Focused Search on the Web using WeQueL

Amar-Djalil MEZAOUR Laboratoire de Recherche en Informatique (LRI), France Email: mezaour@lri.fr

Abstract

Keyword-based web query languages suffer from a lack of precision when searching for a precise kind of documents. Indeed, some documents cannot be simply characterized by a list of keywords. For example, searching for on-line pictures dealing with formula one using only simple keywords with general-purpose search-engines gives imprecise answers. This imprecision is due to the method that considers that a relevant document to a query is one that contains a lot of query keywords occurrences. This method is totally unefficient for poor textual-content documents like pictures, video streams.... On the other hand, keyword based languages are often not powerful enough for expressing sophisticated document search like on-line java or c++ tutorials.

We propose "WeQueL " a multi-criteria query langage for a better characterization of documents. The aim is to increase the precision of document retrieval on the web. In our experiments, we show the gain in accuracy for web document searching using our language.

1 Introduction

Nowadays, the web represents an important heterogeneous data source (news, articles, pictures, streams...). The information is stored in entities called documents. These documents are identified in a unique way by their urls and are linked together by hyperlinks. Searching for an information in the web consists of finding the urls of documents containing this information. General-purpose search engines have been developed to offer simple and powerful tools for users in order to search web documents. A general-purpose search engine, like Google [7], can be divided into three general components : a web crawler, a local web-index repository and a user query language. The web crawler is a program that visits the most possible web documents in the web in order to download them to the local web-index repository. In this local repository, the crawled documents are indexed and stored. A keyword based query language is proposed to the final users for quering the index repository. A user has to specify a query in which he declares a list of relevant keywords characterising, according to him, the documents to search for. This keyword list is then submitted to the repository to retrieve the urls of the documents containing occurences of the keywords in this list. The given answers are often numerous, not enough precise and do not necessarily match the user's actual needs. There are two major reasons for this. The first reason is that the expressive power of a keyword-based query does not define the exact wanted pages. Indeed, a keyword based

query specifies only the keywords that a document must contain in its content without any possibility of describing other caracteristics and properties of a web document. A keyword based search is totally inefficient when searching for poor textual-content documents (pictures, pdf...) or when searching for documents which are difficult to characterise only with keywords (on-line c++ or java courses). The second reason of the imprecise answers to a query is in the query evaluation method. In fact, most keyword based languages consider that the presence of all the given keywords in a document is enough to consider it as relevant to the given query. This evaluation method does not take into account the precise location of the keywords in the document and ignores important properties of a web document, like its context, structure and type. For example, when searching for c++ courses in the web, a user submits the query " c++ courses " to a search engine without any other possibility to describe in a more elaborate way those c++ courses documents. The search engine gives a lot of answers. Some of these answers are relevant c++ courses but the others are irrelevant and may contain schedule and timetable of c++ courses.

In this paper, we show that a focused keyword based search which combines the structure and properties of a document can be an interesting way of increasing the precision of web search document. We propose a Web Query Language : "WeQueL" that combines different criteria concerning a document using logical operators. For example, a query of "WeQuel" allows the user to combine keywords criteria concerning the title, the type, the content of a document with other document properties like its url, incoming links.... In the next section, we present a state of the art of related work on web document search. We define then in the third section the syntax and the semantics of the queries of our language. Our experiments are detailed in the fourth section of the paper. Finally, we conclude with some on-going work.

2 State of the art

Web search documents is based on two complementary components : a local collection of documents (what we called local web-index repository) and a query language over this collection. The repository is populated from the web by programs called spiders or crawlers. These programs crawls the web in order to discover new web documents and to download them in the repository. A crawler crawls the web starting from given urls by following recursively all or some links of the current crawled document. The crawling process is done according to a defined strategy (random, breadth-first, deep-first, intelligent...). Once the repository is populated by the crawler, the user can query this collection for specific documents.

