chapitre IV (c.f. section 2.2 page 94 et figure IV.3 page 95). La particularité de **webcrawler** réside dans la capacité de son module d'application de la stratégie d'exploration à intégrer différentes stratégies. En effet, il est possible d'utiliser **webcrawler** pour effectuer plusieurs explorations utilisant chacune une stratégie différente sans devoir recompiler les sources. Pour cela, l'utilisateur dispose de deux moyens déclaratifs de spécification de la stratégie à utiliser : par argument de la ligne de commande de lancement de **webcrawler** ou par renseignement du champ **StrategyName** dans un fichier de paramétrage du fonctionnement global de **webcrawler**. Ce fichier permet également de spécifier le nombre d'agents à dédier au rapatriement de documents ainsi que d'autres options relatives au traitement des documents rapatriés.

En utilisant **webcrawler**, nous avons mené une série d'explorations expérimentales pour évaluer l'apport de notre approche dans une exploration web sélective à la recherche de documents thématiques. Nous nous sommes conformés pour cela au protocole expérimental décrit ci-dessous :

3.1 Protocole expérimental

Nous avons défini un protocole expérimental spécifiant 4 aspects de notre expérimentation : le type de documents ciblés par les explorations lancées lors de nos tests expérimentaux, les stratégies d'exploration que nous avons confrontées, les mesures de qualité de chaque exploration et le test permettant d'apprécier la robustesse de l'approche.

3.1.1 Documents visés

Ayant défini au préalable un ensemble de requêtes pour 4 thématiques différentes lors de la validation expérimentale du langage WeQueL dans le chapitre III (cf. section 5 page 68), nous avons naturellement réutilisé ces requêtes pour rechercher des documents thématiques sur le web. Nous nous sommes cependant restreints à seulement deux requêtes WeQueL pour cibler la recherche sur deux catégories de documents thématiques : les documents traitant du risque alimentaire et les documents de cours informatiques en-ligne et en français.

Au vu des bons résultats de précision de la requête " Q_{Num} " (voir figure III.7 page 78) employée dans la partie filtrage du projet « eDot » (voir figure III.11 et III.12 page 85 et 86), nous avons réutilisé cette requête pour calculer automatiquement la pertinence des documents rapatriés par différentes stratégies d'exploration. Un document rapatrié par **webcrawler** (selon n'importe quelle stratégie d'exploration) est considéré comme pertinent pour la thématique du risque alimentaire si le score obtenu par l'évaluation de son contenu sur la requête WeQueL " Q_{Num} " dépasse le seuil de pertinence de 0,14 (seuil résultant de l'expérimentation de filtrage reportée au chapitre III).

Pour les documents de cours en informatique, nous avons élaboré une nouvelle requête WeQueL à interprétation numérique en nous inspirant de la requête " Q_{info} " (voir figure III.4 page 69). La réécriture de la requête " Q_{info} " en une requête à interprétation numérique, noté " Q_{info}^{num} ", est montrée dans la figure VI.2 à la page 147.

De la même manière que pour la thématique du risque alimentaire, un document rapatrié sera considéré comme pertinent pour la thématique des cours informatiques en-ligne si le score de son contenu sur la requête " Q_{info}^{num} " est supérieur au seuil de 0,1. Ce seuil a été fixé de façon empirique.

3.1.2 Stratégies d'exploration concurrentes

Afin d'apprécier l'efficacité de notre approche, nous l'avons confrontée à d'autres techniques d'exploration du web. Par cette confrontation, nous ciblons l'étude de deux aspects :

1. l'utilité des approches sélectives face aux approches systématiques dans une exploration à la recherche de documents thématiques ;
2. L'apport de notre approche par rapport à d'autres approches sélectives.

Pour l'étude du premier aspect, nous avons comparé les résultats de 4 explorations différentes pour chaque thématique : une exploration utilisant une stratégie en largeur d'abord, une exploration appliquant une stratégie en profondeur d'abord, une exploration utilisant une stratégie « intelligente » (approche de *Charu C. Aggarwal et al.* [Aggarwal et al. 2001]) et une exploration utilisant notre approche. L'objectif est de mettre en évidence le gain en précision d'une recherche de documents thématiques lors de l'emploi d'une approche sélective à la place d'une approche systématique.

```

<?xml version="1.0" encoding="ISO-8859-1"?>
<WeQueLComplexQuery xmlns="http://www.lri.fr/~mezaour/WebServices/xml-schemas"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.lri.fr/~mezaour/WebServices/xml-schemas http://www.lri.fr/~mezaour/WeQueL/WeQueLschema.xsd"
  evaluation_function="max">

  <combination>
    <atomic_query name="mime" weight="1" semantic_builder="exactly" semantic_builder_parameters="1">
      text/html
    </atomic_query>
    <atomic_query name="title" weight="0.3" semantic_builder="Fscore">
      cours,informatique,java,algorithmique,intelligence artificielle,'ia',html,programmation,support,documentation,cours 'c','c++',cpp,c plus plus,bases donnees,'bd',tutoriel,sgbd,logique,compilation,langage programmation,calculabilite,cours info,architecture,reseaux,bureautique,internet,
      protocole,systeme exploitation,pao,cao,graphisme,sql,prolog
    </atomic_query>
    <atomic_query name="url" weight="0.2" semantic_builder="Fscore">
      cours,informatique,java,algorithmique,intelligence artificielle,'ia',html,programmation,support,documentation,'c','c++',cpp,c plus plus,bases donnees,'bd',tutoriel,sgbd,logique,compilation,langage programmation,calculabilite,cours info,architecture,reseaux,bureautique,internet,protocole,
      systeme exploitation,pao,cao,graphisme,calcul,node,~,slide,transparent,~ sld,cours sld,univ,universite,sql,poly,prolog,enseignement
    </atomic_query>
    <atomic_query name="incoming_links" weight="0.1" semantic_builder="Fscore">
      java,algorithmique,intelligence artificielle,html,programmation,informatique,support,cours,documentation,'c',cpp,c plus plus,'c++',bases donnees,sgbd,'bd',ligne,algo,calculabilite,logique,compilation,langage programmation,calculabilite,cours info,architecture,compilation,reseaux,note,'ia',sql,poly,transparent,slide,introduction,tutoriel,presentation,notion,
      maitrise,licence,iup,master,prolog
    </atomic_query>
    <atomic_query name="outgoing_links" weight="0.2" semantic_builder="Fscore">
      introduction,table matieres,sommaire,chapitre,preambule,precedent,suivant,next,previous,prec,suiv,index,biblio,plan cours,glossaire,notion
    </atomic_query>
    <atomic_query name="body" weight="0.2" semantic_builder="Fscore">
      introduction,table matieres,sommaire,chapitre,preambule,precedent,suivant,next,previous,prec,suiv,index,definition,section,partie,exemple,cours,
      informatique,java,algorithmique,intelligence artificielle,'ia',html,programmation,support,documentation,'c','c++',cpp,c plus plus,bases donnees,'bd',tutoriel,sgbd,logique,compilation,langage programmation,calculabilite,cours info,architecture,reseaux,bureautique,internet,protocole,
      systeme exploitation,pao,cao,graphisme,sql,biblio,plan cours,1,2,3,glossaire,notion,resume,conclusion,exercice,generalite
    </atomic_query>
  </combination>

  <combination>
    <atomic_query name="mime" weight="1" semantic_builder="exactly" semantic_builder_parameters="1">
      application/postscript,application/pdf,application/. *powerpoint,application/ppt,application/. *tar,application/. *zip,application/msword
    </atomic_query>
    <atomic_query name="url" weight="0.7" semantic_builder="Fscore">
      cours,informatique,java,algorithmique,intelligence artificielle,'ia',html,programmation,support,documentation,'c','c++',cpp,c plus plus,bases donnees,'bd',tutoriel,sgbd,logique,compilation,langage programmation,calculabilite,cours info,architecture,reseaux,bureautique,internet,protocole,
      systeme exploitation,pao,cao,graphisme,calcul,node,~,slide,transparent,~ sld,cours sld,univ,universite,sql,poly,prolog,enseignement
    </atomic_query>
    <atomic_query name="incoming_links" weight="0.3" semantic_builder="Fscore">
      java,algorithmique,intelligence artificielle,html,programmation,informatique,support,cours,documentation,'c',cpp,c plus plus,'c++',bases donnees,sgbd,'bd',ligne,algo,calculabilite,logique,architecture,compilation,reseaux,note,'ia',sql,poly,transparent,slide,introduction,tutoriel,presentation,ps,pdf,
      ms word,ppt,power point,notion,maitrise,licence,iup,master,prolog,telecharge,format
    </atomic_query>
  </combination>

  <combination>
    <atomic_query name="mime" weight="1" semantic_builder="exactly" semantic_builder_parameters="1">
      application/postscript,application/pdf,application/. *powerpoint,application/ppt,application/. *tar,application/. *zip,application/msword
    </atomic_query>
    <atomic_query name="titles incoming pages" weight="0.3" semantic_builder="Fscore">
      cours,informatique,java,algorithmique,intelligence artificielle,'ia',html,programmation,support,documentation,'c','c++',cpp,c plus plus,bases donnees,'bd',tutoriel,sgbd,logique,compilation,langage programmation,calculabilite,cours info,architecture,reseaux,bureautique,internet,protocole,
      systeme exploitation,pao,cao,graphisme,sql,index
    </atomic_query>
    <atomic_query name="url" weight="0.7" semantic_builder="Fscore">
      cours,informatique,java,algorithmique,intelligence artificielle,'ia',html,programmation,support,documentation,'c','c++',cpp,c plus plus,bases donnees,'bd',tutoriel,sgbd,logique,compilation,langage programmation,calculabilite,cours info,architecture,reseaux,bureautique,internet,protocole,
      systeme exploitation,pao,cao,graphisme,calcul,node,~,slide,transparent,~ sld,cours sld,univ,universite,sql,poly,prolog,telecharge,format,enseignement
    </atomic_query>
  </combination>
</WeQueLComplexQuery>

```

FIG. VI.2 – Requête " Q_{info}^{num} " pour la recherche de documents de cours en informatique

Pour l'étude du second aspect, nous avons volontairement changé la fonction utilisée par l'approche « intelligente » dans l'évaluation des documents rapatriés. L'objectif de cette étude est de montrer la fiabilité et la robustesse de notre approche face à l'approche « intelligente ». Les détails de cette expérimentation sont donnés plus loin dans la section « test de robustesse » prévu par notre protocole expérimental.

Remarque 3.1 *Dans nos expérimentations, nous avons volontairement choisi de ne pas confronter notre approche aux approches sélectives nécessitant un apprentissage initial sur un graphe d'entraînement (cf. chapitre précédent, section 3.1 page 112). Ce choix de ne pas utiliser ces approches se justifie par le coût élevé de leur mise en œuvre (construction de graphe d'entraînement + étiquetage manuel + apprentissage).*

3.1.3 Paramètres des explorations expérimentales

Pour garantir l'obtention de résultats comparables, nous avons veillé à mettre en œuvre des conditions d'expérimentation identiques entre les différentes stratégies prévues par notre protocole expérimental. Ces conditions concernent le paramétrage du fonctionnement du robot **webcrawler**, le choix de ses points de départ, les fonctions d'évaluation des documents rapatriés utilisées par l'approche « intelligente » et notre approche.

