Université de Montréal

A Domain-specific search engine for the construction sector

par
Qi Zhang

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Mémoire présenté à la Faculté des études supérieures
en vue de l'obtention du grade de Maîtise ès sciences (M.Sc.)
en Informatique

Avril, 2003

Université de Montréal

Faculté des études supérieures

Ce mémoire intitulé:

A Domain-specific search engine for the construction sector

présenté par:

Qi Zhang

a été évalué par un jury composé des personnes suivantes:

El Mostapha Aboulhamid (président-rapporteur)

Jian-Yun Nie (directeur de recherche)

François Major (membre du jury)

Mémoire accepté le 17 juillet 2003

Résumé

Les engins de recherche présentement disponibles sur le Web offrent des outils de recherche d'information (RI) dans des domaines génériques. Aucun effort particulier a été mis sur la RI spécialisée. Des professionnels cherchant des informations spécialisées ne peuvent pas trouver ces informations de façon précise.

Le projet de recherche dans lequel s'inscrit notre étude vise à construire un système de question-réponse dans un domaine spécialisé qu'est la construction - le système ERIC (pour Engin de Recherche d'Information spécialisé en Construction). Différent de la RI traditionnelle, le but ultime de ce système est de fournir une réponse précise à une question de l'utilisateur. Le travail décrit dans ce mémoire constitue la première étape de ce système - la recherche de passages, ou l'identification des portions de document pertinentes pouvant contenir une réponse.

Dans notre travail, nous avons adopté un système de RI générique OKAPI comme notre outil de base d'indexation et de recherche. De plus, nous avons tenté d'améliorer la performance de recherche en:

1. reconnaissant des termes composés spécialisés comme terme d'indexation additionnels,

2. exploitant un thésaurus spécialisé pour identifier des termes reliés afin d'étendre la requête de l'utilisateur de façon à augmenter le rappel.

Nous avons testé notre approche sur une petite collection de test spécialisée en construction. La performance globale s'avère bonne. En particulier,

nous avons obtenu une amélioration d'environ 5% en intégrant des termes composés et l'expansion de requête en comparaison avec la méthode de base fournie par OKAPI. Ce résultat montre des bénéfices possibles de notre approche pour la RI spécialisée.

Le système que nous avons construit s'avère un cadre flexible pour l'intégration des outils additionnels pour l'analyse de la langue naturelle, une analyse nécessaire pour la question réponse.

Mots-clés : RI, Engin de recherche, ERIC, OKAPI, recherche de passage, terme composé, expansion de requête.

Abstract

The current search engines available on the Web perform general-purpose information retrieval (IR). No particular efforts have been put on domain–specific IR. Professionals looking for specialized information may not find the required information accurately. The research project described in this thesis aims to construct a domain-specific question answering system - ERIC (for Engin de Recherche d'Information specialisé en Construction). Different from traditional IR, the ultimate goal of this system is to provide a direct answer to a user's question. The work described in this thesis concerns the first step of this system, namely passage retrieval, or the identification of relevant portions of document that may contain an answer.

In our study, we adapted a general purpose IR system-OKAPI as the basic indexing and retrieval tool. In addition, we try to improve the retrieval effectiveness by

1. Recognizing specialized compound terms as additional indexes;
2. Exploiting a specialized thesaurus to identify related term in order to expand the user's query so as to increase recall.

We have tested our approach on a small test collection in the construction area. The global performance is quite good. In particular, when both compound term and query expansion are used, we obtain an improvement of about 5% in the retrieval performance, in comparison with the basic search method provided by OKAPI. This result shows the possible benefice of these approaches for domain-specific IR.

The system we constructed provided a reasonable framework to integrate additional tools for refined natural language analysis for question answering.

Keywords: IR, search engine, ERIC, OKAPI, passage retrieval, compound term, query expansion.

Table of Contents

List of Figures

List of Tables

Acknowledgments

First and foremost, it is my great pleasure to acknowledge the kind support of my supervisor, Dr. Nie who provided important attention and time with his friendly and valuable assistance throughout the period of my master degree study in University de Montreal. Moreover, I wish to thank the Dr. Colin Davidson and Gonzalo Lizarralde of the Faculté de l'aménagement. This thesis would not have been possible without their professional support. I have to thank my team members, Li-Fang Liu and Zhuo Zhang, with whom I spent a considerable amount of time finding solutions to the problems, which arose while pursuing this project. Finally, I would also like to thank BELL-LUB for supporting this project. The completion of this research was made possible thanks to Bell Canada's support through its Bell University Laboratories R & D program.

Chapter 1. Introduction

Due to the expansion of the Web, more and more pages become available on the Web. People often use the Web as a source of information when they have an information need. However, the huge size of the Web also makes it more and more difficult to distinguish relevant documents from irrelevant ones.

Search engines have been developed precisely to answer this requirement. Search engines are information retrieval (IR) system on the Web. They deal with the problems of information representation, storage and retrieval. They help users quickly identify a subset of documents that may contain relevant information. A number of search engines such as Google and Yahoo are currently available to public. However, those search engines can only perform general retrieval tasks. They do not take into account particularities of specialized areas. As a consequence, a user looking for domain-specific information will likely obtain a set of documents that contain the words in the general domain. To illustrate this, let us consider a specialist in construction who wants to find information about "highway shoulder" – a special concept in construction. For the existing search engines, the query "highway shoulder" is separated into two words: "highway" and "shoulder". With the word "shoulder", it is likely that the systems will retain a set of documents about human and animal shoulders instead of "highway shoulders". This example illustrates a typical problem of general search engines: the low precision rate in search for specialized information, and the search engine's inability to distinguish specialized documents from general

domain documents. This is one of the reasons why experts in construction do not often use the Web to find professional information.

To solve this problem, one means is to build specialized search engines or IR systems. Theses systems only index the documents in the required area, and they are able to recognize specialized terms in the given area. The construction of such a system is the goal of the present study.

This research project aims to provide the experts in construction with an efficient search tool. The ultimate goal is to provide a short yet precise answer to questions of the specialists in the construction sector. To achieve this goal, the first step is the identification of the documents or portions of documents that may contain an answer to the questions. Then in-depth linguistic analysis is carried out on these documents to further confirm if they contain an answer. Out study described in this thesis concerns the first step. As the questions are domain-specific, they cannot be dealt with in the sane way as general domain questions. In particular, the use of domain-specific knowledge will be important for at least two reasons: we need to recognize specialized terms, and we need to recognize the relationships (e.g. synonymy) between these terms. The use of domain-specific knowledge is the focus of this project.

In order to better understand the problems of general search engines for domain-specific questions, we will describe them in more detail in the next section.

1.1 Problems with the current IR systems for domain-specific search

For domain-specific IR, current search engines have several problems. The first problem is related to the use if single words as indexes. Currently, most IR systems use keywords to determine if a document should be retrieved: a document is retrieved if it contains the same keywords as the query. Keywords that are recognized automatically during the indexing process are in fact single words contained in a document. As we showed in an example earlier, a specialized compound term such as "highway shoulder" will be separated into several keywords, leading to less precise representation of the contents of the document. As a consequence, the documents retrieved may match the single words of a compound term without be related to the given compound term. This will introduce noise (i.e. non-relevant documents being retrieved) in the retrieval results.

For domain-specific IR, specialized terms and compound terms represent very important concepts that should be recognized accurately. Therefore, the recognition of specialized compound terms is an important problem in domain-specific IR.

The second problem is the domain-specific document problem. As more information becomes available on the Web, it is more difficult to distinguish specialized documents from general domain documents. The search engines just collect all information without distinguishing their domains. As a consequence, many retrieved documents are not related to the given specialization domain. In order to solve this problem, one has to consider constructing specialized document collection. This is one way to reduce retrieval noise and to provide IR in a specialized area.

The third problem concerns the use of domain in IR. Current search engines do not use any knowledge (e.g. Relationships between terms) during retrieval. However, in a specialized area, there is usually a domain knowledge base available. The simplest form of this knowledge base may be a thesaurus, which stores a set of specialized terms and the relationships between them. The use of such a knowledge base is important in specialized IR. For example, when a user inputs a query on "construction material", he would also like to find documents on "wood", "brick", "stone", "cement", and so on. Without using domain-specific knowledge, it is difficult to know that these latter concepts are strongly related to the one in the user's query. The related concepts, once identified, can be added into the query in a query expansion process.

1.2 Context and goal of the project

Our research project aims to implement a system for domain-specific information retrieval in the construction sector. It is also part of a research project at the Centre International Du Bâtiment (CIBÂT) and IF Research Group at University of Montreal, which is supported by an NSERC-CRD grant with BELL-LUB. This research group provides us with a thesaurus specialized in construction as well as their expertise. In particular, the test data we will use in our experiments are also prepared by them.

The final goal of the whole project is to construct a system capable of gathering information related to the construction sector, and answering user's question in this area. In order to answer user's question with a sentence of several sentences from a document, one first has to identify the relevant documents or portion of the

document that may contain an answer. Then a more refined analysis is carried out on these documents to extract and confirm the answer. Our work described in this thesis concerns the first part: identifying relevant portions of constructing the backbone IR system and document.

The system we constructed is called ERIC (Engin de Recherche d'Information specialisé en Construction). Our contributions in this study are as follows:

1. We adapted a general IR system – OKAPI – for ERIC.
2. By exploiting a specialized thesaurus, we recognize compound terms, and use them as additional indexes to single words.
3. We implemented a mechanism which uses the thesaurus to identify the related terms.
4. We implemented a simple semi-automatic downloading system tool to assist the web master collecting specialized documents on the web. The whole system is operational. This system is the basic search system that makes it possible to continue the research project on the aspects that are more related to question answering.