Search engines are based on the two components seen above. Most of popular search engines offer keyword based languages to users to query their web index repository. For a submited query q, a search engine collects the urls of the documents containing the most occurences of all the keywords of q (without considering their location). The answers are then ranked according to their relevance to the query (PageRanking [2], source and authority [8]). Unfortunately, even the best ranked answers are not necessarily relevant. This can be explained by a lack of characterization strength of a keyword based language which ignores

essential parts of a web document like its structure, type, context. Having bad precision results, keyword based languages can not be used for demanding applications like topic-specific search engines or thematic data warehouses. In addition to its standard keyword search, Google offers an advanced search which allows the user to target his query to specific parts of a document (like the title, the body and the links). With this advanced option, the expressiveness strength of an advanced google query is increased in the way that users can specify where their query keywords should be located. This gives a additional possibility to express more precise queries than just keywords. Unfortunately, it is not possible to combine different targeted queries with advanced google search. For example, one can not target the url and the title in the same advanced google query.

To increase the answers precision, other approaches consists of improving the repository quality over which the queries are evaluated. Junghoo Cho & al.[3] show that crawling the documents having important pagerank first yields to a high repository quality. This means that evaluating a query over this repository gives high quality answers which are most likely to correspond to the desired answers. Focused crawling is another technique for improving the repository quality. Focused crawling consists of limiting the crawled pages to download only the pages dealing with a fixed thematic. This is done by reducing the number of followed links. To do this, a best-first crawling strategy is applied in order to follow the "promising" links first. This selective strategy is based on a function which can estimate the benefit of following a link only from the words occuring in its neighborhood. This function is learned from a sample web graph by using machine-learning techniques. Note that this learning graph should be representative of the fixed thematic. For example, Jason Rennies & A.K.McCallum have build CORA [10], a topic-specific search engine specialised in on-line research papers, using focused crawling. In their selective crawling process, their system uses a discriminant function built by combining reinforcement learning and bayesien classifiers [9]. An other similar approach is the method that have used *Diligenti & al.* for developing CiteSeer [4] (the most popular on-line scientific papers). Their approach consists of training a bayesian classifier over a labelled graph to estimate the distance separating a given document from a possible on-topic documents [5]. Having this estimation, their best-first crawling strategy consists of following at each step the outgoing links of the nearest document to a relevant one. A focused crawling solution can be a good alternative for the web search quality problem but it seems to be a little bit difficult to implement. Indeed, the machine learning techniques that are necessary to learn the discriminative function needs a learning sample graph which must be representative enough of the fixed topic to reach satisfying performances. Unfortunately, this learning graph is not easy to build and needs a cumbersome manual labelling of each of its nodes.

The intelligent crawling is a technique proposed by *Charu C.Aggarwal & al* [1]. Intelligent crawling needs no training on a learning graph. In fact, an intelligent crawler learns, one by one, during its crawling how to favour the promising links. This progressive learning uses a function to estimate the relevance of each crawled document while it is crawled. This function can be for example the presence of a list of keywords in a document. During the crawling process, a set of statistical measures associated with the candidate pages is updated

¹a candiate page is a page not yet crawled but is pointed to by an already crawled page

each time a document is visited and evaluated by the function. These statistical measures expresses different properties of how the relevant documents of the visited graph are linked to the candidate pages. The intelligent crawling strategy consists then of downloading first the candidate page that reaches the highest statistical score (a combination of the measures of the statistical set). The intelligent crawling approach reduces significantly the costs of the training step since it does'nt need to do so. Unfortunately, the efficiency of this approach is strongly related to the precision of the discriminative function.

To summarize, the ability of distinguishing without any ambiguity and with a very high precision a relevant document from a non relevant document is the key of focused web search document. We then propose a web query language : "WeQueL " which allows a better characterisation of the desired documents.