Paramétrage du robot : Sur les deux thématiques d'expérimentation, les quatre stratégies d'exploration choisies utilisent un paramétrage commun et standard du robot **webcrawler**. Ce paramétrage prévoit pour chacune des quatre stratégies de l'expérimentation :

- un même nombre d'agents dédiés au rapatriement de documents (10 agents) ;
- un « *timeout* » identique pour l'abandon d'un téléchargement (pas plus de 10 secondes pour un document) ;
- l'injection de tous les liens extraits d'un document HTML dans la pile d'urls ;
- Une limite maximale de 10.000 documents rapatriés (cette limite a été fixée expérimentalement en tenant compte des restrictions matérielles de la machine exécutant **webcrawler**).

Choix des points de départ : Pour une même thématique et pour chaque stratégie employée, le robot **webcrawler** est lancé à partir des mêmes points de départ. Ceci garantit un même graphe web au départ sans nécessairement emprunter les mêmes chemins et découvrir les mêmes documents. C'est sur ce point précis que la qualité des différentes stratégies est évaluée.

Pour une thématique donnée, les points de départ sont recueillis par union des réponses fournies par Google avancé [Google_advanced site] à une série de requêtes mots clés. Ces requêtes mots clés correspondent à la décomposition des différentes valeurs contenues dans les différentes requêtes atomiques de la requête d'évaluation utilisée par le robot. Le processus de traduction d'une requête WeQueL en une requête Google avancé est décrit en détail dans la section 5.1.2 page 72.

Cette manière de recueillir les points de départ garantit à **webcrawler** un commencement d'exploration à partir de documents en rapport avec la thématique visée. De plus, en extrapolant le principe de proximité thématique dans le graphe du web, nous espérons augmenter les chances de **webcrawler** de trouver des documents pertinents à partir de points de départ déjà en rapport avec la thématique d'exploration. Ceci peut s'avérer utile lors de la validation expérimentale des résultats des explorations des différentes stratégies.

Fonction d'évaluation des documents rapatriés par l'approche « intelligente » : Nous avons montré dans le chapitre précédent (voir section 3.2.2 page 121) qu'une exploration utilisant l'approche « intelligente » repose sur une fonction booléenne d'évaluation de la pertinence des documents rapatriés. Pour l'étude des deux aspects mentionnés plus haut dans la section 3.1.2 à la page 146, nous avons expérimenté l'approche « intelligente » avec deux fonctions booléennes différentes pour chaque thématique : une première fonction évaluant les documents rapatriés conformément au critère de pertinence défini pour chaque thématique dans la section 3.1.1 page 146 et une seconde fonction erronée pour les besoins du test de robustesse (voir plus loin à la page 150). Soient :

- $\mathcal{T}$ une thématique parmi les deux thématiques ciblées par notre expérimentation : risque alimentaire ou cours informatiques en-ligne ;
- Q^T la requête WeQueL décrivant la thématique $\mathcal{T}$: $Q^T \in \{Q_{info}^{num}, Q_{Num}\}$;
- $\mathcal{D}$ l'ensemble des documents du web.

– δ^T le seuil de pertinence défini dans la section 3.1.1 pour la thématique T : $\delta^T \in \{0, 14, 0, 1\}$

Pour une thématique T donnée, les deux fonctions précédentes, notées F_1^T et F_2^T , sont définies comme suit :

$$\begin{aligned} F_1^T &: \mathcal{D} \longrightarrow \{0, 1\} \\ d &\longrightarrow F_1^T(d) = \begin{cases} 1 & \text{si } score(d, Q^T) \geq \delta^T \\ 0 & \text{sinon} \end{cases} \end{aligned} \quad (VI.13)$$

$$\begin{aligned} F_2^T &: \mathcal{D} \longrightarrow \{0, 1\} \\ d &\longrightarrow F_2^T(d) = \begin{cases} 1 & \text{si } score(d, Q^T) \geq \overline{\delta^T} \text{ où : } \overline{\delta^T} > \delta^T \\ 0 & \text{sinon} \end{cases} \end{aligned} \quad (VI.14)$$

Fonction d'évaluation des documents rapatriés par notre approche : Notre approche repose sur une fonction qui utilise la requête thématique guidant l'exploration pour évaluer les documents rapatriés. Cette fonction, notée $\mathcal{F}_*^T$ pour la thématique T , renvoie un score numérique compris entre zéro et un. Précisons que la fonction $\mathcal{F}_*^T$ est totalement indépendante du seuil de pertinence δ^T . La fonction $\mathcal{F}_*^T$ s'exprime comme suit :

$$\begin{aligned} \mathcal{F}_*^T &: \mathcal{D} \longrightarrow [0, 1] \\ d &\longrightarrow \mathcal{F}_*^T(d) = score(d, Q^T) \end{aligned} \quad (VI.15)$$

Remarque 3.2 *Par définition, les approches systématiques n'emploient aucune fonction d'évaluation des documents rapatriés. Nous avons néanmoins paramétré le robot `webcrawler` pour qu'il utilise la fonction F_1^T pour déterminer automatiquement la pertinence de chaque document rapatrié par une approche systématique. Le but de cette évaluation est d'automatiser le processus d'appréciation de la qualité des résultats fournis par nos explorations systématiques. Bien entendu, ceci n'a aucune incidence sur les stratégies de rapatriement des explorations systématiques.*

3.1.4 Estimation du score des candidats pour notre approche

Pour notre approche, nous avons utilisé une partition composée de 100 intervalles de largeurs égales. Par ce nombre, nous souhaitons que l'estimateur *WEst* approxime les candidats le plus finement possible avec une erreur ne dépassant pas 10^{-2} .

3.1.5 Mesure de la qualité de l'exploration

Nous avons retenu la précision comme mesure de la qualité de l'exploration. Contrairement aux systèmes classiques d'apprentissage qui ne s'intéressent qu'à la précision finale, notre mesure de précision est calculée de manière incrémentale à la suite de chaque rapatriement. Ceci nous permet d'apprécier l'évolution de la précision de `webcrawler` dans l'ordre des documents prioritaires qu'il rapatrie. Cette mesure est connue sous le nom de « *taux de moisson* » ou *harvest rate*.

En supposant qu'à l'itération i , le robot `webcrawler` ait rapatrié N^i documents au total dont N_C^i sont pertinents, le taux de moisson p^i de `webcrawler` est calculée par la formule suivante :

$$p^i = \frac{N_C^i}{N^i} \quad (VI.16)$$

La représentation graphique des différents taux pendant toute une exploration revient à tracer une courbe d'élévation ou une courbe *lift* (voir définition au chapitre III, page 84).

Erreur moyenne d'estimation du score : Compte tenu de la nature des estimations de *WEst*, nous avons défini une mesure spécifique à notre approche qui consiste à calculer l'erreur moyenne d'estimation des scores des candidats.

Pour une exploration sélective guidée par une requête WeQueL $\mathcal{Q}$, l'erreur d'estimation du score d'un document est définie comme étant la différence entre le score réel d'un document rapatrié et son score estimé lorsque ce document était candidat (c'est-à-dire $\text{score_estimé}(d, \mathcal{Q})$: voir formule VI.11 page 141) :

$$\text{erreur}(d) = \text{score}(d, \mathcal{Q}) - \text{score_estimé}(d, \mathcal{Q})$$

En supposant que $\mathcal{D}_{ex}$ soit l'ensemble des documents rapatriés par une exploration utilisant notre approche, l'erreur moyenne se calcule par la formule suivante :

$$\text{erreur_moyenne} = \frac{1}{|\mathcal{D}_{ex}|} \sum_{d \in \mathcal{D}_{ex}} \text{erreur}(d)$$

Par cette dernière mesure, nous souhaitons évaluer la fiabilité des estimations de scores de candidats suggérées par l'estimateur *WEst*.

3.1.6 Tests de robustesse

Dans le contexte de notre expérimentation, nous avons défini deux tests dans le but de montrer la robustesse de notre approche.

Test de stabilité : Le test de stabilité sur une thématique $\mathcal{T}$ consiste à comparer les résultats de notre approche aux résultats de l'approche « intelligente » lancée avec la fonction $F_2^{\mathcal{T}}$ (voir définition dans la formule VI.14 page 149). De par sa définition, la fonction $F_2^{\mathcal{T}}$ est une mauvaise estimation de la pertinence réelle des documents. En effet, pour les deux thématiques ciblées par notre expérimentation, cette fonction utilise un seuil de 0,2 qui est plus élevé que les deux seuils fixés expérimentalement pour ces thématiques (voir les seuils de pertinence de chaque thématique dans la section 3.1.1 page 146). Rappelons, que notre approche est totalement indépendante du seuil de pertinence.

Par ce test, nous souhaitons montrer que notre approche est plus stable que l'approche « intelligente ». En effet, la précision des résultats de cette dernière risque d'être très dépendante de la qualité de la fonction booléenne utilisée pour évaluer les documents rapatriés.

Test de rendement : Le test de rendement consiste à borner l'exploration du robot par une limite de temps. Pour ce test, nous avons choisi la limite d'une heure pour chaque stratégie. L'objectif visé consiste à étudier l'aptitude d'une stratégie donnée à orienter rapidement le robot vers les parties du graphe du web contenant le plus de documents pertinents. En plus de l'évolution de la précision du robot dans la limite de temps fixée, nous nous intéressons également à la quantité de documents pertinents rapatriés. Nous souhaitons ainsi mesurer l'efficacité d'une stratégie lorsque le temps devient une contrainte supplémentaire qui s'ajoute à la contrainte de précision. Les résultats

de ce test peuvent être utilisés pour mettre en évidence la stratégie la plus rentable pour une tâche donnée.

Exemple : Pour une tâche d'affinement de réponses d'un moteur de recherche à une requête utilisateur, ce test permet de connaître la stratégie à choisir pour fournir, dans une limite de temps donnée, plus de réponses pertinentes à l'utilisateur.

3.2 Résultats expérimentaux

3.2.1 Risque alimentaire

Nous avons récolté 2256 urls de départ à la suite de la soumission à Google de la réécriture de " Q_{Num} " en syntaxe Google. Après rapatriement et évaluation de ces points de départ sur la requête d'exploration (" Q_{Num} "), le critère de pertinence a révélé la présence de 71 documents pertinents, soit une précision de 3,14% pour Google.

Sur cette thématique, les résultats obtenus à l'issue de chacune des 4 explorations prévues dans notre protocole expérimental, se sont avérés extrêmement décevants (voir figure VI.3 page 151). En effet, aucune des 4 stratégies employées n'a réussi à trouver des documents pertinents en quantité suffisante pour valider l'expérimentation sur le domaine du risque alimentaire. Pour preuve, la meilleure précision finale atteinte (par l'exploration intelligente) n'a pas dépassé 0,4%, soit à peine une quarantaine de documents pertinents trouvés sur plus de 10.000 documents explorés. Ce taux extrêmement faible justifie l'abandon du reste des expérimentations sur cette thématique.