1.3 Organization of this thesis

The rest of the thesis is organized as follows. The second chapter reviews the IR concepts, algorithms, and main models. The third chapter presents the OKAPI system that we will use as a basic IR system. The fourth chapter describes our implementation of the ERIC system. The fifth chapter presents the experiments with ERIC. The sixth chapter presents some conclusions and future work in this project.

Chapter 2. Related work in IR

Domain-specific IR is a further improvement on classical IR. All the aspects in classical IR are thus related to our project. In order to give a complete review of the related work, we will start with the classical technique developed in IR, namely the indexing process and the retrieval models. Thus, we will describe some of the improvements that are closely related to our project:

— passage retrieval;

— the utilization of compound terms;

— question expansion;

— Web IR;

— question answering.

2.1 Document Indexing

The goal of indexing is two fold. 1) It identifies the most important concepts described in the document; and 2) it measures the importance of each concept in the document [Nie01]. Document indexing also creates a representation of document contents that is manipulable, and useful for the retrieval of document according to their contents. Since the original documents are usually unstructured texts, their contents cannot be manipulated directly. The classical indexing process then tries to determine simpler elements to represent them. Traditionally, keywords are used as the representation elements. They are usually single words. Keywords are often weighted in such a way that the weight of a keyword denotes importance of the keyword in the

document. There are three main steps in document indexing process: stop-words elimination, word stemming, and term weighting.

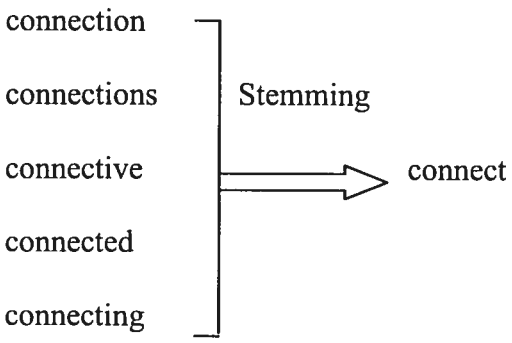
1. Stop-words Elimination

Not all words in a nature language are meaningful entities for representing the document contents. In each nature language there are words that only play a linguistic role without bearing a particular semantic meaning. For example, the preposition such as "in" or "of" has no meaning in document. It is useless and even harmful to keep these words as indexes because these words would introduce noise in the retrieval result. Therefore, when a word is encountered in a document, we first determine if the word is meaningful, and should be kept as an index. Usually a list of non-meaningful words is determined and used at this step. Such a list is called a stop-list. The words in the stop-list are called stop-words. Normally, the stop-list contains meaningless words such as prepositions, adverbs, adjectives, and some verbs. For example, the words "the", "with", "very" are usually in English. On the other hand, sometimes, a noun with a very high occurrence frequency is also included in the stop-list. In fact, a word, which occurs in 80% of the documents in the collection, is useless for purposes of retrieval [Salton83].

2. Word stemming

Not all the morphological differences between words are significant from a semantic point of view. Different word forms may denote the same meaning. This is the case for conjugated verbs, singular and plural forms, and so on. In order to remove such non-meaningful morphological differences, stemming is usually applied. Word

stemming removes word suffixes so that only word stems are kept. For example, all of following words are stemmed into the same form "connect" as follows.



To do stemming, a set of stemming rules is usually used. For example, one may remove all the –ive suffix as in the case of "connective". Some more sophisticated process may also be used. For example, Porter proposes a stemming process consisting of several successive steps, each step dealing with one type of suffix.

Stemming can improve the performance of IR system in several ways. First, it can reduce the size of the indexing structure. For the above example, five keywords are transformed to the same "connect" word. This may increase the search speed. Second, it also increases the recall, which is the capability to find all the relevant documents. For example, if a user query is "connection", after a query stemming, the document retrieval will cover the documents that contain terms "connecting", "connected" and so on. In this case, the retrieval coverage is extended, and the recall can be improved.

Instead of stemming, one can also use a more linguistically motivated approach such as lemmatization, which determines the standard form of a word (e.g. singular form of nouns). We will not give details about it.

3. Term weighting

The weights of index term indicate the importance of terms in a document. They can be calculated in many different ways. The *tf* * *idf* weighting scheme a widely used weighting method. In this weighting scheme, the *tf* denotes term frequency and *idf* denotes inverted document frequency. The weight w_t of a term t is based on its local frequency in the document (*tf*) and its distribution in the entire document collection (*idf*). One of the *tf* * *idf* formulas is as below:

$$w_t = [\log(f(t, d)) + 1] \times \log (N/n)$$

where: $f(t, d)$ is the frequency of the term t in the document d,

N is the total number of documents in the collection,

n is the number of documents including t.

In this formula, $[log(f(t, d)) + 1]$ is derived from the term frequency $f(t, d)$, and $log (N/n)$ is what we call *idf*.

The basic idea of *tf* * *idf* is that – the more frequency a word appears in a document, the more the word is important (*tf*), the less a word appears in different documents, the more specific the word is to the document (*idf*). Therefore, it should be weighted higher for this word. There are several other weighting methods used in IR systems, for example, probabilistic weighting, which we will describe in a later subsection.

2.2 Retrieval models

Information retrieval aims to find out the relevant information for a user need. It matches the query terms and index term from document, and ranks the retrieval

results to display for the user. The ranking of search results usually, follows a retrieval model. This section will give more detail about the most commonly used retrieval models, namely vector space model, Boolean model and probabilistic model.

2.2.1 Vector space model

In vector space model, a document, as well as a query, is represented as a vector of weights [Nie01]. A graphical illustration is shown in figure 1, in which *t1*, *t2* and *t3* are terms that correspond to dimensions of the vector space.

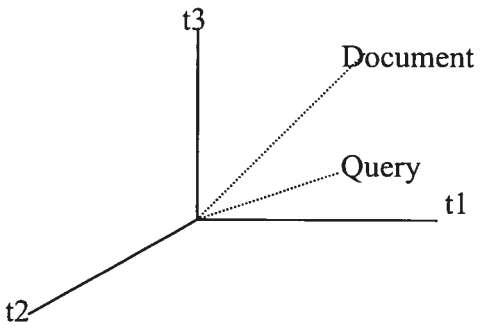


Figure 1. Document and query represented in vector space

The vector space is defined by all the index terms, which are usually selected words in the document during indexing, as we described in Section 2.1.

Suppose that there are terms t_1, t_2, ..., t_n, then the vector space is defined as follows:

Vector space: $< t_1, \quad t_2, \ ..., \ t_n >$

A document d and a query q are represented as the following vectors of weights:

$$d \rightarrow <w_{d_1}, \ w_{d_2}, ..., w_{d_n}>$$

$$q \rightarrow \langle w_{q_1}, w_{q_2}, \ldots, w_{q_n} \rangle$$

where: w_{d_i} and w_{q_i} are the weights of term t_i in document d and query q. These weights can be determined by the *tf*idf* weighting scheme.

Query matching involves measuring the degree of similarity $sim(d, q)$ between the query vector q and each document vector d. The following calculation of similarity is among the ones often used in IR:

$$sim(d, q) = \frac{\sum_{i=1}^{n} (w_{d_i} * w_{q_i})}{[\sum_{i=1}^{n} (w_{d_i}{}^2) * \sum_{i=1}^{n} (w_{q_i}{}^2)]^{1/2}}$$

Another common formula is the inner product:

$$sim(d, q) = \sum_{i=1}^{n} (w_{d_i} * w_{q_i})$$

The documents are then ranked in the reverse order of their similarities to the query.

2.2.2 Boolean model

In Boolean IR model, documents are represented by sets of terms that are considered to be index terms, and queries are represented by Boolean expressions on terms that are connected by Boolean operator AND, OR, and NOT. For example, "(A AND B) OR C". In practice, the user just enters a natural language query, which the system converts into Boolean expressions. Boolean retrieval considers Boolean operators as follows:

"A OR B" retrieves those items that contain the term A or the term B or both;

"A AND B" retrieves those items that contain both terms A and B;

"A NOT B" retrieves those items that contain term A and do not contain term B.

In the above expression, no term weighting is used. This raises the problem for ranking documents in the result list, i.e., there is no means allowing us to order the documents in that list. To solve this problem, term weighting should be incorporated in a Boolean model. In this way, query evaluation can take into account the weight of the terms, and result in a non-binary value. For example, one of the extended Boolean models uses the following evaluation. In this case, assume that w_t is the weight of term t in a document, and the correspondence $R(d, q)$ between a document d and a query q is as follows:

$$R(d, t) = w_t$$

$$R(d, q_1 \text{ AND } q_2) = \min(R(d, q_1), R(d, q_2))$$

$$R(d, q_1 \text{ OR } q_2) = \max(R(d, q_1), R(d, q_2))$$

$$R(d, \text{NOT } q_1) = 1 - R(d, q_1)$$

Here, the operator "min" and "max" are used for "AND" and "OR".

The documents that have the highest values of R with the query are presented to the user first in the retrieval result.

2.2.3 Probabilistic model

A probabilistic approach was first proposed in IR in the 1960s. It has been further developed and applied in many IR systems such as OKAPI system, which is

developed at City University in London. The probabilistic model aims to capture the IR problem within a probabilistic framework [Salton83].