3 WeQueL

We have defined a web query language that we called "WeQueL ". WeQueL is an attributevalue language that allows a user to combine different criteria using logical operators to define the documents interests. Each criterion specified in a WeQueL query allows a user to focus on a special part of a document (title, link neighborhood . . .) or to describe a particular propriety of a document (url tokens, mime type . . .). The criteria of WeQueL can be combined using the logical operators of conjunction and disjunction to focus for example in the same query the title, the incoming links with the type and url of a document. Each WeQueL query is built from the logical combination of atomic queries. An atomic query is a keyword based query having this form : " attribute = values ". We fixed a set of possible attributes for the WeQueL queries. The different atomic queries are the following :

• $page_title = val_1, \ldots, val_n$:

where each val_i is a string that may contain more than one word separated with white space character. This query is evaluated to true over a page p if and only if all the words appearing in one of the above values val_i are present in the title of p in the same order that they have been declared in their corresponding value val_i . The page title, when it exists, is defined with the content of the tags : $\langle title \rangle$ or $\langle h1 \rangle$ or $\langle meta$ name="title"> \rangle .

page title	q_a evaluation
cours de licence en java	true
java cours introductif	false

Example : Consider q_a : "page_title = cours java, introduction à java"

• $mime = type_1, \ldots, type_n$:

Mime types $type_i$ must be conform to the standard mime types defined for the html language. This query is evaluated to true over a page p if p mime type corresponds to one of the specified mime types in the list $type_i$.

• $title_incoming_page = val_1, \ldots, val_n$:

This query is evaluated to true over a page p if and only if it exists a page p' pointing to p and p' is evaluated to true with the atomic query $page_title = val_1, \ldots, val_n$.

• $url = val_1, \ldots, val_n$:

This query is evaluated to true over a page p if and only if all the words of one of the above val_i appears (in the same order with their order in val_i) in the url tokens of p.

Example : Consider q_a : *url* = *univ* cours java

page url	q_a evaluation
http://www.infres.enst.fr/~charon/coursJava/	false
http://www.univ-reunion.fr/~courdier/cours/java	true

• $incoming_links = val_1, \ldots, val_n$:

- This query allows the user to target the words appearing in the neighborhood of the incoming links of a page. We defined the neighborhood of a link as the words appearing in : its anchor, the tokens of the urls that this link refers to and the 10 words before and after the link (before the $\langle a \text{ href} \rangle$ tag and after the closing tag $\langle /a \rangle$). This query is evaluated to true over a page p if and only if it exists one link l that points to p and having all the words of at least one of the val_i appearing in the neighborhood of l (with the same order in this val_i).
- $outgoing_links = val_1, \ldots, val_n$:

This query allows the user to focus on the words appearing in the neighborhood of the outgoing links from a page. This query is evaluated in the same way as the previous atomic query with the only difference that it concerns the links of the page p.

• keywords = val_1, \ldots, val_n :

The keywords of a page are extracted from the content of the tag <meta name = "keywords"> when this last one exists in a document.

• url_length = number_restriction :

where $number_restriction$ is a cardinality restriction (atleast[n], atmost[n]). This query is useful when one has to limit the length of the desired documents. We defined the length of a url as the number of '/' that it contains.

Example : *url_length = atmost[2]* was useful for us to characterise the notion of being a homepage.

page url	q_a evaluation
http://www.gofast.com	true
http://www.yahoo.fr/tourisme	false

In the upcoming sections, we note $\mathcal{G}(\mathcal{P}, \mathcal{L})$ the graph of web pages that will be used for evaluating the queries of WeQueL. \mathcal{P} is the set of the nodes of the graph (web pages) and \mathcal{L} the set of links between the nodes of \mathcal{P} . The associated semantics, noted \mathcal{S}_{q_a} , of an atomic query q_a is defined as the set of the pages from \mathcal{P} that satisfy q_a . These answer pages are obtained after evaluating q_a over \mathcal{P} and \mathcal{L} : $S_{q_a} = \{url(p) \mid (p \in \mathcal{P}) \land (q_a \text{ is evaluated to } true \text{ over } p)\}$

We implemented a java program for evaluating WeQueL queries over \mathcal{P} and \mathcal{L} . This program generates first a list of regular expressions from the values of each atomic query of a WeQueL query according to java/Perl/unix regular expression syntax. The regular expression are obtained by replacing all the blank charaters by the string ".*". This replacement guaranties that the words of the different values of an atomic query are searched in the order that they are specified. The program matches then these expressions on the content of the corresponding target part of a document for each page $p \in \mathcal{P}$.