Remarque 3.3 *Les précisions indiquées sur l'axe des ordonnées des figures VI.3, VI.4, VI.5, VI.6 sont comprises entre 0 et 1 et ne sont donc pas en pourcentage.*

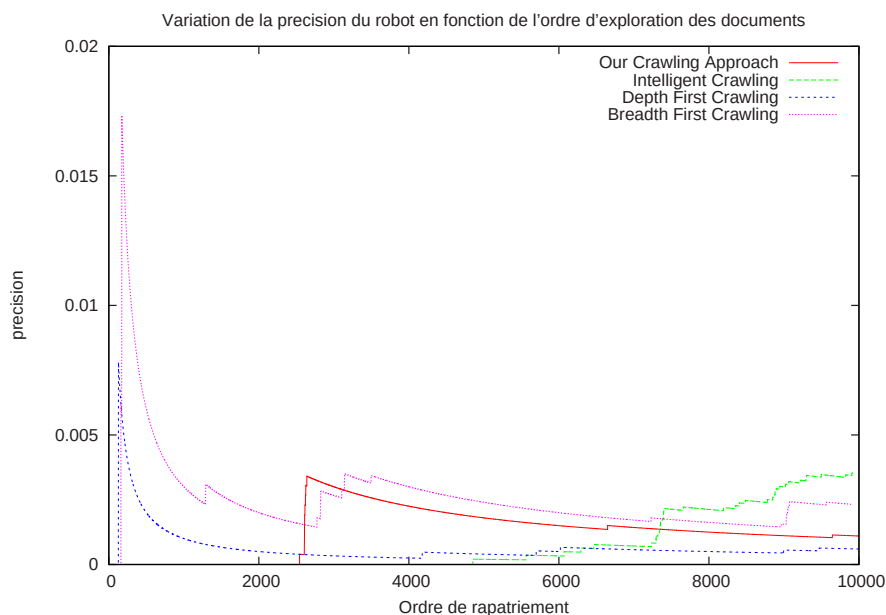


FIG. VI.3 – Courbe lift des explorations expérimentales à la recherche de documents traitant du risque alimentaire

3.2.2 Documents en-ligne de cours informatiques

Nous avons récolté 525 urls de départ à la suite de la soumission à Google de la réécriture de " Q_{info}^{num} " en syntaxe Google. Le critère de pertinence a révélé la présence de 63 documents pertinents parmi les points de départ, soit une précision de 12% pour Google.

Les résultats issus des différentes explorations menées sont montrées dans la figure VI.4 page 152.

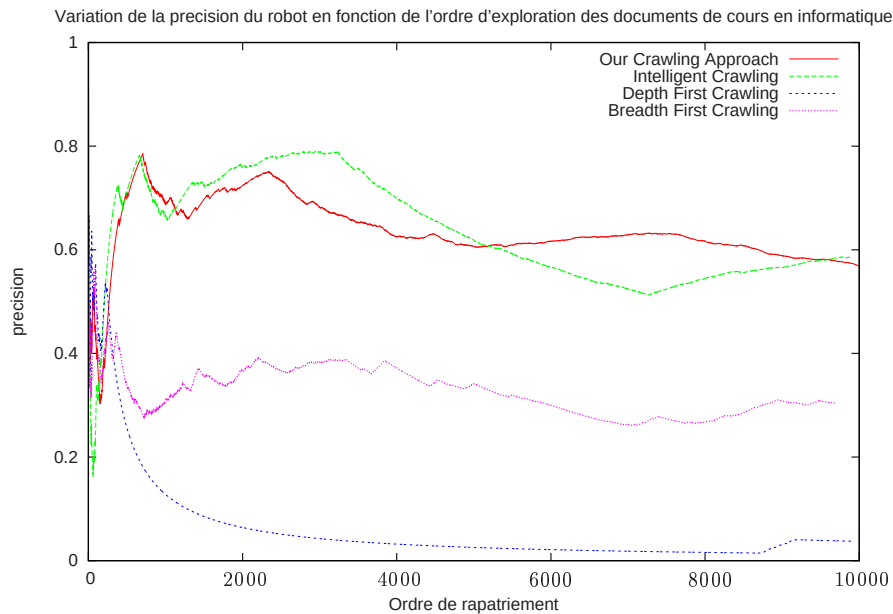


FIG. VI.4 – Courbe lift des explorations expérimentales à la recherche de documents de cours informatiques

Les résultats des différentes stratégies à la suite des deux tests de robustesse (stabilité et rendement) sont présentés respectivement dans la figure VI.5 page 153 et la figure VI.5 page 153.

Remarque 3.4 *Les courbes des approches systématiques dans la figure VI.5 page 153 sont données à titre indicatif.*

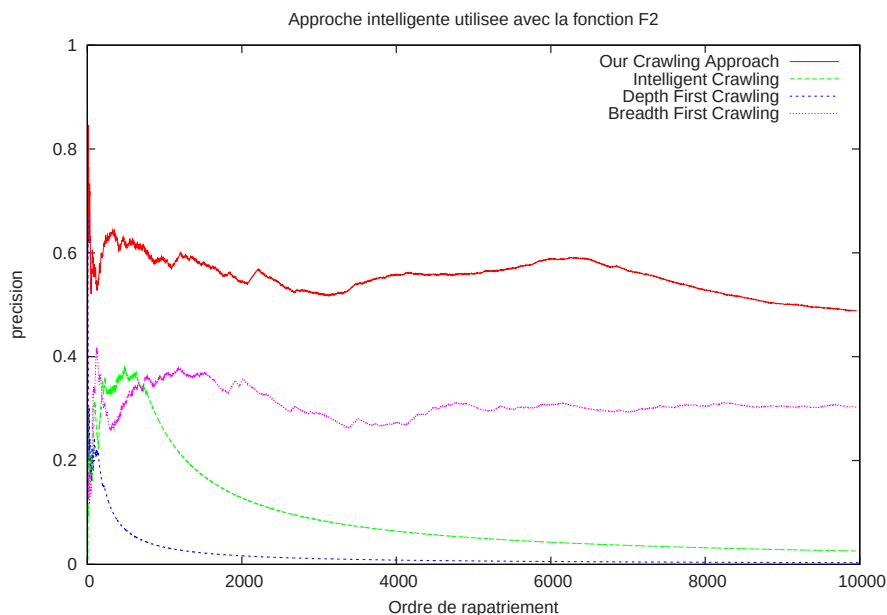


FIG. VI.5 – Courbe lift issue du test de stabilité sur la recherche de documents informatiques

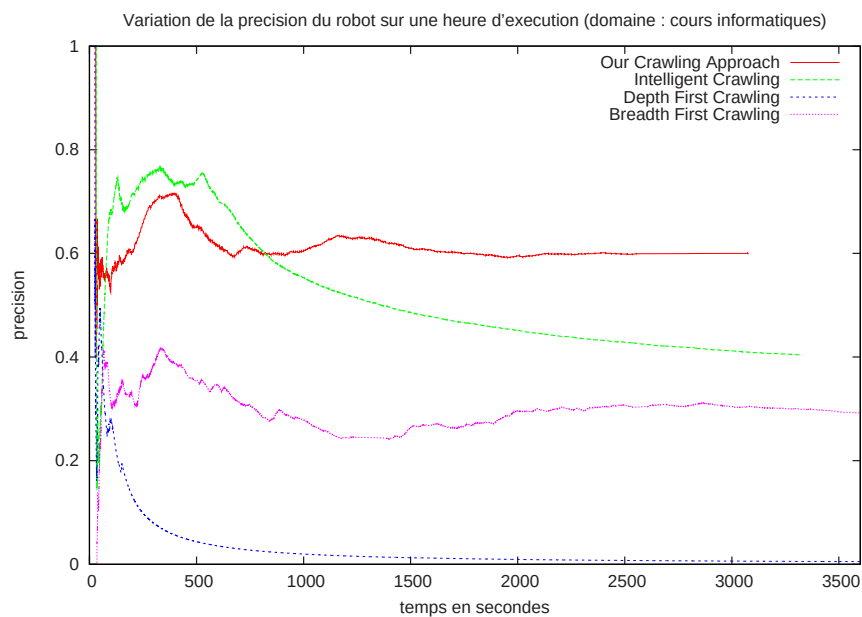


FIG. VI.6 – Courbe lift issue du test de rendement sur la recherche de documents informatiques

3.2.3 Discussion

Le risque alimentaire La médiocrité des résultats d'exploration obtenus sur cette thématique s'explique par la rareté de documents web correspondant au critère de pertinence fixé par nos experts. Cette rareté est confirmée, d'une part, par le peu de documents pertinents présents dans les réponses de Google aux requêtes caractérisant le risque alimentaire. D'autre part, l'extension de ces réponses par exploration n'a pas permis de recueillir significativement plus de documents pertinents.

Par ailleurs, les courbes des différentes stratégies de la figure VI.3 page 151 montrent que, dans une situation de pénurie de documents pertinents, les approches systématiques (largeur d’abord et profondeur d’abord) ont tendance à trouver plus rapidement les rares documents pertinents qui existent que les approches sélectives choisies dans notre protocole expérimental.

Pour l’approche « intelligente », nous expliquons ce phénomène par le mode de calcul de l’intérêt de rapatriement de chaque candidat. En effet, l’intérêt de rapatriement d’un candidat donné pour une approche intelligente s’appuie exclusivement sur les succès antérieurs de l’exploration. En d’autres mots, pour estimer la pertinence d’un candidat à sa juste valeur, l’approche intelligente doit absolument avoir déjà rencontré et rapatrié suffisamment de documents pertinents. Lorsque ceux-ci sont rares, il lui est difficile de statuer « objectivement » sur l’intérêt d’un candidat. Ceci entraîne le robot intelligent à parcourir le web de manière quasi-aléatoire.

Concernant notre approche, le peu de documents pertinents présents dans le web fait que le robot a également peu de chances de les rapatrier au départ. Pour un candidat réellement pertinent, l’estimateur *WEst* ne peut l’estimer correctement faute de cas similaires déjà rencontrés dans le graphe exploré. À la place, l’estimation qui est donnée à ce candidat correspond au score le plus vraisemblable parmi les scores observées chez les documents du graphe exploré ayant un voisinage similaire au candidat considéré. Autrement dit, si aucun score $s \in [s', s'']$ n’a été rencontré chez les documents du graphe exploré, aucun candidat ne sera approximé par l’intervalle $[s', s'']$. Ainsi, pour le risque alimentaire, le peu de documents pertinents existants explique le fait qu’ils aient été sous-estimés par l’estimateur *WEst*. Les priorités obtenues suite aux estimations de scores n’ont pas permis de privilégier les candidats réellement pertinents. Précisons néanmoins que, malgré que les documents pertinents aient été sous-estimés, les estimations de *WEst* se sont avérées exactes dans la majorité des cas. L’erreur moyenne dans l’estimation des candidats pour cette exploration n’a pas excédé 0.015.

Documents en-ligne de cours informatiques Les courbes d’élévation de la figure VI.4 page 152 montrent une nette supériorité des stratégies sélectives dans le rapatriement de documents pertinents par rapport au domaine de cours informatiques en-ligne. Cette supériorité s’explique principalement par l’absence de priorités de rapatriement dans les approches systématiques. En effet, alors que notre approche (ou l’approche intelligente) cible en priorité le rapatriement de documents pertinents, les explorations systématiques sont restreintes par la topologie du graphe du web à parcourir tous les documents non pertinents séparant les documents pertinents. Ceci montre l’utilité d’employer une méthode d’estimation de l’intérêt d’un candidat pour mettre en place un système de priorité de rapatriement. Des précédents résultats, nous pouvons également conclure à une relative égalité de performances entre notre approche et l’approche intelligente dans les conditions normales d’expérimentation. L’erreur moyenne dans l’estimation des scores des documents dans cette expérimentation pour notre approche est égale à 0.017. Ce résultat peut être interprété comme suit : en moyenne, l’estimateur *WEst* s’est trompé d’un centième dans l’estimation du score d’un document.