The earliest probabilistic model is called independent probabilistic model [Nie01]. Its goal is to calculate the probability that a document entails the satisfaction of a query, formulated as the following conditional probability: $P (R_q | d)$. This probability is further developed using Bayes Law as below:

$$P(R_q|d) = \frac{P(d|R_q) * P(R_q)}{P(d)} \qquad (1)$$

Where :

$P(d)$ - the probability that a document d is selected. It is assumed to be a constant that is only dependent on the document collection.

$P(R_q)$ - the probability that a query q may be satisfied by a relevant document. It is another constant that is only dependent on the query.

$P(d|R_q)$ - the probability that a document d is part of the relevant documents for the query. This is the key element to be estimated.

$P(R_q|d)$ – the probability that a query q is satisfied by a relevant document if the document d is presented.

If we consider all the dependencies of the elements (words) appearing in a document d, it is difficult to estimate $P(d|R_q)$. To simplify it, the following independence assumption is made: The terms in d are stochastically independent. This makes it possible to decompose d into all terms that occur in it. Let us denote the

presence and absence of a term t_i by $x_i = 1$ and $x_i = 0$ respectively. Then a document d can be represented as a set of x_i as below:

$d = \{ x_1, x_2, ... x_n\}$

Using the independence assumption, $P(d|R_q)$ becomes the following one as be:

$$P(d|R_q) = \prod_{i=1}^{n} P(x_i \mid R_q) = \prod_{i=1}^{n} P(x_i=1| R_q)^{xi} P(x_i=0| R_q)^{(1-xi)}$$

$$P(d|R_q) = \prod_{i=1}^{n} P(x_i=1| R_q)^{xi} P(x_i=0| R_q)^{(1-xi)} \qquad (2)$$

In order to estimate $P(x_i=1| R_q)$ and $P(x_i=0| R_q)$ i.e., the probabilistic of the presence or absence of a term among the relevant documents, one can use a set of samples of documents whose relevance is judged manually. In an IR system, it is not important to obtain a precise value for $P(R_q|d)$ because one is only interested in ordering documents according to their relevance estimation. For this purpose, one often uses the comparison of $P(d|R_q)$ and $P(d|NR_q)$, where NR_q represents the set of non-relevant documents, to estimate if a document should be retrieved, and how it should be ranked in the result [Nie01]. Therefore, one often uses the Odd expressed as the following formula:

$$Odd(d) = \frac{P(d|R_q)}{P(d|NR_q)} = \frac{\prod_{i=1}^{n} P(x_i = 1 | R_q)^{x_i} P(x_i = 0 | R_q)^{(1-x_i)}}{\prod_{i=1}^{n} P(x_i = 1 | NR_q)^{x_i} P(x_i = 0 | NR_q)^{(1-x_i)}} \qquad (3)$$

Or its logarithm form:

$$log\ Odd(d) = log \frac{P(d \mid R_q)}{P(d \mid NR_q)} \quad (4)$$

In 1976, Sparck Jones and Robertson derived a formula from the above *Odd* that gives a better numeric weight to a term t from a query q. The formula is as follows:

$$w_{(t)} = log \frac{p(1-q)}{(1-p)q} \quad (5)$$

where:

p is the probability that term appears in a relevant document as, i.e. $P(x_i=1 \mid R_q)$.

q is the probability that term appears in a non-relevant document i.e. $P(x_i=1 \mid NR_q)$.

$1 - p$ is the probability that term does not appear in a relevant document i.e. $P(x_i=0 \mid R_q)$.

$1 - q$ is the probability that term does not appear in a non-relevant document i.e. $P(x_i=0 \mid NR_q)$.

If we N is the number of indexed documents, n the number of documents containing the term, R the number of known relevant documents among the samples, and the r is the number of relevant documents containing the term, these probabilities are estimated as follows:

$$p = P(x_i=1 \mid R_q) = \frac{r}{R}$$

$$q = P(x_i=1 \mid NR_q) = \frac{n-r}{N-R}$$

$$1-p = P(x_i=0|\, R_q) = \frac{R-r}{R}$$

$$1-q = P(x_i=0|\, NR_q) = \frac{N-n-R+r}{N-R}$$

Therefore,

$$w_{(i)} = log\ \frac{p\,(1-q)}{(1-p)\,q} = log\ \frac{r/(R-r)}{(n-r)/(N-n-R+r)} \qquad (6)$$

The problem with this formula is that for terms that do not appear in the sample documents, its probability is undetermined. Therefore, a smoothing factor of 0.5 is add into the above formula, which is also used in the OKAPI system reading to the following formula:

$$w_{(i)} = log\ \frac{(r+0.5)/(R-r+0.5)}{(n-r+0.5)/(N-n-R+r+0.5)} \qquad (7)$$

This formula is called Robertson-Sparck Jones formula.

2.2.4 Discussion

Boolean model allows for complete query expression with logical operators "and", "not" and "or". However in the unweighted Boolean model, the IR system may retrieve too few or too many documents. In general, as the original user query is usually not in Boolean form, it is difficult to transfer the query into a Boolean expression.

Vector space model allows partial matching. Its similarity calculation allows us to rank the documents according to their degree of similarity to the query. However, the assumption that index terms are orthogonal is not always reasonable.

An IR system can use a probabilistic model. In recent experiments, it has been shown that some modified probabilistic models are effective for retrieval on large collections. This is the case for OKAPI, which we will describe in the next chapter in more detail. However, the problem of using probabilistic model is that it needs a set of relevant and non-relevant documents to estimate the probabilities of index terms.

In our work, we choose to use a probabilistic model, in particular the OKAPI system, because it has shown consistently good results in IR experiments in Text Retrieval Conference (TREC). We will give a more detailed description of OKAPI in chapter 3.

2.3 Other search methods

Beside the basic indexing and retrieval methods we just described, there are a number of techniques developed in IR in order to improve the quality of retrieval. A few of them are described in this section that are related to our project.

2.3.1 Passage retrieval

Traditional IR retrieves complete documents for a query. A problem with document retrieval is that a long document is treated in the same way as a short document. A long document could include several different topics, while some particular topics are probably not developed in depth. Thus, it is difficult to find

proper information from such a document even if a topic is mentioned in it. Since different topics exist in different portion of a document, a complete document can be cut into several passages so that IR systems are able to identify relevant passages instead of full documents. Passage retrieval aims to retrieval fragments information from texts instead of complete documents.

There are different ways of defining passages. In general, we can identify three types of passage: discourse passage, semantic passage, and window passage. A discourse passage is a unit of document that corresponds to a unit of discourse such as a paragraph or a sentence. A semantic passage is a logical unit of a text that presents a logical concept. It may be a group of several sentences. A window passage is defined as a sequence of words of fixed length. The length of the sequence is called window size. Usually, the selection of a particular passage type is influenced by the suitable passage size. The passage size can affect retrieval performance. If the passages are too small, the IR system performance may suffer because there is a high probability that relevant concepts are separated by passage boundaries. On the other hand, if we use passages that are too large, there may be increasing noise in the retrieve because irrelevant passages may contain sufficient query terms to match the query.

Passage retrieval has proven to be an efficient way of identifying instances and minimising user effort for extracting relevant information [Salton83]. This is especially the case when the length of documents varies a lot. Passage retrieval can also be used as the first selection means for a more complex task such as question answering.

For passage retrieval, indexes are created for each passage. To do this, a document is first broken down into more or less equally sized passages before indexing. Those document passages are then matched against the query. Passages can be provided to the user as the retrieval result. It is also possible to return complete documents based on passage-level evidence, or a combination of passage-level and document-level evidence [Salton83]. For example, the similarity of a document to a query may be the sum of the similarities of its passages to that query.

In a domain-specific area or in the case where an IR system is used to locate precise information, passage retrieval is better than document retrieval because the precise answer usually is located in a small portion of the text, It is better to provide the related passages as answer rather than asking a searcher to browse a long document in order to find a specific passage. In our case, we want to construct a question answering system for the construction sector. The first selection of potential texts should be done with passage retrieval instead of document retrieval.

2.3.2 Using compound term

Usually, keywords used in IR are single words. However, in nature language, sometimes, the compound terms have more precise meaning in the document that cannot be presented by any single word. Thus, developing IR system with compound term can increase retrieval precision logically.

Without a specific way of dealing with compound terms, these latter will be separated into single words. For example, the term "computer hardware" will be replaced by "computer" and "hardware". In keyword-based IR systems, such a

transformed query documents containing both "computer" and "hardware" but not necessarily "computer hardware". In order to solve this problem to increase retrieval precision, we aim to use compound terms such as "computer hardware" as additional index terms.

To achieve this goal, we first have to recognize compound terms. The recognition of compound term will consider syntactic analysis and/or the use of an available thesaurus.

Syntactic analysis aims to identify the syntactic structure of sentences. From the result of this result, it is possible to identify noun phrases as document descriptors in order to provide more accurate representation of the document's contents. In fact, a set of syntactic patterns such "NOUN NOUN" or "NOUN PREP NOUN" is used for the identification. However, the problem with syntactic analysis is that along with the correct terms identified, wrong groups of words also come out. For example, from "question answering system", not only the correct term "question answering" is identified, but also "question system" and "answering system" are generated. These noisy terms will increase the noise in retrieve.

Thesaurus is another resource for the identification of compound term one tries to identify the terms that are stored in the thesaurus. Of course the phases in thesaurus are correct terms to be identified. It would be more precise to use those terms as indexes of document contents.

In a specialized area such as construction, specialized terms are often compound terms. Therefore, it is particularly important to identify them. In the construction sector, a specialized thesaurus has been made available to us. Therefore,

we use the thesaurus of construction sector to identify compound terms in documents and queries. The identified compound terms will be used as additional indexes.