A conjunctive WeQueL query is a conjunction of atomic queries without repetition of attributes in the same conjunction and with exactly one attribute " mime type " in the conjunction.

Example : This conjunctive query expresses the needs of a search for java courses documents in french only in pdf or postscript format :

(mime = application/pdf, application/postscript) \land (url = univ java, cours java) \land (incoming Links = cours java, introduction à java)

The semantic of a conjunction of atomic queries $q = q_a^1 \wedge \ldots \wedge q_a^n$ is defined as :

$$\mathcal{S}_q \;=\; igcap_{i=1}^n \; \mathcal{S}_{q_a^i}$$

A disjunctive WeQueL query is defined as a disjunction of WeQueL conjunctions. The semantics of a WeQueL disjunction $Q = q_1 \vee \ldots \vee q_n$ is defined as :

$$\mathcal{S}_Q \;=\; igcup_{i=1}^n \; \mathcal{S}_{q_i}$$

Example : The following query expresses the search for java courses documents in french and in ps, pdf or html format :

(mime = application/pdf, application/postscript) \lapha (incoming links = cours java, introduction java) \lapha (url = univ java, cours java, enseignement java)

(mime = text/html) \land (url = univ cours java) \land (page_title = cours java, introduction java) \land (incoming_links = cours java, introduction java) \land (outgoing_links = sommaire, intro)

4 Experimental evaluation of WeQueL

We implemented a WeQueL query evaluator in java. We defined an experimental protocol for evaluating WeQueL queries. In this protocol, we specified the sample of tested queries, the sample of web documents over which the tested queries have been evaluated and finally the two measures to estimate the WeQueL quality.

4.1 Sample of tested queries

We fixed three topics for testing WeQueL : computer science courses documents (in french), homepages of touristic documents (in french also) and finally formula one pictures. We manually defined 13 different WeQueL queries expressing different kind of web document search : 4 queries for the computing-science courses documents, 5 queries for tourismo and 4 queries for the formula 1 pictures. These queries have been written in two steps. In the first step, we wrote one query for each topic. We obtained 3 queries that we call " initial queries ". We consider that we are " advanced users " in WeQueL with good knowledge about the three fixed topics. Writing an initial query consists of targeting the relevant parts of the desired documents by choosing the right logical combination of atomic queries with their corresponding relevant values. The initial Queries shows the power of expressiveness of a WeQueL query.

The other queries have been generated by relaxing a given number k of atomic queries from all the conjunctions of the initial queries of each topic. These relaxations are called "relaxation k" of the initial query. Let's consider the conjunction : $q_1 \wedge \ldots \wedge q_n$ of a WeQueL initial query Q. To relax k atomic queries from this conjunction where n > k, one has to remplace this conjunction by the disjunction of the C_n^k possible conjunctions : $q_{i_1} \wedge \ldots \wedge q_{i_{n-k}}$. For a semantic coherence, the mime-type atomic query can not be relaxed and must be combined (by conjunction) with at least another different atomic query. This means that it not possible to have or to relax until conjunctions of only one atomic query. Note also that no relaxation is possible for $k \geq n-2$.

A relaxed query of an initial query Q is less restrictive (and so less precise) than Q. We can also note that a relaxation k of an initial query Q is more restrictive (and so more precise) than a relaxation k + 1 of it.