Sur les expérimentations de robustesse, le test de stabilité montre une dégradation considérable de la précision de l’approche intelligente, passant de 58% de précision avec une fonction de pertinence correcte (F_1^{info}) à 2,79% en utilisant une fonction de pertinence erronée (F_2^{info}). Cette dégradation s’explique par la forte dépendance qui existe entre le calcul du ratio d’intérêt de rapatriement d’un candidat et la fonction d’évaluation de la pertinence des documents rapatriés. En effet, le calcul du ratio d’intérêt de rapatriement d’un candidat repose exclusivement sur le résultat retourné par

la fonction booléenne d'évaluation des documents. Lorsque cette fonction renvoie souvent la valeur « vrai », l'approche intelligente accorde un ratio d'intérêt de plus en plus important aux candidats présentant un voisinage similaire aux voisinages des documents déjà rapatriés et évalués à « vrai ». Or, en choisissant un seuil $\delta^{info} = 0,2$ supérieur au seuil empirique de pertinence $\delta^{info} = 0,1$ pour la thématique des cours informatiques, nous avons volontairement construit la fonction F_2^{info} pour qu'elle renvoie peu de fois la valeur « vrai » (voir définition VI.14 page 149). Par conséquent, la rareté des documents satisfaisant F_2^{info} a entraîné un comportement quasi-aléatoire de l'exploration intelligente utilisant la fonction F_2^{info} .

Concernant notre approche, l'indépendance entre le seuil de pertinence et la fonction d'évaluation des documents garantit la stabilité de l'exploration. Autrement dit, quel que soit le critère de pertinence choisi, notre approche rapatrie en priorité les candidats susceptibles de totaliser les plus grands scores.

Le tableau ci-dessous résume, par stratégie, les quantités de documents satisfaisant le critère de pertinence défini pour la thématique des cours informatiques (voir section 3.1.1 page 146) :

Stratégie	Fonction d'évaluation des documents rapatriés	nombre de documents trouvés satisfaisant le critère de pertinence
Largeur d'abord	aucune	3036
Profondeur d'abord	aucune	54
Approche intelligente	F_1^{info}	5838
	F_2^{info}	279
Notre approche	$\mathcal{F}_*^{info}$	6206

TAB. VI.1 – Quantité de documents pertinents rapatriés lors du test de stabilité sur la thématique des cours informatiques en-ligne

À titre indicatif, le tableau ci-dessous montre la quantité de documents pertinents qu'aurait rapatriés chaque stratégie si le critère de pertinence correspondait au critère établi par la fonction F_2^{info} , c'est-à-dire 0,2 :

Stratégie	Fonction d'évaluation des documents rapatriés	nombre de documents trouvés satisfaisant le critère de pertinence établi par F_2^{info}
Largeur d'abord	aucune	38
Profondeur d'abord	aucune	0
Approche intelligente	F_1^{info}	4
	F_2^{info}	4
Notre approche	$\mathcal{F}_*^{info}$	39

TAB. VI.2 – Quantité de documents de cours informatiques rapatriés si le seuil de pertinence était de 0,2

Les résultats du test de rentabilité (voir figure VI.6 page 153) montrent que notre approche maintient une bonne précision tout le long de son exploration (60%) pour totaliser à la fin du temps

imparti la meilleure précision finale (59.40%).

Toutefois, les résultats du tableau VI.3 page 156 montrent que notre approche et l'approche « intelligente », respectivement première et deuxième approches en terme de précision, ont récolté moins de documents pertinents qu'une exploration en largeur d'abord (3^{ème} meilleure précision finale) dans les délais impartis. Ce résultat est dû à la lenteur de **webcrawler** dans son parcours du web. En effet, avec les approches sélectives, le robot **webcrawler** doit évaluer l'intérêt de tous les candidats de la pile d'urls avant de prendre une décision de rapatriement. Ces évaluations nécessitent un temps de calcul qui croît rapidement au fur et à mesure que de nouvelles urls viennent s'ajouter à la pile. N'ayant aucune évaluation de candidats à faire lors d'une exploration en largeur ou en profondeur d'abord, les prises de décision concernant le prochain candidat à rapatrier sont immédiates. L'exploration en largeur d'abord est plus rapide que les explorations nécessitant une estimation des candidats. Avec cette rapidité, l'exploration en largeur d'abord visite plus de documents et donc accroît ses chances de visiter en plus grand nombre des documents pertinents..

Stratégie	nombre de documents pertinents (score > 0.1) trouvés par webcrawler en 1 heure de temps	nombre de documents parcourus
Largeur d'abord	3520	12144
Profondeur d'abord	114	17796
Approche intelligente	2032	5042
Notre approche	1491	2510

TAB. VI.3 – Quantité de documents pertinents rapatriés en conditions normales d'expérimentation sur la thématique des cours informatiques en-ligne

4 Conclusion

Dans ce chapitre, nous avons présenté notre approche d’exploration sélective du web à la recherche de documents satisfaisant un critère de pertinence défini par une requête du langage WeQueL (requête d’exploration) et un seuil de pertinence. La comparaison du score d’évaluation de la requête d’exploration sur chaque document exploré permet de déterminer la pertinence de ces documents.

Notre approche d’exploration sélective du web guidée par une requête WeQueL consiste à estimer le score d’évaluation d’un candidat donné par la connaissance du score de certains documents de son voisinage, déjà rapatriés par le robot. Ce principe repose sur un système de calculs de probabilités conditionnelles réalisé par l’estimateur *WEst*. Pour éviter le recours à un graphe d’entraînement pour calculer les précédentes probabilités, l’estimateur *WEst* exploite un ensemble de statistiques de manière similaire à l’approche « intelligente ». L’objectif de notre approche est d’utiliser les estimations faites par *WEst* pour établir, à chaque étape, un classement de priorités entre les rapatriements des différents candidats.

Dans ce chapitre, nous avons mis en évidence le gain significatif en précision d’une recherche de documents thématiques par exploration sélective du web en utilisant notre approche. En effet, notre étude expérimentale montre que, sur la recherche de documents de cours informatiques, notre approche totalise une précision supérieure de deux fois à la précision atteinte par les explorations systématiques testées.

Par ailleurs, le test de stabilité et de rendement, définis dans notre protocole d’expérimentation, nous ont permis de montrer la stabilité et la fiabilité de notre approche dans des conditions d’exploration plus contraignantes (ex : critère de pertinence plus restrictif, limitation du temps d’exploration).

Troisième partie

Conclusion et perspectives

Conclusion et perspectives

*« It is a very sad thing that nowadays there is so little useless information »
Oscar WILDE (1854–1900). Dramaturge, poète, romancier irlandais*

Dans ce travail de thèse, nous avons présenté le langage de requêtes WeQueL qui permet de combiner plusieurs critères de recherche mots clés à la fois, ciblés chacun sur diverses parties et propriétés de la structure d'un document. Sur des recherches thématiques (cours informatiques en ligne, services touristiques en ligne et photos de formule 1), nous avons montré que notre langage permet d'exprimer des requêtes complexes pour décrire et détailler le plus finement possible les caractéristiques des documents souhaités.

Nous avons, par ailleurs, montré qu'il est possible de spécifier déclarativement différentes méthodes d'évaluation des requêtes atomiques combinées dans une requête WeQueL complexe. De plus, l'attribution de poids différents aux requêtes atomiques précédentes autorise les utilisateurs de notre langage à adapter le calcul du score de leurs requêtes en fonction de leurs besoins. Dans l'élaboration de la requête de filtrage du projet eDot par exemple, cette dernière possibilité nous a permis d'accorder une importance particulière au critère de pertinence défini par les experts microbiologistes, à savoir la présence des termes de l'ontologie d'eDot dans des tableaux HTML. Ceci s'est traduit par une requête atomique ciblant les tableaux HTML avec le poids le plus élevé des poids des requêtes atomiques de la requête de filtrage.

Afin de valider notre langage, nous avons réalisé une série d'expérimentations sur différentes tâches de recherches documentaires sur le web. Les résultats de cette étude expérimentale ont permis de mettre en évidence le gain significatif de la précision des requêtes de notre langage comparativement à des requêtes mots clés Google comparables. Cela a conforté notre intuition que l'amélioration de la précision de la recherche d'informations sur le web nécessite un langage de requête expressif combinant différents critères mots clés ciblés sur des parties spécifiques du contenu d'un document. Par ailleurs, les résultats expérimentaux de la tâche de filtrage du projet eDot soulignent l'intérêt et

l'utilité des requêtes WeQueL produisant un score numérique dans la mise en œuvre d'un classement de pertinence entre documents du web.

Nous avons également présenté, dans ce manuscrit, notre seconde contribution pour améliorer la recherche d'informations thématiques sur le web. Nous proposons une approche sélective d'exploration du web à la recherche de documents pertinents par rapport à une thématique décrite par une requête WeQueL (appelée requête d'exploration). La pertinence de chaque document exploré est déterminée par la comparaison de son score d'évaluation sur la requête d'exploration avec un seuil fixé avant le départ de l'exploration (seuil de pertinence). L'objectif de notre approche d'exploration consiste donc à guider le parcours du robot pour rapatrier en priorité les documents susceptibles de totaliser (sur la requête d'exploration) un score d'évaluation supérieur au seuil de pertinence. Pour atteindre cet objectif, l'approche que nous proposons repose sur le système *WEst* pour estimer le score d'évaluation de chaque candidat qui se présente lors de l'exploration. Le système *WEst* calcule, pour chaque candidat, un certain nombre de probabilités conditionnelles afin de déterminer le score le plus vraisemblable à associer à ce candidat compte tenu de son voisinage. De manière similaire à l'approche « intelligente » [Aggarwal et al. 2001], nous proposons de réaliser les précédents calculs de probabilités en s'appuyant sur un ensemble de statistiques recueillies « dynamiquement » depuis le graphe exploré pour décrire les caractéristiques du voisinage de chaque document exploré. Précisons que grâce à cet ensemble de statistiques, notre approche ne nécessite aucun apprentissage préalable.

Les estimations du système *WEst* permettent au robot explorateur d'établir un classement de priorité de rapatriement entre les différents candidats de l'exploration. Le premier de ce classement (« le meilleur ») correspond au candidat « pressenti » pour être le document susceptible de totaliser le score d'évaluation le plus élevé.

Sur cette partie de notre travail, nous avons présenté le protocole expérimental suivi lors de nos expérimentations pour évaluer la qualité d'une exploration utilisant notre approche pour rechercher des documents de cours informatiques. Nous avons pour cela utilisé deux mesures : le taux de moisson (*harvest rate*) et l'erreur moyenne dans l'estimation du score des candidats. Face aux stratégies concurrentes définies dans notre protocole expérimental (deux stratégies systématiques et une stratégie « intelligente » [Aggarwal et al. 2001]), le taux de moisson de notre approche est en moyenne deux fois plus important qu'une stratégie systématique et quasiment équivalent au taux totalisé par la stratégie « intelligente ». Par ces résultats, nous avons montré et confirmé la supériorité des explorations sélectives pour la recherche de documents thématiques.

Nous avons, par ailleurs, montré la stabilité de notre approche d'exploration du web par rapport au critère de pertinence choisi pour l'exploration. Pour la même requête d'exploration et les mêmes points de départ, la stabilité de notre approche se traduit par un parcours traversant les documents ayant les scores les plus élevés quelque soit le seuil de pertinence utilisé. Cette caractéristique a fait défaut à l'approche « intelligente » [Aggarwal et al. 2001] qui lors de l'utilisation d'une fonction plus restrictive pour évaluer la pertinence des documents explorés adopte un comportement quasi-aléatoire.