2.3.3 Query expansion

Traditional IR requires that retrieved documents contain the same keywords/terms as the query. However, if a relevant document does not contain the same terms as those of the query, then that document will not be retrieved. However, the terms in the document may be strongly related to the terms in the query, thus the document is still relevant. In this case, query expansion is a way to reduce such a mismatching problem. The aim of query expansion is to reduce query and document mismatch by expanding the query using words or phrases with a similar meaning, or with a strong relationship.

Query expansion consists of expanding an initial query of the user by adding some related words in it. This will result in a new query. The addition of the new words extends the original query so that it has a wider coverage than the original query [Nie01]. In this case, even if a set of document representatives – index terms do not match the same words as the original query, it can still be judged to be relevant if it contains the keywords that are added in through query expansion. As a consequence, more relevant documents may be retrieved, and the recall ratio may be increased [Nie02].

Query expansion has long been suggested as an effective way to resolve the short query and word mismatching problems. A number of query expansion methods

have been proposed in traditional information retrieval. Some IR systems use an approach based on the similarity thesaurus to execute query expansion by adding similar terms into the queries. The similarity thesaurus is based on term-to-term relationships derived from a matrix of co-occurrence of terms. Another way is to use synonym words, which are stored in thesaurus, to perform query expansion by adding synonym words into the query. For example, if we know that "construction" is a synonym of "building", then a query on "construction" will be extended to "construction" and "building".

In our project, we also choose to use query expansion in our system because we want to find all of the potential answer of user's query, even if the user users a different term. To perform query expansion, we will make use of a domain-specific thesaurus, which contains synonyms of terms.

2.4 Web IR

The Web is creating new challenges for IR systems. Every year, the amount of information available on the Web grows exponentially and has reached over one billion pages by the year 2002. The Web has become a common place for people to search for information, do their shopping, etc. Web IR performs IR on the Web to help people finding their needed information. However, there are a few important differences with the traditional IR.

- Dynamic data collection

  Traditional data collection is static, however, the Web is dynamic. Therefore, we need to collect documents dynamically with a crawler.

- Link analysis

Link analysis tries to exploit the links between documents in order to provide additional criteria for document retrieval. PageRank algorithm [Page98] uses the connections between documents to weigh the importance of pages. The HITS algorithm [Kleinberg98] further refines the connection and derives two important types of page: hub and authority. A higher degree of hub or authority of a page would increase the weight of that page.

- New ways for term weighting

In general, the Web IR system gives preference to the words found in the title or leading paragraph or in the metadata of a document so that the terms occurring in the title of a document or page that match a query term are weighted more heavily than terms occurring in the body of the document. Similarly, the query terms appearing in section headings or the first paragraph of a document may be more likely to be relevant.

- Popularity

Popularity is a measure of the frequency that a page is used. If many users choose a page, it is a sign that many users prefer that page, so its weight should be increased.

- Date of publication

It is usually assumed that the more recent the information is, the more likely that it will be useful or relevant to the user. Therefore, most Web IR system present search results beginning with the most recent to the less recent.

In our project, although the final system will work in the Web environment as a search engine, this aspect is not yet the central concern at the current stage. We are more concerned with the way that Web pages are crawled so that indexing may be performed on it. In our project, we will implement a simple crawling mechanism, in collaboration with Web master assistance, in order to gather new Web pages related to the construction area.

2.5 Question Answering System

Question answering (QA) is different from IR in that it has to find the answer to a question by returning a small fragment of a text, where the answer actually lies. In contrast, IR systems allow us to locate full documents that might contain the pertinent information, leaving it to the user to extract the answer from a ranked list of texts.

As there is more and more information on the Web, QA is more and more required in order to identify precise answers to a question. There are several portals on the Web providing QA services, for example, the START system (http:// www. ai.mit.edu/projects/infolab/start.html) and AskJeeves (http:// www. askjeeves. com).

The START Natural Language System is a system designed to answer questions that are posed to it in natural language START parses incoming questions, matches the queries created from the parse trees against its knowledge base and presents the appropriate information segments to the user. START provides untrained users with speedy access to knowledge that in many cases would take an expert some time to find [Salton83].

AskJeeves is equipped with a fairly sophisticated natural language question parser. If it finds an answer in its factoid database, an answer is generated. Otherwise, it performs IR as a traditional search engine does [Robertson92].

Usually, QA systems use architecture as the one shown in figure 2 below.

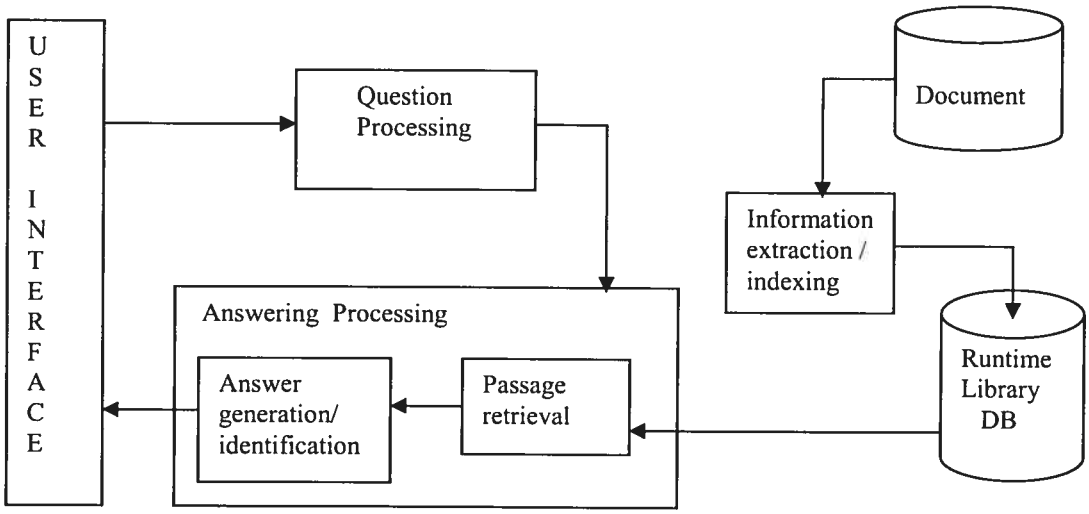


Figure 2. The Question Answering System Architecture

- The question processing module parses the user question. The role of this module is to:
    1. Determine the type of the question, for example, when, where, who, and how far etc..
    2. Determine the type of the answer expected, for example, time and date (when), place and position (where), person (who), and length (how far) etc..

3. Transform the question into queries for the search engine, for example, elimination stop words, query keyword extraction, and/or query expansion.

- The information extraction module identifies special type of information from the texts such as dates, addresses, person names, company or institute name, and so on.
- The answer processing tries to identify and extract the answer from the paragraphs that correspond to the question type and extracted keywords. It usually contains two parts: one for passage retrieval in order to identify segments of text which may contain an answer; and the other for answer verification / generation (passage analysis) in order to confirm if it contains the required answer.

In our project, as our final goal is also to create a QA system for construction sector, all the aspects just mentioned above are concerned. However, the further analysis of questions and answers are developed by two other master degree students in our research group. The present study is only concerned with the construction of a basic IR system which provides a passage retrieval mechanism and integrate some domain-specific knowledge, in order to provide the further analysis with a set of candidate passages. So in the following chapters, they will only talk about the specific work in this study, namely the following aspects:

- Passage retrieval by adapting an existing IR system – the OKAPI system.
- Compound term extraction with the help of a domain-specific thesaurus.
- Query expansion with the domain-specific thesaurus.

- A simple user interface with CGI application.

As the backbone IR system, we will use in this project is OKAPI. We will describe this system in more detail in the next chapter before presenting our work.

Chapter 3. The OKAPI System

OKAPI is an experimental text retrieval system, designed to use simple, robust techniques internally. It is developed in City University at London. It can be used for searching files of records whose fields contain textual data of variable length. It implements a variety of search techniques based on probabilistic retrieval model, with easy-to-use interfaces, for databases of operational size and under operational conditions [Robertson92].

OKAPI has been used in several TREC experiments. It has consistently produced one of the best performances among all the participants. It is available to researchers for research proposes. Due to these reasons, we chose OKAPI as our basic IR system. This chapter will give a brief description of the OKAPI system.

3.1 The Architecture of the OKAPI system

OKAPI uses a standard three-tiers architecture as figure 3 indicates.

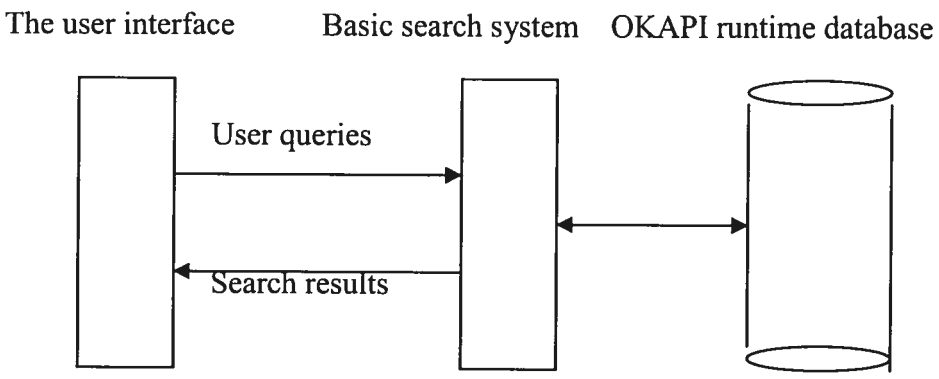


Figure 3. The OKAPI project architecture

1. The OKAPI interactive interface

The first tier is user interface. It is an interactive interface for user to use this system. A user can submit a query in the query box. The retrieved results will be displayed in the result window. If the user wants to further read a document, he only has to click on that document.