Example : Let's consider the initial query Q composed by two conjunctions :

$$\mathcal{Q} = q_1 q_2 q_3 \vee q_1^{\prime} q_2^{\prime}$$

where q_1 and $q_1^{'}$ are mime-type atomic queries. The relaxation 1 of $\mathcal Q$ is :

$$q_1q_2 \lor q_1q_3 \lor q_1^{'}q_2^{'}$$

We can see that it is not possible to relax the second conjunction $\dot{q_1} q_2$ nor to write a relaxation k for $k \ge 2$.

The relaxation of the initial queries of each of the three topics have been done to show the benefit of the conjunctions of the initial queries. The experiment results are shown in the tables 3, 4 et 5. tourism homepages

 $(URL = voyage, tourisme, sejour, reservation billet...) \land (Outgoing Links = reservez, vol, sejour...) \land (page_title = agence tourisme, tour operateur, office tourisme, compagnie aerienne, vol charter...) \land (Incoming_Links = agence tourisme, tour operateur, achat reservation billets, vol regulier...) \land (MIME = text/html) \land (keywords = NOT_NULL) \land (url length = atmost[2])$

french computer science courses

 $(url = cours cpp, cours c, cours slide, cours transparent...) \land (page title = cours informatique...) \land (Incoming Links = cours algo, cours info, notes cours...) \land (Outgoing Links = introduction...) \land (MIME = text/html)$

(Incoming Links = notes cours , cours reseaux...) \land (MIME = application/postscript , application/pdf...) \land (URL = cours ia , cours bd , univ \sim cours , info cours ...)

(title_Incoming_Page = cours informatique, documentation cours, notes de cours, cours algo ...) \land (MIME = application/ppt, application/msword application/pdf ...) \land (URL = cours 'bd', \sim sld ...)

Formula one pictures

 $(page_title = photo gallery `f1', schumacher picture, montoya picture ...) \land (URL = photo `f1', image `f1', coulthard picture, villeneuve picture ...) \land (Incoming Links = photo formule 1, image formule 1, grand prix photo ...) \land (Outgoing Links = suivant, precedent, previous, back ...) \land (MIME = text/htm)$

 $(URL = f1 photo, image f1, grand prix photo, gallerie f1...) \land (Incoming Links = formula one pic, f1 picture, formula one gallery...) \land (MIME = image)$

 $(URL = photo ferrari f1, photo mac laren f1, ralf picture ...) \land (title Incoming Page = formula one gallery, photo ferrari 'f1' ...) \land (MIME = image, text/htm)$

Table 1: The three initial queries

4.2 Evaluated web pages

We have been confronted with the dilemma of choosing the right graph to build for our queries evaluation. For limitations reasons (capacity and time), we could'nt evaluate our queries over all the web documents or over a repository equivalent to Google index. We have then explored other costless alternatives which best-fit our machines capacity. We found two interesting possibilities : build a random web graph by sampling the web or build three thematic graphs corresponding to the three topics. The first possibility is incontestably the most obvious to implement. Unfortunately, we have no guarantee of having the desired thematic documents which are necessary for a good evaluation of our queries. We have then chosen to build three thematic graphs. For each thematic graph, we used Google to obtain a set of urls related to the topic of the graph. All the documents corresponding to these urls are downloaded and stored in a relational database (MySql). These stored documents build up the level zero of the thematic graph. We implemented in java a web crawler in order to extend the level zero of each graph. Due to storage limitation and crawling time limitation, only the two best ranked documents of the level zero (from Google responses) are extended. The extension is done by downloading and storing in the local database the documents pointed to by the documents of the level zero. This extension process is repeated until reaching a fixed depth (p = 10 for our experiments). We finally obtain 3 thematic graphs. The level "zero" of each graph contains Google responses to some atomic queries from the initial query. The level "1" contains the pages pointed to by the two best ranked google responses from the level "0". Finally, levels "i, 1 < i < p" contain pages that are pointed to by pages from level i - 1. Due to the same limitations as above, only b% random choosen links from each page of level i-1 are followed $(b = \frac{1}{i+1}).$

The thematic graphs that have been built contains sufficient number of relevant documents to validate the evaluation measure quality of WeQueL (see section 4.3). The choices that have been made for building the thematic graphs are justified by two hypothesis. The first one consists in the reliability of the best ranked google responses [2]. The second hypothesis is based on the fact that the web is thematically connex [8, 2, 5]. In fact, a relevant document points often to other relevant documents. Following links of relevant documents increases the chance to include other relevant documents.