Perspectives

La méthode que nous proposons pour évaluer les requêtes du langage WeQueL exploite exclusivement les mots clés des valeurs spécifiées dans chaque requête atomique pour cibler leur recherche

sur des parties spécifiques d'un document. Nous avons expérimentalement vérifié qu'en procédant ainsi, nous garantissons une précision satisfaisante des réponses fournies à une requête WeQueL donnée. Cependant, nous avons souligné le faible rappel des résultats des requêtes de notre langage. Nous avons montré que la médiocrité du rappel d'une requête WeQueL donnée s'explique par le fait que l'utilisateur ne peut pas spécifier toutes les valeurs et tous les mots clés pertinents par rapport à sa recherche. Par conséquent, les documents pertinents employant un vocabulaire disjoint au vocabulaire des valeurs de la requête soumise ne sont pas couverts ou sont mal classés (avec de faibles scores). Une perspective intéressante à notre travail serait donc d'étudier les mécanismes appropriés pour augmenter le rappel de notre langage. Une solution possible serait par exemple d'intégrer des thésaurus en-ligne (WordNet [Felbaum 1998, Voorhees 1998, Miller et al. 1990]) ou des ontologies de domaines dans le moteur d'interprétation et d'évaluation du moteur de requêtes WeQueL. Dans ce cas de figure, ces outils auraient pour objectif d'enrichir le vocabulaire initial de chaque requête soumise avec des mots équivalents ou proches des mots initialement employés dans la requête.

Concernant notre contribution sur la partie exploration du web, nous avons vu que notre modèle de probabilité actuel d'estimation du score d'un candidat repose sur deux éléments : les scores des voisins rapatriés du candidat considéré et la répartition statistique des scores entre documents du graphe exploré (matrice de fratrie et matrice de parenté). Bien que l'erreur d'estimation moyenne sur nos différentes expérimentations n'excédait pas 10^{-2} , nous avons remarqué que certains candidats pertinents ont été sous-estimés par l'estimateur *WEst*. Compte tenu de leur voisinage et du graphe exploré, les scores qui leur ont été attribués correspondaient aux scores les plus vraisemblables c'est-à-dire les scores les plus répandus dans le graphe exploré. Ce phénomène s'avère être pénalisant pour les explorations recherchant des documents rares dans le web. Dans ce cas par exemple, le peu de documents pertinents présents sont estimés avec les scores les plus répandus dans le graphe exploré (ces scores sont sans doute très inférieurs au seuil de pertinence). Ceci a donc pour conséquence d'empêcher la mise en évidence de ces documents pertinents dans le classement de la file d'attente.

Une perspective intéressante serait donc de minimiser les effets « pervers » de l'estimation par la vraisemblance du score d'un candidat. Une solution envisageable consisterait à intégrer la description textuelle entrante (les mots clés dans l'ancre et le voisinage de mots des liens entrants) des candidats dans la méthode d'estimation des scores. Par un apprentissage semblable à celui que nous proposons pour estimer les scores des candidats à partir de la répartition statistique dans le graphe exploré, l'estimateur *WEst* pourrait traduire les occurrences des mots des liens entrants du graphe exploré en un ensemble de statistiques pour permettre le calcul de l'influence de leur présence dans la détermination du score d'un candidat. Lorsque la requête d'exploration combine des requêtes atomiques portant sur la description entrante des documents (*title_incoming_page* et/ou *incoming_links*), il serait intéressant d'exploiter les valeurs mots clés de ces deux requêtes atomiques dans l'estimation des candidats. Par exemple, ayant la description textuelle entrante de chaque candidat, nous pouvons envisager le calcul préliminaire de leur score sur ces requêtes atomiques. Ce score préliminaire (qui n'est pas une approximation mais bien une évaluation exacte) peut par la suite être combiné au score estimé du candidat considéré par l'estimateur *WEst*. Le but d'une telle combinaison serait de permettre aux candidats potentiellement pertinents, pointés par des liens ayant une description textuelle similaire aux valeurs des requêtes atomiques *title_incoming_page* et/ou *incoming_links*, d'émerger dans le classement de la file d'attente lorsqu'ils sont estimés avec les mêmes scores que les autres candidats (à priori non intéressants).

Une seconde perspective à notre travail sur l’exploration du web consiste à optimiser le processus de mise à jour des estimations des candidats en fonction des changements des mesures statistiques du graphe exploré. En effet, la croissance du graphe exploré entraîne une forte multiplication du nombre de candidats accessibles depuis ce graphe. Sur le test de rendement, nous avons vu que cette multiplication engendre un temps de calcul important pour estimer les scores de tous les candidats de l’exploration. L’avancée du robot dans le web est considérablement retardée lorsque nous la comparons à celle d’un robot muni d’une stratégie systématique où le choix du prochain candidat est quasi-immédiat. Bien que la précision ne soit pas affectée, le surcoût de calcul pénalise notre approche en terme de quantité de documents pertinents rapatriés. Pour contourner ce problème, nous avons substitué le processus de mise à jour systématique des estimations des candidats (c’est-à-dire après chaque rapatriement) par un processus périodique (tous les k rapatriements). Nous avons fixé la périodicité de ce processus à la suite de nos observations expérimentales de sorte à garantir que le fait de geler les estimations des candidats pendant k rapatriements ne remet pas en cause le classement naturel des candidats tel qu’il aurait pu être établi par une mise à jour systématique. Pour que ceci soit possible, la période doit nécessairement être petite. Nous l’avons fixé en conséquence à environ 25 rapatriements. Ceci n’a pas suffi pour limiter les effets du surcoût de calcul des estimations liés à la croissance de la taille de la file d’attente des candidats.

Il serait donc intéressant d’étudier d’autres heuristiques plus élaborées contribuant à la réduction du temps nécessaire à l’estimation du score des candidats de l’exploration. Une possibilité serait, par exemple, de remarquer qu’au delà d’une certaine taille de la file d’attente, les candidats mals classés ont peu de chances de bondir des dernières positions vers les premières positions du classement après seulement quelques rapatriements. Il est alors possible d’envisager une stratégie mixte de mise à jour : systématique pour les N premiers du classement et périodique pour les autres (avec une période plus longue).

Table des figures

I.1	Architecture fonctionnelle du projet « eDot »	11
II.1	Schématisation de la notion de Hub et Authority	22
II.2	Représentation <i>AST</i> d'un document HTML sous WebOQL	33
II.3	Évaluation d'une requête WebOQL sur l'arbre <i>AST</i> de la figure II.2 page 33	34
II.4	Récapitulatif de la classification des langages de requêtes pour le web	37
III.1	schéma XML associé à une requête WeQueL	53
III.2	schéma XML associé à la requête WeQueL Q de l'exemple de la page 51	55
III.3	Exemple d'un document HTML transformé en une table du modèle <i>WDM</i>	57
III.4	Q_{info} : Requête descriptive des cours informatiques en ligne	69
III.5	Q_{F1} : Requête descriptive des photos de formule 1 en ligne	70
III.6	Q_{tour} : Requête descriptive des services de tourisme en ligne	71
III.7	La requête " Q_{Num} " de filtrage	78
III.8	La requête " Q_{Log} " de filtrage	80
III.9	La courbe ROC idéale	82
III.10	La courbe LIFT idéale	84
III.11	Résultats expérimentaux du filtrage : Courbe ROC	85
III.12	Résultats expérimentaux du filtrage : Courbe LIFT	86
IV.1	Lien entrant / sortant	92
IV.2	Voisinage d'un document	93
IV.3	Architecture et fonctionnement général d'un robot explorateur du web	95
IV.4	La partition du web engendrée par un robot explorateur	96
V.1	Exploration en largeur d'abord	104
V.2	Exploration en profondeur d'abord	106
V.3	Exemple d'un graphe contextuel d'entraînement	116
V.4	Parents et frères d'un candidat	129
V.5	Classification des différentes techniques d'exploration du web	132
VI.1	Exemple d'estimation du score d'un candidat	143
VI.2	Requête " Q_{info}^{num} " pour la recherche de documents de cours en informatique	147
VI.3	Courbe lift des explorations expérimentales à la recherche de documents traitant du risque alimentaire	151
VI.4	Courbe lift des explorations expérimentales à la recherche de documents de cours informatiques	152

VI.5 Courbe lift issue du test de stabilité sur la recherche de documents informatiques . .	153
VI.6 Courbe lift issue du test de rendement sur la recherche de documents informatiques .	153

Liste des tableaux

I.1	Dates importantes de l'évolution d'internet et du web	2
II.1	Tableau de l'alphabet utilisé dans les expressions régulières du langage WebSQL . . .	28
II.2	Tableau comparatif des langages de requêtes pour le web	39
III.1	Syntaxe d'une requête atomique WeQueL	44
III.2	Exemple d'une requête atomique ciblant l'url des documents à évaluer	47
III.3	Exemple d'une requête atomique ciblant le titre des documents à évaluer	47
III.4	Exemple d'une requête atomique ciblant le titre des documents à évaluer	47
III.5	Exemple d'une requête atomique ciblant les mots apparaissant dans les liens entrants	47
III.6	Syntaxe d'une combinaison WeQueL par catégorie de documents	48
III.7	Compatibilité des requêtes atomiques WeQueL avec les types MIME de documents .	49
III.8	Exemple d'une combinaison de requêtes atomiques WeQueL	50
III.9	Syntaxe d'une requête WeQueL complexe	51
III.10	Exemple d'une requête WeQueL complexe	52
III.11	Conversion de la syntaxe abstraite d'une requête WeQueL en syntaxe XML	54
III.12	Calcul du score d'une requête atomique en fonction du constructeur sémantique employé	59
III.13	Table WDM associée au document d_1	61
III.14	Table WDM associée au document d_2	61
III.15	Table WDM associée au document d_3	61
III.16	Table WDM associée au document d_4	62
III.17	Table WDM associée au document d_5	62
III.18	Ensembles $REGEX$ générés à partir des valeurs des requêtes atomiques de la page 47	62
III.19	Ensembles MV générés à la suite de l'évaluation des requêtes atomiques de la page 47 sur l'ensemble $\mathcal{D}$	63
III.20	Ensembles MW générés à la suite de l'évaluation des requêtes atomiques de la page 47 sur l'ensemble $\mathcal{D}$	63
III.21	Scores d'évaluation des documents de l'ensemble $\mathcal{D}$ sur la requête atomique Q_a^{ex1} . .	63
III.22	Scores d'évaluation des documents de l'ensemble $\mathcal{D}$ sur la requête atomique Q_a^{ex2} . .	64
III.23	Scores d'évaluation des documents de l'ensemble $\mathcal{D}$ sur la requête atomique Q_a^{ex3} . .	64
III.24	Scores d'évaluation des documents de l'ensemble $\mathcal{D}$ sur la requête atomique Q_a^{ex4} . .	64
III.25	Évaluation d'une combinaison de requêtes atomiques sur un exemple de documents .	65
III.26	Calcul du score d'une requête complexe suivant la fonction d'évaluation employée . .	66
III.27	Évaluation d'une requête WeQueL complexe sur un exemple de documents	67
III.28	Taille des trois graphes thématiques d'expérimentation	74
III.29	Précision et couverture des requêtes thématiques du langage WeQueL	75
III.30	Classification de la qualité d'un système à partir de l'aire sous sa courbe ROC	82

IV.1 Exemple illustratif de la profondeur du document dans le graphe exploré par un robot	98
VI.1 Quantité de documents pertinents rapatriés lors du test de stabilité sur la thématique des cours informatiques en-ligne	155
VI.2 Quantité de documents de cours informatiques rapatriés si le seuil de pertinence était de 0,2	155
VI.3 Quantité de documents pertinents rapatriés en conditions normales d'expérimentation sur la thématique des cours informatiques en-ligne	156