2. The OKAPI search system

The second tier, the Basic Search System (BSS), is the core module in the OKAPI system. The BSS provides retrieval functionalities to produce ranked output. It is designed primarily for probabilistic retrieval of textual materials using inverted indexing files together with built-in weighting functions. It is a low level platform offering APIs to perform the searching process. The BSS was implemented in the language C. Developers who want to develop a complex retrieval system using the OKAPI system can call the BSS APIs. When BSS receives the queries, it will look into the indexed OKAPI runtime databases and the invert files to find out the correct documents by keywords. We will go into more details about BBS in the next section.

3. The OKAPI runtime database

The third tier, OKAPI runtime database, is a library of the OKAPI system, which allows accessing the index files and the documents. This is a set of basic data access tools. Notice that OKAPI also have a set of indexing tools allowing

constructing the runtime database. These tools are used offline (i.e. before any user query is submitted).

In order to construct an IR system using OKAPI, we usually use the APIs provided by OKAPI.

3.2 Probabilistic models used by OKAPI

OKAPI implements a series of probabilistic models. We give a brief description of them in this section.

3.2.1 Basic Robertson-Sparck Jones model

As we described in chapter 2, the basic probabilistic model uses the following weighting formula for a term t in a document D, which is called Robertson-Sparck Jones formula:

$$w_1(t,D) = log \frac{p(1-q)}{(1-p)q} = log \frac{(r+0.5)/(R-r+0.5)}{(n-r+0.5)/(N-n-R+r+0.5)} \quad (1)$$

where:

N is the number of indexed documents;

n is the number of documents containing the term;

R is the number of known relevant documents; and

r is the number of relevant documents containing the term.

This weighting formula has been implemented in OKAPI as one of the weighting schemas, called the *BM1* weighting schema. In OKAPI, the search terms or

index terms are keywords or phrases, or any other record components. They are extracted automatically from a query or a document.

In practice, no relevance feedback is available. This means that we do not know R and r for each term. Therefore, search terms are assigned weights, based on inverse document frequency (*idf*) without relevance information, i.e. we assume $R = r = 0$ in the above formula, leading to

$$w_1(t,D) = log \frac{(N - n + 0.5)}{n + 0.5}$$

which is similar to the classical *idf* $\log \frac{N}{n}$.

Several other weighting schemes are also implemented in OKAPI. They are based on a different probabilistic model – 2-Poisson model.

3.2.2 2-Poisson model

In formula BM1, the calculation does not consider the term frequency (*tf*) in a document. It is clear that this factor should be taken into account for better retrieval performance. The 2-Poisson model tries to consider this factor. This model postulates that the distribution of within-document frequencies of a content-bearing term is a mixture of 2-Poisson distributions: one set of documents – the "elite" set for the particular term (which may be interpreted to mean those documents which can be "about" the concept represented by the term) will exhibit a Poisson distribution of a certain mean, while the remainder may also contain the term but much less frequently [Robertson93]. The following formula considers the within-document term frequency (*tf*) for a term t which occurs *tf* times in a document D [Robertson93].

$$w_2(t,D) = \log \frac{(p'\lambda^{tf}e^{-\lambda} + (1-p')\mu^{tf}e^{-\mu})(q'e^{-\lambda} + (1-q')e^{-\mu})}{(q'\lambda^{tf}e^{-\lambda} + (1-q')\mu^{tf}e^{-\mu})(p'e^{-\lambda} + (1-p')e^{-\mu})} \quad (2)$$

where :

λ is the Poisson mean for *tf* in the elite set for *t*;

μ is the Poisson mean for *tf* in the non-elite set;

p' is the probability of a document being elite for *t* given that it is relevant;

q' is the probability of a document being elite given that it is non-relevant; and

tf is term frequency of the document.

It is clear that the formula (2) for $w_2(t,D)$ is difficult to calculate; but from this formula we can see how *tf* behaves. There is an approach that suggests a simpler function of *tf*, which behaves in a similar way. If *tf* is zero, the $w_2(t,D)$ is zero too. If *tf* gets large, the weight will be close to an asymptotic maximum, which is *tf/(tf + constant)*. As shown in [Salton83], the maximum is approximately the binary independence weight that would be assigned to an infallible indicator of elite. From this observation, a simplified formula is proposed, which multiplies the asymptotic maximum with the appropriate binary independence weight, i.e. the formula (1) BM1. This results in the following new weighting formula:

$$w_3(t,D) = \frac{tf}{(k_1 + tf)} w_1(t,D) \quad (3)$$

where k_1 is a constant that depends on the weight $w_1(t,D)$ and *tf*.

The constant k_1 in the formula is not in any way determined by the argument. The effect of this constant is to determine the strength of the relationship between the

weight and *tf*. This has an asymptotic limit of unity, so it must be multiplied by an appropriate OKAPI basic weight $w_1(t,D)$. In practice, k_1 is set at around 1.

3.2.3.Document length

For document indexing, the above 2-Poisson model assumes that documents are all of equal length. But the difference in document length could have a great impact on the results of retrieval. Usually, a longer document will have a higher *tf* for a term than a shorter document. As a consequence, longer documents are favoured by the formula (3). However, longer documents are not necessarily preferred to shorter ones. In many cases, users prefer shorter, but more specific documents to longer but less specific documents. In order to correct the bias created by the length factor, the following weighting formulas integrates an average document length as follows:

$$w_4(t,D) = \frac{tf}{(\frac{k_1 \times dl}{avdl} + tf)} w_1(t,D) \qquad (4)$$

where:

avdl is the average document length;

dl is the length of the document *D*.

So the weight of each term depends on both *tf* and *dl*. The higher the *dl* of a document, the lower the $w_4(t,D)$ in that document.

3.2.4 Document length and number of query terms

Another correction factor (*cf*) tries to make the weight dependent on the document length and the number of terms in the query (*nq*) when terms match. For appropriate matching value in document similarity, two components have to be considered:

1. One component is the sum of term weights, and the weight of each term depending on both *tf* and *dl* as above presented; and
2. The other component is the correction factor that depends on *dl* and *nq*. This component is only added for document match not for terms match.

   The correction factor (*cf*) behaves as follows:

   1. when *dl* tends to zero, *cf* tends to the maximal value, i.e. $dl \rightarrow 0$, $cf \rightarrow \max$.
   2. when *dl* equals to *avdl*, *cf* tends to zero, i.e. $dl = avdl$, $cf \rightarrow 0$.
   3. when *dl* tends to the maximal value, *cf* tends to minimum value, i.e. $dl \rightarrow \infty$, $cf \rightarrow \min$.

   The new weighting formula with the correction factor is as follows:

$$\mathrm{cf} = \mathrm{k}_2 \times \mathrm{nq} \ \frac{(avdl - dl)}{(avdl + dl)} \qquad (5)$$

where:

k_2 is another constant;

nq is the number of terms in the query.

3.2.5 Weighting query terms

On the query side, one may also take into account the term frequency. The more frequent a term is, the more important it is. In considering this factor, the weighting formula could be as follows for a terms *t* occurring in the query Q:

$$w_6(t,Q) = \frac{qtf}{(k_3 + qtf)} w_1(t,D) \tag{6}$$

where:

qtf is the query term frequency.

k_3 is a constant playing the same role as k_1 in formula (3).

Several combinations of the weighting formulas and functions discussed above have been implemented at one time or another during the development course of OKAPI. Most of the functions are referred to as Best Match (BMxx) functions in OKAPI as follows.

$$\text{BM1: } w = \log \frac{(r+0.5)/(R-r+0.5)}{(n-r+0.5)/(N-n-R+r+0.5)} = w_{(1)} \tag{7}$$

$$\text{BM11: } w = s_1 s_3 \times \frac{tf}{(\frac{k_1 \times dl}{avdl} + tf)} \times \frac{qtf}{k_3 + qtf} \times w_{(1)} \oplus k_2 \times nq \frac{(avdl - dl)}{(avdl + dl)} \tag{8}$$

where:

s_1, s_3 are scaling constants related to k_1, k_3;

$\oplus$ indicates the following component is added only once per matched documents. It is second component for matching value.

$$\text{BM15: } w = s_1 s_3 \times \frac{tf}{k_1 + tf} \times \frac{qtf}{k_3 + qtf} \times w_{(1)} \oplus k_2 \times nq \frac{(avdl - dl)}{(avdl + dl)} \tag{9}$$

In order to combine BM11 and BM15 in a unified formula, the term frequency component in formula BM11 (8) (i.e. $\frac{tf}{(\frac{k_1 \times dl}{avdl} + tf)}$), and BM15 (9) (i.e $tf/(k_1 + tf)$) is replaced by: $tf/(k + tf)$. This result is the following BM25 formula:

$$\text{BM25: } w = s_1 s_3 \times \frac{tf}{k_1((1-b) + b(\frac{dl}{avdl})) + tf} \times \frac{qtf}{k_3 + qtf} \times w_{(1)} \oplus k_2 \times nq \frac{(avdl - dl)}{(avdl + dl)} \quad (10)$$

where $k = k_1((1-b) + b(\frac{dl}{avdl}))$

We notice that if $b = 1$ it is BM11; if $b = 0$ it becomes BM15.