Let's see now how the level zero of each graph is obtained. Let's consider Q an initial query. The zero level is obtained by submitting a set of queries to Google. These queries correspond to the atomic queries that Google can evaluate. The WeQueL atomic queries that can be evaluated by Google are : title, incoming and outgoing links, url. This is an example of the sent Google queries having an initial query $Q : Q = (page_title = cours java, cours c++) \lor (url = univ java, cours cpp)$. The advanced Google queries are : (allintitle: cours java) OR (allintitle: cours c++) OR (allinurl: univ java) OR (allinurl: cours cpp).

The graph characteristics of each topic is given in this table :

WeQueL query	# Google responses	# pages	# followed links
computer science courses	2.404	271.850	4.347.930
tourism documents	2.027	239.817	5.916.248
formula one pictures	1.599	149.634	8.990.615

Table 2: Number of nodes and links in each thematic graph

4.3 Quality measures

We retained two measures for estimating WeQueL quality : the precision and the recall. The precision of a WeQueL query Q is the proportion of really relevant Q answers among all the responses of Q (pages evaluated to true). The recall of a WeQueL query Q is the proportion of really relevant Q answers among the relevant documents of the nodes set P of the thematic graph.

Let's consider \mathcal{P} the set of nodes (documents) of a thematic graph $\mathcal{G}(\mathcal{P}, \mathcal{L})$ and \mathcal{S} the set of answers of the evaluation of \mathcal{Q} over \mathcal{P} . The relevance modality divide each of the previous sets into two subsets : relevant nodes (\mathcal{P}^r and \mathcal{S}^r) and non-relevant nodes (\mathcal{P}^{nr} and \mathcal{S}^{nr}) :

$$\mathcal{P} = \mathcal{P}^r \cup \mathcal{P}^{nr} \qquad \mathcal{S} = \mathcal{S}^r \cup \mathcal{S}^{nr}$$

The precision and the recall are then defined as :

precision =
$$\frac{|\mathcal{S}^r|}{|\mathcal{S}|}$$
 recall = $\frac{|\mathcal{S}^r|}{|\mathcal{P}^r|}$

In our experiments, the number of graph nodes (see Tab.2) and the number of query answers are very important and can not be manually labelled. We used then random sample sets of 100 elements to estimate precision and recall (manual labelling of $|S^r|$ and $|\mathcal{P}^r|$)

We also want to know how precise our language is comparing to Google. Unfortunately, it is not possible to express the same WeQueL queries in Google (neither classic Google nor advanced Google offers the possibility to do logical combinations). We defined then for each initial query a set of comparable Google queries :

- google classical query is made of all the values in the initial query ;
- different advanced google queries (title, urls, incoming and outgoing links) made of the values of their corresponding occurrence in the initial query.

With these comparisons, we first show the benefit of a targeted query (google advanced queries comparing to classical google) and show also the gain of precision of a logical combination (WeQueL query comparing to advanced google).

The results are shown in the tables Tab.3, Tab.4 et Tab.5. In these tables, we show in the line " advanced Google " the best result of the advanced google queries evaluations and its corresponding google advanced query.