Liste des Algorithmes

1	Construction d'un graphe contextuel d'entraînement	117
---	--	-----

Bibliographie

- [Abiteboul 1997] SERGE ABITEBOUL (1997). « Querying Semi-Structured Data ». Dans *Proceedings of the 6th International Conference on Database Theory*, pages 1–18. Springer-Verlag.
@ : <http://citeseer.ist.psu.edu/abiteboul97querying.html>.
Cité page 36
- [Abiteboul et al. 2003] SERGE ABITEBOUL AND MIHAI PREDAN AND GREGORY COBENA (2003). « Adaptive on-line page importance computation ». Dans *Proceedings of the twelfth international conference on World Wide Web*, pages 280–290. ACM Press.
@ : <http://www2003.org/cdrom/papers/refereed/p007/p7-abiteboul.html>.
Cité pages 79,107
- [Abiteboul et al. 1997] SERGE ABITEBOUL AND DALLAN QUASS AND JASON MCHUGH AND JENNIFER WIDOM AND JANET L. WIENER (1997). « The Lorel query language for semistructured data ». *International Journal on Digital Libraries*, 1(1) :68–88.
@ : <http://citeseer.ist.psu.edu/article/abiteboul97lorel.html>.
Cité page 36
- [Aggarwal et al. 2001] CHARU C. AGGARWAL AND FATIMA AL-GARAWI AND PHILIP S. YU (2001). « Intelligent crawling on the World Wide Web with arbitrary predicates ». Dans *World Wide Web*, pages 96–105. ACM Press.
@ : <http://citeseer.nj.nec.com/aggarwal01intelligent.html>.
Cité pages 119,121,130,136,137,141,146,162
- [AltaVista site] ALTAVISTA (website). « Le moteur de recherche AltaVista ».
@ : <http://www.altavista.com>.
Cité pages 20,21,29,105
- [ARANEUS 1998] ARANEUS (1998). « The ARANEUS Project ».
@ : <http://www.dia.uniroma3.it/Araneus/>.
Cité page 34
- [Arocena et Mendelzon 1998] G. AROCENA AND A. MENDELZON (1998). « WebOQL : Restructuring Documents, Databases and Webs ». Dans *Proc. of 14th. Intl. Conf. on Data Engineering (ICDE 98)*.
@ : <http://www.cs.toronto.edu/~gus/webowl/webowl.ps.gz>.
Cité pages 31,32
- [Arocena 1997] GUSTAVO O. AROCENA (1997). « WebOQL : Exploiting Document Structure in Web Queries ». Master’s thesis, University of Toronto, Canada.

@ : <http://www.cs.toronto.edu/~gus/thesis.ps.gz>.

Cité page 32

[Arocena et al. 1997] GUSTAVO O. AROCENA AND ALBERTO O. MENDELZON AND GEORGE A. MIHAILA (1997). « Applications of a Web query language ». Dans *Proceedings of the sixth international conference on World Wide Web, Santa Clara, California, United States*, pages 589–596. Elsevier Science Publishers Ltd.

@ : <http://www.cs.toronto.edu/~websql/www-conf/wsql/PAPER267.html>.

Cité page 26

[Atzeni et al. 1997] PAOLO ATZENI AND GIANSAVATORE MECCA AND PAOLO MERIALDO (1997). « To Weave the Web ». Dans *VLDB '97*, pages 206–215.

@ : <http://citeseer.ist.psu.edu/article/atzeni97to.html>.

Cité pages 31,34,35

[Austin site] UNIVERSITY TEXAS AUSTIN (website). « Site d'accueil du département informatique de l'université de Texas, Austin USA ».

@ : <http://www.cs.utexas.edu/home/home/>.

Cité page 115

[Azé 2003] J. AZÉ (2003). « *Extraction de Connaissances dans des Données Numériques et Textuelles* ». PhD thesis, Université de Paris 11.

@ : <http://www.lri.fr/~aze/TheseHtml/index.html>.

Cité page 123

[Bancilhon et Ferran 1994] FRANCOIS BANCILHON AND GUY FERRAN (1994). « ODMG-93 : The Object Database Standard ». *Data Engineering Bulletin*, 17(4) :3–14.

@ : <http://citeseer.ist.psu.edu/357206.html>.

Cité page 31

[Beeri et al. 1998] CATRIEL BEERI AND GERSHON ELBER AND TOVA MILO AND YEHOASHUA SAGIV AND ODED SHMUELI AND NAFTALI TISHBY AND YAKOV A. KOGAN AND DAVID KOPNICKI AND PINI MOGILEVSKI AND NOAM SLONIM (1998). « WebSuite : A Tool Suite for Harnessing Web Data ». Dans *International Workshop on the Web and Databases*, pages 152–171.

@ : <http://citeseer.ist.psu.edu/beeri98websuite.html>.

Cité page 29

[Berners-Lee et Connolly 1995] T. BERNERS-LEE AND D. CONNOLLY (November 1995). « RFC 1866 : Hypertext Markup Language - 2.0 ».

@ : <http://www.ietf.org/rfc/rfc1866.txt>.

Cité page 45

[Berners-Lee et al. 1994] T. BERNERS-LEE AND L. MASINTER AND M. MCCAHILL (December 1994). « RFC 1738 : Uniform Resource Locators (URL) ».

@ : <http://www.ietf.org/rfc/rfc1738.txt>.

Cité page 91

[Brin et Page 1998] SERGEY BRIN AND LAWRENCE PAGE (1998). « The anatomy of a large-scale hypertextual Web search engine ». *Computer Networks and ISDN Systems*, 30(1–7) :107–117.

@ : <http://citeseer.nj.nec.com/brin98anatomy.html>.

Cité pages 3,7,20,21,41,74,107,108

- [Brown site] UNIVERSITY BROWN (website). « Site d'accueil du département informatique de l'université de Brown, Providence USA ».
@ : <http://www.cs.brown.edu/>.
Cité page 112
- [Cattell et al. 2000] R. CATTELL AND DOUGLAS BARRY AND MARK BERLER AND JEFF EASTMAN AND DAVID JORDAN AND CRAIG RUSSELL AND OLAF SCHADOW AND TORSTEN STANIENDA AND FERNANDO VELEZ AND R. G. G. CATTELL (January 19, 2000). « *The Object Data Standard : ODMG 3.0* ». Morgan Kaufmann.
Cité page 31
- [Cho et al. 1998] JUNGHOO CHO AND HECTOR GARCÍA-MOLINA AND LAWRENCE PAGE (1998). « Efficient crawling through URL ordering ». *Computer Networks and ISDN Systems*, 30(1-7) :161-172.
@ : <http://citeseer.nj.nec.com/cho98efficient.html>.
Cité pages 107,108
- [Cora site] WHIZBANG CORA (website). « Moteur de recherche spécialisé dans les publications informatiques ».
@ : <http://cora.whizbang.com>.
Cité page 112
- [Cornell site] UNIVERSITY CORNELL (website). « Site d'accueil du département informatique de l'université de Cornell, Ithaca USA ».
@ : <http://www.cis.cornell.edu/>.
Cité page 112
- [Cornuéjols et al. 2002a] ANTOINE CORNUÉJOLS AND LAURENT MICLET AND YVES KODRATOFF (2002a). « *Apprentissage artificiel, Concepts et algorithmes* », chapter 16 : L'apprentissage de reflexes par renforcement. Éditions Eyrolles.
Cité page 112
- [Cornuéjols et al. 2002b] ANTOINE CORNUÉJOLS AND LAURENT MICLET AND YVES KODRATOFF (2002b). « *Apprentissage artificiel, Concepts et algorithmes* », chapter 14 : L'apprentissage bayésien et son approximation. Éditions Eyrolles.
Cité page 112
- [De Bra et Post 1994] P. M. E. DE BRA AND R. D. J. POST (1994). « Information retrieval in the World-Wide Web : Making client-based searching feasible ». *Computer Networks and ISDN Systems*, 27(2) :183-192.
@ : <http://citeseer.ist.psu.edu/debra94information.html>.
Cité page 109
- [Diligenti et al. 2000] MICHELANGELO DILIGENTI AND FRANS COETZEE AND STEVE LAWRENCE AND C. LEE GILES AND MARCO GORI (2000). « Focused Crawling using Context Graphs ». Dans *26th International Conference on Very Large Databases, VLDB 2000*, pages 527-534, Cairo, Egypt.
@ : <http://citeseer.nj.nec.com/diligenti00focused.html>.
Cité pages 74,112,115,136

- [eDot site] eDOT (website). « Site du Projet RNTL e.dot ».
@ : <http://www-rocq.inria.fr/gemo/edot/>.
Cité pages 9,24,68,77
- [Ester et al. 1996] MARTIN ESTER AND HANS-PETER KRIEGLER AND JÖRG SANDERS AND XIAO-WEI XU (1996). « A density based algorithm for discovering clusters in large spatial databases with noise ». Dans *Proceedings of the 2nd International conference on Knowledge Discovery and Data Mining, Portland, USA*.
@ : http://www.cs.ualberta.ca/~joerg/papers/KDD-96_final.pdf.
Cité page 24
- [Felbaum 1998] C. Felbaum, editor (1998). « *WordNet : an electronic lexical database* ». Boston : MIT Press.
Cité pages 24,163
- [Fernandez et al. 1998a] MARY FERNANDEZ AND DANIELA FLORESCU AND JAEWOO KANG AND ALON LEVY AND DAN SUCIU (1998a). « Catching the boat with Strudel : experiences with a Web-site management system ». Dans *Proceedings of the 1998 ACM SIGMOD international conference on Management of data*, pages 414–425. ACM Press.
@ : <http://www.research.att.com/~mff/strudel/doc/files/sigmod98.ps.gz>.
Cité page 36
- [Fernandez et al. 1997a] M. FERNANDEZ AND D. FLORESCU AND A. LEVY AND D. SUCIU (1997a). « A Query Language and Processor for a Web-Site Management System ». Dans *Proc. Workshop on Management of Semistructured Data*, Tucson.
@ : <http://citeseer.ist.psu.edu/article/fernandez97query.html>.
Cité page 36
- [Fernandez et al. 1998b] MARY F. FERNANDEZ AND DANIELA FLORESCU AND JAEWOO KANG AND ALON Y. LEVY AND DAN SUCIU (1998b). « Overview of Strudel - A Web-Site Management System ». *Networking and Information Systems*, 1(1) :115–140.
@ : <http://citeseer.ist.psu.edu/fernandez98overview.html>.
Cité pages 31,36
- [Fernandez et al. 1997b] MARY F. FERNANDEZ AND DANIELA FLORESCU AND ALON Y. LEVY AND DAN SUCIU (1997b). « A Query Language for a Web-Site Management System ». *SIGMOD Record*, 26(3) :4–11.
@ : <http://citeseer.ist.psu.edu/article/fernandez97query.html>.
Cité page 36
- [Ferri et al. 2002] C. FERRI AND P. FLACH AND J. HERNANDEZ-ORALLO (2002). « Learning decision trees using the area under the ROC curve ». Dans *Proceedings of 9th International Conference on Machine Learning, ICML'02*, pages 139–146.
Cité page 81
- [Fielding et al. 1997] R. FIELDING AND UC IRVINE AND J. GETTYS AND J. MOGUL AND H. FRYSTYK AND T. BERNERS-LEE (January 1997). « RFC 2068 : Hypertext Transfer Protocol – HTTP/1.1 ».
@ : <http://www.ietf.org/rfc/rfc2068.txt>.
Cité page 94