3.3 The modules of the OKAPI system

OKAPI contains eight modules: user interface module, stemming module, basic search system module, browser module, indexing module, stop words module, OKAPI exchange DB module, and OKAPI runtime DB module. Figure 4 illustrates how these modules work together for a typical IR task.

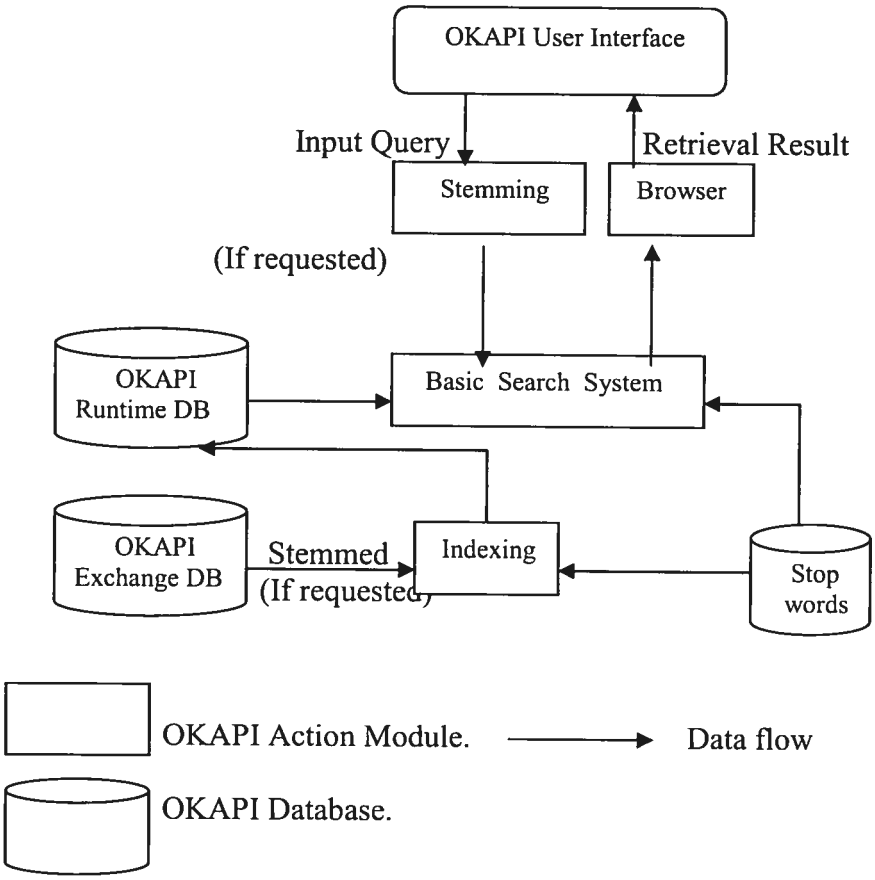


Figure 4. The module structure of the OKAPI system

1. OKAPI Exchange DB

The OKAPI exchange DB holds the data in OKAPI exchange format. This format is used for OKAPI indexing. The OKAPI exchange format is as follows:

- Every record in the database must have the same number of fields with an "end of record" character;
- Each field in an exchange format record is terminated by an "end of field" character.

A record is a complete unit in the converted DB. A filed is a unit for OKAPI indexing. One record can contain several at most 31 fields. Although any fields should exist in a record, the field could be empty.

Once a set of documents is transformed into this format, the OKAPI indexing can take place.

2. Stopwords

   OKAPI can incorporate any stoplist that the system administrator provides. All the words in this list will not be taken as indexes (see Chapter 2).

3. The stemming Module

   This model aims to remove suffixes from words appearing in a document or query. We have described this function in Chapter 2.

4. The indexing Module

Before documents can be indexed, they first have to be converted into a suitable internal form. Then a large index file is constructed, which indicates for each index term, what documents contain the term, and how important it is. The importance is calculated according to one of the weighting formulas we described in the previous section.

When new documents must be added, OKAPI has to re-index the whole collection. So OKAPI does not support incremented indexing. This is due to the weighting formulas used.

5. OKAPI Runtime Library

The OKAPI library holds an inverted file. In IR, the inverted file is used to store a record for each index term, which lists each collected documents, which

contain the term. The inverted file allows for a quick identification of all the documents that contain a term.

6. Basic Search System

The basic search system (BSS) is the OKAPI's core search tool. It provides a set of APIs offering text retrieval functionalities. The BSS consists of a set of low-level API commands, implemented as a C library, which enables users to build their own interface based on it. In our project, we will build our own IR system upon OKAPI's BBS.

7. OKAPI user interface

OKAPI user interface is an interactive interface. It accepts user's query and then displays the retrieval results. However this interface does not satisfy all our needs. For example, it does not support multiple users, and cannot be directly connected to the Web. Therefore, we will not use the OKAPI interface and will construct our own user interface in our project.

8. Browser Module

The browser module handles the retrieval results. It shows a list of the retrieval results in decreasing order of their correspondence. The user can also provide relevance feedback on some of the results, and the system can incorporate this information and perform a second round retrieval. We will not use this module in our work.

3.6 Conclusion

We chose to use OKAPI system because it has proven its effectiveness in previous IR experiments. In addition, OKAPI also allows for passage retrieval, which fits the need of our project. We will use the OKAPI basic search system (BSS) for basic retrieval and the index module for document indexing.

We do not use OKAPI user interface. Rather we will construct our own because it does not support the online, multi-user, and cross-platform retrieval.

To summarize, here is the list of the main OKAPI BSS functions that we use:

For indexing:

```
1. int convert_db (structure db_param_info *db_info)
```

This function converts OKAPI exchange file to the OKAPI realized converted format for indexing preparation. The *"structure db_parameter_info"* holds all the pre-setting database parameters as the path of the exchange file etc..

```
2. int make_index (struct db_param_info *db_info)
```

To run this function, it will start to make document indexing. This function has to be run after previous *convert_db()* function.

For search:

```
1.   int open_database ( char *database_name)
```

This function opens the database whose name is passed to the program, then search can be performed in this database.

```
2.   void initialize_bss ( )
```

This function initializes the BSS. Every time, we want to use the BSS, we have to initialize it first.

3. `int do_search (char *term_file, int *docset, int *npostings)`

This is a BSS search function, allowing the system searching information from runtime database. The parameters "*term_file*" – the file that contains query terms, "*docset*" – the BSS document set number (return value), and "*npostings*" – number of posting result from BM25 search.

4. `void do_show (int set, int no_docs)`

This function displays the retrieval results for user.

Chapter 4. ERIC – A system for IR in construction sector

ERIC (Engin de Recherche d'Information Spécialisé en Construction), is an information retrieval system in a specialized domain – the construction area. This is a framework in which a component of Question Answer (QA) will be added later. The current ERIC framework tries to identify potential passages that may contain answers for a question or query. Those answers will be further analyzed by other components to confirm if they really provide an answer to the question.

ERIC is developed on top of OKAPI, which we described in the last chapter. It uses the basic indexing and search modules of OKAPI for indexing documents and retrieving a set of candidate passages. In addition, we also integrated in ERIC a Web crawler to download Web pages from a Web site selected manually. A simple user interface is also constructed, allowing the user to interact with the system through the Internet.

4.1 The requirement of the ERIC system

As ERIC aims to provide refined search capability for specialists in the construction area on the Web, it has the following requirements:

- Operability on the Web (online search)

  We want to provide online search through the Internet. Because HTML is the most popular data format used on the Web, the system should have the ability

to handle data in this format. In addition, a simple interface should be provided for interactions between search system and online users.

- Semi-Automatic document crawling and downloading from the Web

  As we described in Chapter 2, one has to construct a document collection dynamically for Web IR. In our case, we only want to include domain-specific documents in the collection. We do not integrate an automatic resource discovery tool in our system, because automatic discovery of specialized resources we will ask the Webmaster to select a set of Web sites related to construction, and the task of our crawler is to explore the selected Web sites and download the documents. The reason that motivated the manual selection is the requirement of including only high quality domain-specific documents. To achieve this, a manual selection is the most secure means. However, we do not require the Webmaster to indicate every Web page, but only the interesting Web site. Our crawler will explore the Web sites and download the Web pages stored on them into the local document collection.

- Content search

  The target users of ERIC are professionals in the construction sector who looks for precise information. The system should be precision-oriented. In our approach, we will make use of a domain-specific thesaurus to enhance the classical keyword-based retrieval. In addition, workers in the construction sector are often frustrated by irrelevant information retrieved by the search engines on the Web. As a result, they do not often use search engines in their

professional activities. So another goal of ERIC is to attract professional users by providing a few but relevant answers.

- Passage retrieval

  ERIC search engine will use passage retrieval instead of document retrieval because our final goal is to find short answers to query questions. The answers are usually contained in relatively short passages. Therefore, passage retrieval is more appropriate. For passage retrieval, ERIC has to delimit the boundaries of passages before OKAPI can index them.

4.2 The ERIC system architecture

ERIC aims to incorporate domain knowledge in IR operations. In our case, domain knowledge means a set of specialized terms and relations between them. The specialized terms will be used as more precise indexes, and the relations between them will be used for query expansion. The basic modules of the ERIC operations can be illustrated in figure 5. On the indexing side, the knowledge base provides specialized terms. On the retrieval side, it provides related terms for query expansion.