5 Experimental results

In our manual labelling, we have been very restrictive : a document which contains no relevant data but points to relevant documents is considered as not relevant. The evaluation results are:

computer science courses query	precision	recall
Classical Google	9,00%	6,68%
Advanced Google (best = url)	26,00%	5,82%
initial query	71,00%	11,16%
relaxation 1	56,00%	56,57%
relaxation 2	65,00%	63,31%
relaxation 3	10,00%	100%

tourism query	precision	recall
Classical Google	10,00%	7,29%
Advanced Google (best = title)	21,00%	4,14%
initial query	100,00%	2,30%
relaxation 1	29,29%	10,56%
relaxation 2	8,00%	21,74%
relaxation 3	7,00%	61,56%
relaxation 4	2,00%	100%

Table 3: computer science courses

Table 4: tourism documents

formula one pictures	precision	recall
Classical Google	9,00%	22,99%
Advanced Google (best = url)	32,00%	9,47%
initial query	65,53%	9,44%
relaxation 1	41,00%	36,11%
relaxation 2	36,00%	35,99%
relaxation 3	8,00%	93,01%

Table 5: formula one pictures

We can see from the two first lines from the above tables that a targeted query is more precise than a non targeted query. We can also see that on our initial queries WeQueL is three times more precise than Google. This shows how expressive and powerful are logical combinations of different targeted queries. Unfortunately, recall results of a WeQueL queries are a little bit low. This can be explained in two ways : the first way is that we have been very retrictive in our manual labelling. The second way is in the evaluation method which matches only the defined values in the WeQueL queries without exploiting synonyms for example or machine learning statistics. One way for improving the recall results consists of expanding the query keywords values with synonyms from ontologies for example.

6 Conclusion

In this paper, we have presented a multi-criteria web query language which allows the user to target his keywords on different parts of a document in the same query. We used different queries to illustrate the expressiveness of WeQueL language. We have experimentally shown that logical combinations of targeted queries are more precise than simple keywords based queries. WeQueL is in use in the "eDot" project [6] as a filtering tool for populating thematic data warehouses. "eDot" project consists of developing a data warehouse populated from the web with documents dealing with food risk. For the warehouse needs, we semiautomatically defined a characteristic WeQueL query for describing the documents to include in the eDot warehouse. This WeQueL query is based on given relevant examples and a risk food ontology. In this project, we are studying how our language can be a good filtering tool of urls given by a general purpose crawler (Xyleme crawler [11]). Having the precision and recall filtering results, we can use WeQueL for an intelligent crawling process to increase the number of relevant documents to include in the warehouse. WeQueL queries will then be used as discriminative functions.

References

- [1] Charu C. Aggarwal, Fatima Al-Garawi, and Philip S. Yu. Intelligent crawling on the world wide web with arbitrary predicates. In *World Wide Web*, pages 96–105, 2001.
- [2] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, 30(1–7):107–117, 1998.
- [3] Junghoo Cho, Hector García-Molina, and Lawrence Page. Efficient crawling through URL ordering. *Computer Networks and ISDN Systems*, 30(1–7):161–172, 1998.
- [4] CiteSeer. http://citeseer.nj.nec.com/cs, 2003.
- [5] Michelangelo Diligenti, Frans Coetzee, Steve Lawrence, C. Lee Giles, and Marco Gori. Focused crawling using context graphs. In 26th International Conference on Very Large Databases, VLDB 2000, pages 527–534, Cairo, Egypt, 10–14 September 2000.
- [6] eDot. http://www-rocq.inria.fr/verso/gemo/projects/edot.pdf, 2002.
- [7] Google. http://www.google.com, 2003.
- [8] Jon M. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46(5):604–632, 1999.
- [9] Jason Rennie and Andrew Kachites McCallum. Using reinforcement learning to spider the web efficiently. In *Proc. 16th International Conf. on Machine Learning*, pages 335– 343. Morgan Kaufmann, San Francisco, CA, 1999.
- [10] whizbang. Cora version 2.0 : Computer science research paper search engine, 2001. http://cora.whizbang.com.
- [11] Xyleme. http://www.xyleme.com/, 2002.