- [Florescu et al. 1998] DANIELA FLORESCU AND ALON LEVY AND ALBERTO MENDELZON (1998). « Database Techniques for the World-Wide Web : A Survey ». *SIGMOD Record*, 27(3) :59–74.
@ : <http://www.cs.washington.edu/homes/alon/site/files/sigrec98.ps>.
Cité pages 24,26
- [Freed et Borenstein 1996a] N. FREED AND N. BORENSTEIN (November 1996a). « RFC 2045 : Multipurpose Internet Mail Extensions (MIME) Part One : Format of Internet Message Bodies-MIME Media Types ».
@ : <http://www.ietf.org/rfc/rfc2045.txt>.
Cité pages 22,32,43,46
- [Freed et Borenstein 1996b] N. FREED AND N. BORENSTEIN (November 1996b). « RFC 2046 : Multipurpose Internet Mail Extensions (MIME) Part Two : Media Types ».
@ : <http://www.ietf.org/rfc/rfc2046.txt>.
Cité pages 22,32,43,46
- [Freed et Borenstein 1993] N. FREED AND N. BORENSTEIN (September 1993). « RFC 1521 : MIME (Multipurpose Internet Mail Extensions) Part One : Mechanisms for Specifying and Describing the Format of Internet Message Bodies ».
@ : <http://www.ietf.org/rfc/rfc1521.txt>.
Cité pages 22,32,43,46
- [Google site] GOOGLE (website). « Google classique ».
@ : <http://www.google.com>.
Cité pages 1,3,4,20,21,72,101,105
- [Google_advanced site] GOOGLE_ADVANCED (website). « Advanced Google ».
@ : http://www.google.fr/advanced_search.
Cité pages 22,72,148
- [Google_API site] GOOGLE_API (website). « Google Web API ».
@ : <http://www.google.com/apis/>.
Cité pages 73,116
- [Halkidi et al. 2002] MARIA HALKIDI AND BENJAMIN NGUYEN AND IRAKLIS VARLAMIS AND MICHALIS VAZIRGIANNIS (octobre 2002). « Organising Web Documents into Thematic Subsets using an Ontology (THESUS) ». Dans *Actes électroniques des Journées Web Semantique*, Paris.
@ : http://www.lalic.paris4.sorbonne.fr/stic/octobre/octobre6/22_Nguyen.pdf.
Cité pages 20,23
- [Hersovici et al. 1998] MICHAEL HERSOVICI AND MICHAL JACOVI AND YOELLE S. MAAREK AND DAN PELLEG AND MENANCHEM SHTALHAIM AND SIGALIT UR (1998). « The shark-search algorithm. An application : tailored Web site mapping ». *Comput. Netw. ISDN Syst.*, 30(1-7) :317–326.
@ : <http://www.ra.ethz.ch/CDstore/www7/1849/com1849.htm>.
Cité page 110
- [Heydon et Najork 1999] ALLAN HEYDON AND MARC NAJORK (1999). « Mercator : A Scalable, Extensible Web Crawler ». *World Wide Web*, 2(4) :219–229.
@ : <http://citeseer.nj.nec.com/heydon99mercator.html>.
Cité pages 6,105

- [Kaelbling et al. 1996] LESLIE PACK KAEHLING AND MICHAEL L. LITTMAN AND ANDREW P. MOORE (1996). « Reinforcement Learning : A Survey ». *Journal of Artificial Intelligence Research*, 4 :237–285.
@ : <http://www.cs.washington.edu/research/jair/volume4/kaelbling96a-html/r1%-survey.html>.
Cité page 112
- [Kleinberg 1999] JON M. KLEINBERG (1999). « Authoritative sources in a hyperlinked environment ». *Journal of the ACM*, 46(5) :604–632.
@ : <http://citeseer.nj.nec.com/kleinberg97authoritative.html>.
Cité pages 21,74,107,128
- [Konopnicki et Shmueli 1995] DAVID KONOPNICKI AND ODED SHMUELI (1995). « W3QS : A Query System for the World-Wide Web ». Dans *Proceedings of the 21th International Conference on Very Large Data Bases*, pages 54–65. Morgan Kaufmann Publishers Inc.
@ : <http://www.vldb.org/conf/1995/P054.PDF>.
Cité page 29
- [Konopnicki et Shmueli 1998] DAVID KONOPNICKI AND ODED SHMUELI (1998). « Information gathering in the World-Wide Web : the W3QL query language and the W3QS system ». *ACM Trans. Database Syst.*, 23(4) :369–410.
@ : <http://www.cs.technion.ac.il/~konop/todsonline.ps.gz>.
Cité page 28
- [Lewis 1998] DAVID D. LEWIS (1998). « Naive (Bayes) at forty : The independence assumption in information retrieval. ». Dans C. Nédellec et C. Rouveirol, editors, *Proceedings of ECML-98, 10th European Conference on Machine Learning*, number 1398, pages 4–15, Chemnitz, DE. Springer Verlag, Heidelberg, DE.
@ : <http://citeseer.ist.psu.edu/lewis98naive.html>.
Cité page 112
- [Lycos site] LYCOS (website). « Le portail internet Lycos ».
@ : <http://www.lycos.com>.
Cité pages 18,26
- [Martijn Koster 1996] MARTIJN KOSTER (1996). « The Web Robots Pages : Evaluation of the Standard for Robot Exclusion ».
@ : <http://www.robotstxt.org/wc/eval.html>.
Cité page 94
- [McBryan 1994] OLIVER A. MCBRYAN (1994). « GENVL and WWW : Tooles for Taming the web ». Dans *First International Conference on the World Wide Web*, pages 79–90.
@ : <http://www.cs.colorado.edu/home/mcbryan/mypapers/www94.ps>.
Cité page 20
- [McCallum et al. 1999] ANDREW MCCALLUM AND KAMAL NIGAM AND JASON RENNIE AND KRISTIE SEYMORE (1999). « Building domain-specific search engines with machine learning techniques ». Dans *Proc. AAAI-99 Spring Symposium on Intelligent Agents in Cyberspace, 1999*.
Cité page 112

- [Menczer 1997] FILIPPO MENCZER (1997). « ARACHNID : Adaptive Retrieval Agents Choosing Heuristic Neighborhoods for Information Discovery ». Dans *Machine Learning : Proceedings of the Fourteenth International Conference*, pages 227–235.
@ : <http://citeseer.ist.psu.edu/menczer97arachnid.html>.
Cité pages 119,120
- [Menczer 2003] FILIPPO MENCZER (2003). « Complementing search engines with online web mining agents ». *Decis. Support Syst.*, 35(2) :195–212.
Cité page 119
- [Menczer et al. 2004] FILIPPO MENCZER AND GAUTAM PANT AND PADMINI SRINIVASAN (2004). « Topical web crawlers : Evaluating adaptive algorithms ». *ACM Trans. Inter. Tech.*, 4(4) :378–419.
@ : <http://dollar.biz.uiowa.edu/~pant/Papers/T0IT.pdf>.
Cité page 119
- [Mendelzon et al. 1996] ALBERTO O. MENDELZON AND GEORGE A. MIHAILA AND TOVA MILO (1996). « Querying the World Wide Web ». Dans *Proceedings of the fourth international conference on Parallel and distributed information systems, Miami Beach, Florida, United States*, pages 80–91. IEEE Computer Society.
@ : <http://citeseer.ist.psu.edu/mendelzon97querying.html>.
Cité page 26
- [Mendelzon et al. 1997] ALBERTO O. MENDELZON AND GEORGE A. MIHAILA AND TOVA MILO (1997). « Querying the World Wide Web ». *Int. J. on Digital Libraries*, 1(1) :54–67.
@ : <http://citeseer.ist.psu.edu/article/mendelzon97querying.html>.
Cité page 26
- [Mezaour 2001] AMAR-DJALIL MEZAOUR (2001). « Conception d’un robot explorateur du web sélectif en utilisant des techniques d’apprentissage. Mémoire de stage de DEA ».
@ : <http://www.lri.fr/~mezaour/publications/rapport.ps>.
Cité page 112
- [Mezaour 2003] AMAR DJALIL MEZAOUR (2003). « Focused Search on the Web using WeQueL ». Dans *Proceedings of the 10th International Workshop on Knowledge Representation meets Databases (KRDB 2003), Hamburg, Germany*, pages 63–74.
@ : <http://www.lri.fr/~mezaour/publications/kldb2003mezaour.ps>
<http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-79/mezaour.pdf>.
Cité page 37
- [Mezaour 2004a] AMAR DJALIL MEZAOUR (2004a). « Filtering Web Documents for eDot, a food risk warehouse ». Dans *Proceedings of the 2nd International Conference on Computational Intelligence (ICCI 2004), Istanbul, Turkey*, pages 249–252. Prof. Dr. Ali OKATAN.
@ : <http://www.lri.fr/~mezaour/publications/icci2004mezaour.doc>.
Cité page 77
- [Mezaour 2004b] AMAR DJALIL MEZAOUR (2004b). « Recherche ciblée de documents sur le web ». *Revue des Nouvelles Technologies de l’Information (RNTI-E2), numéro spécial EGC’2004*, 2 :491–502.

@ : <http://www.lri.fr/~mezaour/publications/egc2004mezaour.ps>.

Cité page 37

[Mezaour 2005] AMAR DJALIL MEZAOUR (2005). « Filtering Web Documents for a Thematic Warehouse, Case Study : eDot a Food Risk Data Warehouse (extended) ». Dans *Proceedings of New Trends in Intelligent Information Processing and Web Mining Conference (IIPWM'05), Gdansk, Poland*. Springer Verlag series-Advances in Soft Computing-.

@ : <http://www.lri.fr/~mezaour/publications/iis2005mezaour.ps>.

Cité page 77

[Mignet et al. 2001] LAURENT MIGNET AND VINCENT AGUILERA AND SEBASTIEN AILLERET AND PIERANGELO VELTRI (2001). « Xyro : The Xyleme Robot Architecture ». Dans *DIWeb*, pages 91–99.

@ : citeseer.ist.psu.edu/mignet01xyro.html.

Cité pages 79,108

[Mihaila 1996] GEORGE ANDREI MIHAILA (1996). « WebSQL - An SQL-like Query Language for the World Wide Web ». Master's thesis, University of Toronto, Canada.

@ : <http://www.cs.toronto.edu/~websql/thesis.ps>.

Cité page 26

[Miller et al. 1990] G.A. MILLER AND R. BECKWITH AND C. FELLBAUM AND D. GROSS AND K.J. MILLER. (1990). « Introduction to WordNet : an on-line lexical database ». *International Journal of Lexicography*, 3(4) :235–244.

@ : <http://www.cogsci.princeton.edu/~wn/5papers.pdf>.

Cité pages 24,163

[Mitchell 1997a] TOM M. MITCHELL (1997a). « *Machine Learning* », chapter 13 : Reinforcement Learning. McGraw Hill.

Cité page 112

[Mitchell 1997b] TOM M. MITCHELL (1997b). « *Machine Learning* », chapter 6 : Bayesian Learning. McGraw Hill.

Cité page 112

[Najork et Wiener 2001] MARC NAJORK AND JANET L. WIENER (2001). « Breadth-first crawling yields high-quality pages ». Dans *World Wide Web*, pages 114–118. ACM Press.