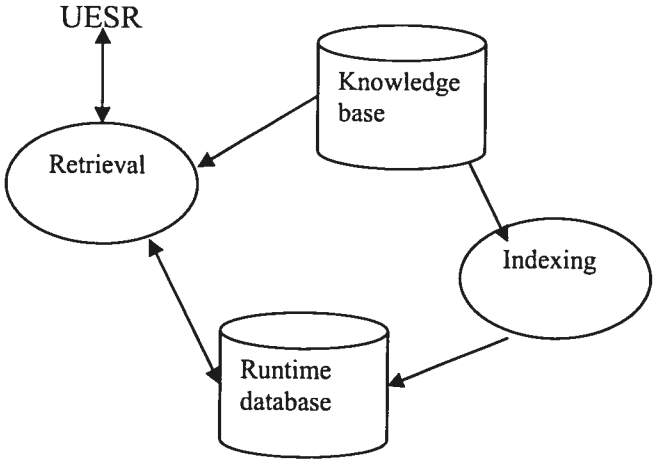


Figure 5. Core component of the ERIC system

Different functionalities are presented in some more detail below:

- The knowledge base is a resource, which is a domain-specific thesaurus. As the figure 6 indicates, the query analyser and document pre-processing will use this resource to identify domain-specific compound terms for indexing and to generate the relevant words for query expansion.

  The following figure illustrates the system architecture in more detail.

- The Indexing module includes:

  1. Document crawler, which constitutes an information collection from the Web:
  2. Document pre-processing, which prepares the documents for indexing such as segment document passages;
  3. Indexing, which calls the OKAPI indexing functions to generate the index file (runtime database).

- The retrieval module contains

  1. System interface, which accepts user's query and outputs the search results;
  2. User query analyser, which handles query analysis for generating an internal representation of the query after eliminating stop words, stemming words, and possibly performing query expansion; and
  3. Search, which calls the OKAPI basic search system (OKAPI BSS) to perform probabilistic search.

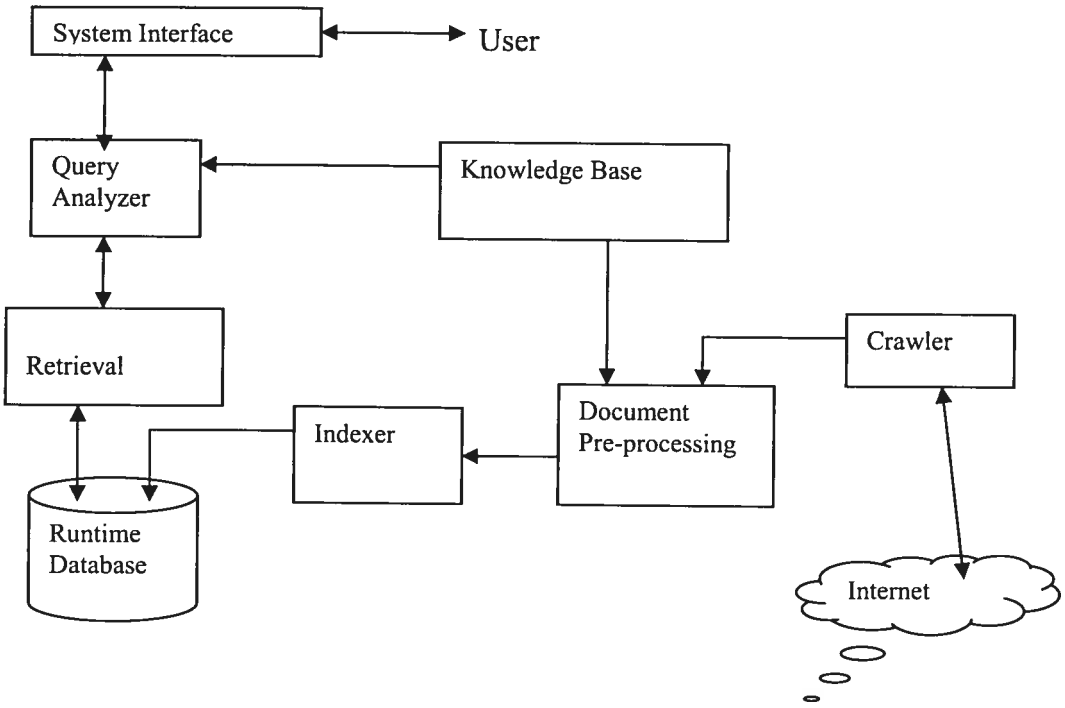


Figure 6. The system architecture

4.3 Approaches used in ERIC

4.3.1 Introduction

As we mentioned in chapter 2, a classical keyword-based retrieval is not precise enough. One of the means to increase precision is to use compound terms, which provide a more precise representation of the contents. As we indicated in chapter 2, one can use a dictionary of compound terms or a thesaurus to recognize the compound terms from texts. In comparison with a recognition based on a syntactic-statistical analysis, this approach only recognize the terms that have been manually confirmed to have a term status. It may be more precise. In our case, we have a

domain-specific thesaurus that contains a set of compound terms in construction. Therefore, we will use it in our system.

Another problem we raised in chapter 2 is that a concept can be expressed in several ways (synonyms). This also happen in specialized area such as construction. For example, "corrosion" and "erosion" are synonyms that can be used interchangeable in some cases. In order to allow a query matching a document containing a synonym, we will perform a query expansion. Again, synonyms have been stored in the thesaurus, so our query expansion process will exploit the thesaurus to find synonyms for query expansion.

There are many Web sites, but only a small number of them concerns the construction sector, and contains high-quality documents useful for construction professionals (architects, contractors, etc). A blind automatic Web crawling would not be able to crawl only documents in construction. Therefore, we ask the Webmaster to examine if a Web site is useful. A Web site judged to be useful is then crawled by our system. Currently, the IF research group has identified about 200 high quality Web sites related to the construction sector. The documents on these sites are deemed useful for construction professionals. Finally, as we have mentioned, for our purpose, a passage retrieval is more appropriate than document retrieval.

In following sections, we will describe in more detail how the above approaches are implemented.

4.3.2 Passage retrieval

Passage retrieval aims at retrieving the most relevant passages or document segments. OKAPI can be used for passage retrieval through the use of "fields". However, OKAPI does not determine the passages itself. We have to segment each document into passages before indexing. The segmentation of a document into passages is the first pre-processing that we perform, as shown in figure 7.

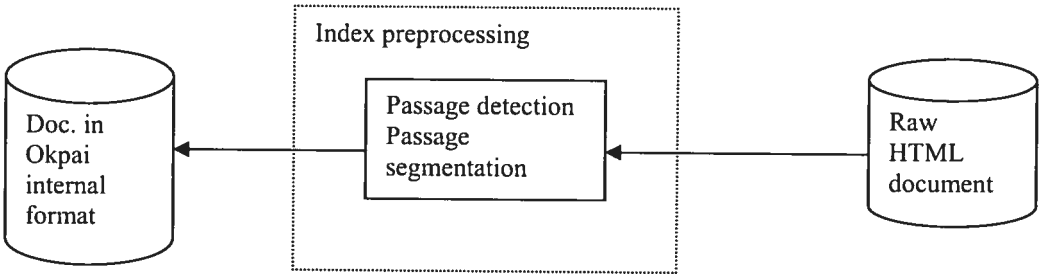


Figure 7. Index preprocessing in passage segmentation

As described in Chapter 2, there are different ways to cut a document into passages. In our case, we use a paragraph as a passage. This choice is motivated by the fact that a paragraph in technical documents usually describes a topic, and it is the most likely container of an answer to a question (however, this is not the only possible choice). As the documents we deal with are Web pages in HTML, it is easy to implement our segmentation process. We delimit the passages by <P> tags in documents. Normally, <P> tag introduces a new paragraph in HTML. So all the <p> tags are considered as a separator of passages.

Once all the paragraphs are marked as passages, OKAPI will follow the marked passages to index them. The retrieval results will be the passages instead of whole documents.

4.3.3 The thesaurus

Normally a thesaurus contains a set of hierarchical classes forming a tree structure as shown in figure 8 below:

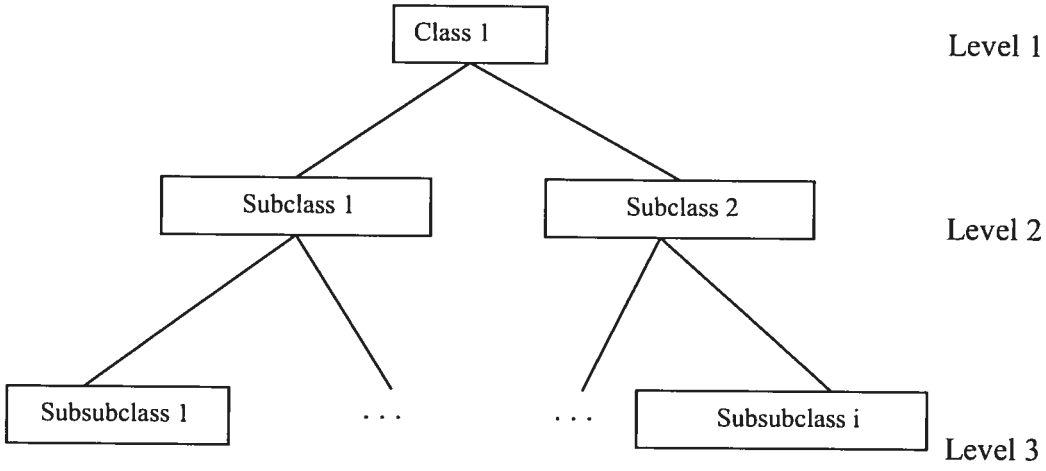


Figure 8. Hierarchical class structures in thesaurus

Each term in a class in the thesaurus represents a concept. A term may consist of one or more words. A thesaurus serves as an authority list of the terms used for indexing and retrieval. The thesaurus we use in our study has been developed by professor Colin H Davidson of the Faculty de l'aménagement. It is specific to the construction sector. It is provided as two large files: one contains all the terms, and the other contains all the relations. There are 15354 terms in this Thesaurus.