@ : <http://citeseer.nj.nec.com/najork01breadthfirst.html>.

Cité page 108

[Nguyen 2003] BENJAMIN NGUYEN (26 décembre 2003). « *Construction and Maintenance of a Web Warehouse* ». PhD thesis, Université Paris-Sud, France.

@ : <http://www-rocq.inria.fr/~bnguyen/these.pdf>.

Cité page 23

[Papakonstantinou et al. 1995] YANNIS PAPAKONSTANTINOY AND HECTOR GARCIA-MOLINA AND JENNIFER WIDOM (1995). « Object Exchange across Heterogeneous Information Sources ». Dans P. S. Yu et A. L. P. Chen, editors, *11th Conference on Data Engineering*, pages 251–260, Taipei, Taiwan. IEEE Computer Society.

@ : <http://citeseer.ist.psu.edu/papakonstantinou95object.html>.

Cité page 36

- [Pittsburgh site] UNIVERSITY PITTSBURGH (website). « Site d'accueil du département informatique de l'université de Pittsburgh, Pittsburgh USA ».
@ : <http://www.cs.pitt.edu/>.
Cité page 113
- [Postel et Reynolds 1985] J. POSTEL AND J. REYNOLDS (October 1985). « RFC 959 : FILE TRANSFER PROTOCOL (FTP) ».
@ : <http://www.ietf.org/rfc/rfc959.txt>.
Cité page 97
- [Quass et al. 1995] DALLAN QUASS AND ANAND RAJARAMAN AND YEHOSHUA SAGIV AND JEFFREY D. ULLMAN AND JENNIFER WIDOM (1995). « Querying Semistructured Heterogeneous Information ». Dans *Deductive and Object-Oriented Databases*, pages 319–344.
@ : <http://citeseer.ist.psu.edu/quass95querying.html>.
Cité page 36
- [Raggett et al. 1999] DAVE RAGGETT AND ARNAUD LE HORS AND IAN JACOBS (24 December 1999). « HTML 4.01 Specification ».
@ : <http://www.w3.org/TR/REC-html40/>.
Cité page 45
- [Rennie et McCallum 1999] JASON RENNIE AND ANDREW KACHITES MCCALLUM (1999). « Using reinforcement learning to spider the web efficiently ». Dans *Proc. 16th International Conf. on Machine Learning*, pages 335–343. Morgan Kaufmann, San Francisco, CA.
@ : <http://citeseer.nj.nec.com/rennie99using.html>.
Cité pages 112,115
- [Roche 2003] MATHIEU ROCHE (2003). « Extraction paramétrée de la terminologie du domaine ». *Revue RSTI série RIA-ECA, numéro spécial EGC03*, 17 :295–306.
Cité page 42
- [Saïs 2004] FATIHA SAÏS (2004). « Transformation d'informations structurées en documents XML guidée par une ontologie ». Mémoire de stage de DEA.
Cité page 12
- [Saïs et al. 2005] FATIHA SAÏS AND HÉLÈNE GAGLIARDI AND OLLIVIER HAEMMERLÉ AND NATHALIE PERNELLE (2005). « Enrichissement sémantique de documents XML représentant des tableaux ». Dans *Extraction et Gestion de Connaissances (EGC)*, pages 407–418. RNTI-E-3.
Cité page 12
- [Sédraati-Dinet 2003] GÉRALD SÉDRATI-DINET (2003). « E.dot - Acquisition des données du Web ».
@ : <http://www-rocq.inria.fr/verso/edot/Reu-Plen-4-Avr-03/Slides-Xyleme/>.
Cité pages 79,108
- [Socolofsky et Kale 1991] T. SOCOLOFSKY AND C. KALE (January 1991). « RFC 1180 : A TCP/IP Tutorial ».
@ : <http://www.ietf.org/rfc/rfc1180.txt>.
Cité page 94
- [Spertus 1998] ELLEN SPERTUS (February, 1998). « *ParaSite : Mining Structural Information on the Web* ». PhD thesis, Department of EECS, MIT, Cambridge, MA, USA.

@ : http://www.mills.edu/ACAD_INFO/MCS/SPERTUS/Thesis/thesis.pdf.

Cité page 29

[Spertus et Stein 2000b] ELLEN SPERTUS AND LYNN ANDREA STEIN (2000b). « Squeal : a structured query language for the Web ». Dans *Proceedings of the ninth international conference on World Wide Web, Amsterdam, Netherlands*.

@ : <http://www9.org/w9cdrom/222/222.html>.

Cité page 29

[Spertus et Stein 2000a] ELLEN SPERTUS AND LYNN ANDREA STEIN (June, 2000a). « Squeal : a structured query language for the Web ». *Computer Networks*, 33(1-6) :95-103.

@ : <http://www9.org/w9cdrom/222/222.html> ; <http://www.elsevier.nl/gej-ng/10/15/22/48/25/33/abstract.html> ; <http://www.elsevier.nl/gej-ng/10/15/22/48/25/33/article.pdf>.

Cité page 29

[Sutton et Barto 1998] RICHARD S. SUTTON AND ANDREW G. BARTO (1998). « *Reinforcement Learning An Introduction* ». MIT Press.

Cité page 112

[SymPrevious site] SYMPREVIUS (website). « Site du Projet Sym'Previous ».

@ : <http://www.symprevious.net>.

Cité pages 9,42

[Van-Risbergen 1979] C.J. VAN-RISBERGEN (1979). « *Information Retrieval* ». 2nd edition, London, Butterworths.

Cité page 42

[Voorhees 1998] VOORHEES, EM. (1998). « *WordNet : An Electronic Lexical Database and some of its Applications* », chapter 12 : Using WordNet for Text Retrieval. MIT Press, Christiane FellBaum editor.

Cité pages 24,163

[Wu et Palmer 1994] ZHIBIAO WU AND MARTHA PALMER (1994). « Verb Semantics And Lexical Selection ». Dans *Proceedings of the 32nd Annual Meetings of the Associations for Computational Linguistics*, pages 133-138.

@ : <http://www.ai.mit.edu/people/jimmylin/papers/WuZ94.pdf>.

Cité page 24

[Xyleme site] XYLEME (website). « L'entreprise Xyleme ».

@ : <http://www.xyleme.com/>.

Cité page 108

[Yahoo! site] YAHOO! (website). « Le portail internet Yahoo! ».

@ : <http://www.yahoo.com>.

Cité pages 18,26,99

Index

- Exploration du web
 - Focused crawling
 - Approche de Jason Rennie et al., 112
 - Approche de Michelangelo Diligenti et al., 115
 - general purpose crawling, 103
 - meilleur d'abord
 - Fish-Search, 109
 - Shark-Search, 110
 - Stratégie le meilleur d'abord, 109
 - Systématique, exhaustive, 103
 - aléatoire, 106
 - en largeur d'abord, 103
 - en profondeur d'abord, 105
 - popularité d'abord, 107
 - Sélective, 111
 - Avec apprentissage initial, 112
 - Focused crawling, 112
 - Guidée par une requête WeQueL, 133, 135
 - InfoSpiders, 119
 - Intelligente, 121
 - sans apprentissage initial, 119
- Exploration sélective
 - Guidée par une requête WeQueL, 135
 - L'estimateur *WEst*, 137
- Exploration sélective guidée par une requête WeQueL
 - L'estimateur *WEst*, 138
 - discrétisation des scores, 138
 - La matrice statistique de fratrie, 140
 - La matrice statistique de parenté, 140
 - Modèle de probabilité, 138
 - Score d'un candidat, 140
 - Stratégie d'exploration, 141
- Google, 20, 21
 - Advanced Google, 22
 - avancé, 22
 - Hit List, 20
 - IR score, 21
 - Lien entrant, 20
 - PageRank, 21
- Langages
 - avec ontologie, 23
 - THESUS, 23
 - de navigation, 24
 - SQUEAL, 29
 - W3QL, 28
 - WebSQL, 26
 - de restructuration de sites web, 30
 - PENELOPE & ULIXES, 34
 - StruQL, 36
 - WebOQL, 32
 - sur le contenu, 18
 - Altavista, 21
 - Google, 21
 - Lycos, 18
 - mots clés avancés, 20
 - mots clés classiques, 18
 - Yahoo, 18
- Mesures de popularités
 - Authority, 21
 - Hub, 21
 - PageRank, 21
- Modèles de données
 - PENELOPE & ULIXES
 - ADM, Araneus Data Model, 35
 - SQUEAL
 - Modèle relationnel, 29
- Langages mots clés
 - Vecteur de mots, 18
- StruQL
 - OEM, 36

- THESUS
 - Liens entrants, 23
- WebOQL
 - AST, Abstract syntax tree, 32
- WebSQL
 - Modèle relationnel, 26
- WeQueL
 - WDM, 56
- Requêtes
 - de navigation, 27
 - de navigation, de connexion, 24
 - Expression régulière de chemin, 25
 - Regular path expression, 25
 - mots clés, 18
- Robot explorateur
 - Agent de rapatriement, 97
 - Document candidat, 96
 - Graphe
 - Exploré, 98
 - Module de stockage, 98
 - Pile d'urls, 95
 - Stratégie d'exploration, 97
- Smart crawling
 - L'estimateur *WEst*
 - Confirmation d'une estimation, 142
 - Exemple de fonctionnement, 143
 - Infirmation d'une estimation, 142
 - Intérêt d'un candidat, 144
 - La matrice statistique d'ancêtres, 144, 145
 - La matrice statistique de fratrie, 144, 145
- Web, 91
 - Exploration, 93
 - Condition d'arrêt, 100
 - Cycle, 94
 - Initialisation, 98
 - Systématique, 103
 - Graphe, 93
 - Lien hypertexte, 91
 - entrant, 92
 - sortant, 92
 - Profondeur, 98
 - Robot explorateur, crawler, spider, 94
 - architecture, 94
 - Itération, 99
 - Url, 91
 - Web invisible, 92
- WeQueL, 37, 43
 - Combinaison de requêtes atomiques, 47
 - partie modérée, partie pondérée, 48
 - partie nécessaire, 48
 - Constructeur sémantique, 44, 45
 - logique, 45
 - numérique, 46
 - Constructeur sémantique logique
 - AtMost*(*k*), 46, 59
 - Atleast*(*k*), 46, 59
 - Conjunct*(), 46
 - Disjunct*(), 45, 59
 - Exactly*(*k*), 46, 59
 - Constructeur sémantique numérique
 - Conjunct*(), 59
 - Cosine*(), 46
 - Fscore*(), 46, 59
 - Precision*(), 46, 59
 - Recall*(), 46, 59
 - fonction d'évaluation $\mathcal{F}_Q$, 50
 - average*(), 50
 - balanced_sum*(), 51
 - max*(), 50
 - min*(), 50
 - paramètres optionnels, 51
 - Modèle de données
 - WDM, 56
 - requête atomique, 43
 - requête complexe, 50
 - attribut d'évaluation, 50
 - fonction d'évaluation $\mathcal{F}_Q$, 50
 - score d'évaluation
 - combinaison de requêtes atomiques, 64
 - requête complexe, 66
 - Section, 44
 - MIME*, 45
 - incoming_links*, 44
 - keywords*, 44
 - object[type = 'T']*, 45
 - outgoing_links*, 44
 - page_title*, 44
 - title_incoming_page*, 44
 - url*, 44
 - url_length*, 44

- syntaxe abstraite
 - conversion vers syntaxe opérationnelle, 54
- syntaxe opérationnelle, 52
 - schéma XML de définition, 52