Through those two available thesaurus files, we can find out the query terms and their related terms. There are several types of relationships in the thesaurus to make up its own hierarchical classes, for example, BT relationship (Broader term relationship), NT relationship (narrower term relationship), WT relationship (Whole term relationship) etc. Those relationships can make the different data tree structures. However, we will only use PT and RT relation in this study for query expansion.

The table 1 below shows some examples.

Identification number	French term	English term	Level number
8756	Revenu national	National income	6
8760	Produit national	National product	6
13555	Revenu fiscal	Tax revenues	7

Table 1. Example of the thesaurus

In table 1, the identification number is a unique number assigned to a term. A concept is expressed in both French and English. The level number indicates the level in the class hierarchy where the term is.

The Following Table 2 illustrates the relationships between terms (concepts) in the thesaurus.

Term identify number	Term identify number	Term relationship
8756	8760	RT
8756	13555	PT

Table 2. Thesaurus terms relationship table

In this table, the concept 8756 ("*National income*") has an *RT* relationship of 8760 ("*National product*"), and a *PT* relationship of 13555 ("*Tax revenues*"). The "*RT*" is related term relationship. The "*PT*" is part term relationship. The *RT* relationship usually indicates the terms without equivalent or hierarchical, but they are semantically associated in thesaurus, as for the above examples of "*national income*" and " *tax revenue*". The *PT* relationship is kind of hierarchical relationships with the *WT*, but *PT* indicates the terms separated from their parents in thesaurus. For example, the above terms "*national income*" and "*national product*" all are in level 6 and have *PT* relationship. It indicated those two nodes are in same level, and associated to different parent nodes.

To process query expansion in ERIC, the Mysql database is used to process those two thesaurus files. The two files are converted into Mysql database, and the SQL language is used to find out expansion query terms through the relationships *PT* and *RT*

- Creating Mysql databases

1. create two database names to store two thesaurus files. One name "*Thesaurus*" contains all of terms. The other "*Lien*" contains those term relationships;

2. create database tables and then to load the available thesaurus files into database tables.

- Using Mysql databases

1. choose the correct database, for example, command "*use Thesaurus*" to choose named "Thesaurus" database to use; and
2. retrieval information from database tables by using the "*select*" command. For example:

   select id (term identification number) from Thesaurus;

   in Thesaurus where e_term (English term) ="construction";

The identification number of the term "construction" will retrieve immediately if this term is in database.

4.3.4 Compound terms

The thesaurus offers a means to identify compound terms. The identification of compound terms is done before the system performs documents indexing, and query searching. Compound terms are identified in both documents and queries. The following figure 9 shows how compound terms are added into a document and a query through a preprocessing:

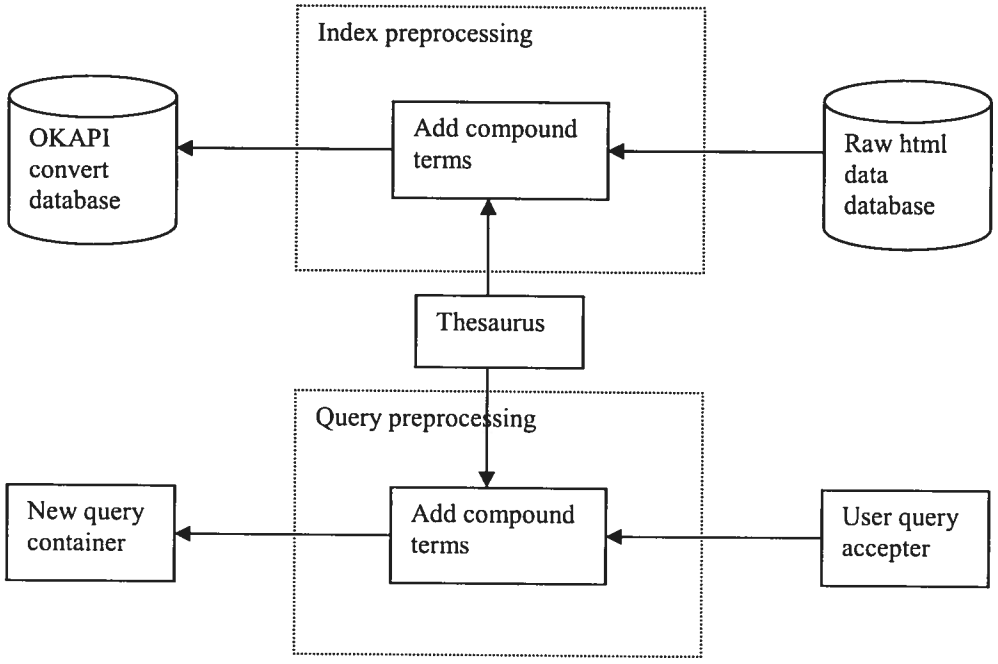


Figure 9. The compound term processes

The preprocessing goes through the text (a document passage or a query), and once a compound term is encountered, it is added into the text in a new field. For example, suppose we have the following passage:

"<p> Although there is no **standard test** available for assessing the capacity of mortar to contribute to efflorescence, cementing materials for mortar are available which are low in content of salts producing efflorescence, of which sodium and potassium compounds appear to be important. The use of lime, and of low-alkali **portland cement** and low-alkali **masonry cement** will greatly reduce the capacity of mortar to contribute to efflorescence. Careful storage of the masonry materials on the job site is also necessary to avoid contamination from salt-carrying **ground water**."

In the above passage, four compound terms appeared: "*standard test*", "*Portland cement*", "*masonry cement*", and "*ground water*". In ERIC system, once the term "*standard test*" is identified as a term in the thesaurus, the space in it is replaced

by underscore "_". The term "*standard_test*" is then added into the new field *<ADD TERMS>*. So for the above example, the following field is added:

"<ADD TERMS> standard_test, Portland_cement, masonry_cement, ground_water"

The reason to replace space by "_" underscore in compound terms is that we want the indexer of OKAPI consider the term as a unit, and "_" underscore is not considered as a word separator by OKAPI (an option offered in OKAPI).

4.3.5 Query expansion

In ERIC system, query expansion aims to expand the query terms by their synonym or related terms. It is a way to find the potential passages that do not contain the original query keywords, but their synonym or related terms. Query expansion is a way to solve this problem. Query expansion adds synonyms into the query so that the document that contains synonyms can also be identified.

The thesaurus is used again to help query expansion in ERIC system. As we can see in the following figure, query expansion takes place before the query is submitted to the OKAPI search tool.

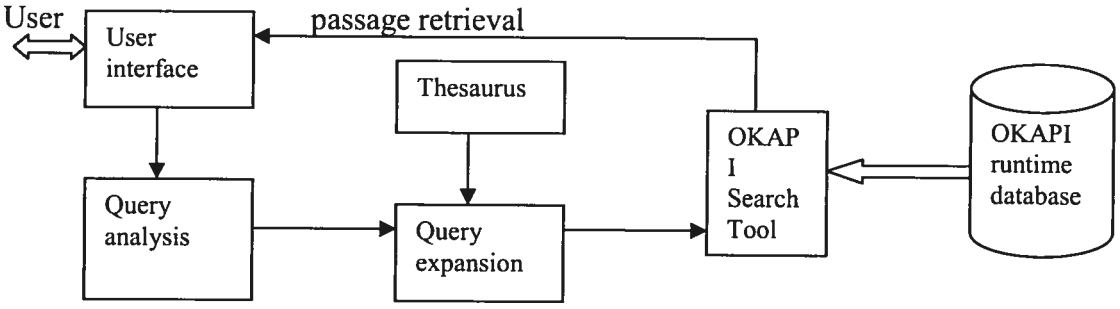


Figure 10. The system query expansion module diagram

When a query is submitted, the query analyzer module starts querying processing by filtering the stopwords from the query, and then the following steps are performed:

- Detect the occurrence of terms of the thesaurus in the query;
- Find synonyms of query terms in thesaurus;
- Add synonyms into the original query to generate a new query;
- Stem the words in the new query.

The expended new query is then submitted the OKAPI basic search system (BSS) through a call of BSS APIs. The retrieval results will be displayed on the user interface.

To find synonyms of a term, we first identify its ID in the thesaurus; then we retrieve all the terms (Ids) having "*RT*" or "*PT*" relationship with the original term. :

4.3.6 Web site crawler

This section presents our implementation of the crawler in the ERIC system. The crawler aims to collect document semi-automatically from Web.

The ERIC crawler starts with an URL address provided by the Webmaster, and tries to discover WebPages from the site, and to download them into the local database. The exploration is based on the links contained in the "known" pages – by known pages, we refer to the pages given by the Webmaster (the home page of a site) or those pages obtained in a previous cycle of exploration. When we encounter an URL in a known page, we compare it with a local URL database that contains the URLs the collected documents in order to avoid repeated information collection.

When an URL is new, it is explored only if it is stored on the same Web site as the URL originally submitted by the Webmaster. This restriction is made in order to not expand the exploration to a different site which may contain irrelevant documents for construction.

When a new URL is discovered, the referenced page is downloaded. The whole crawler is designed as an independent system with a user interface provided to the Webmaster. The crawler's architecture is illustrated in the following figure 11.

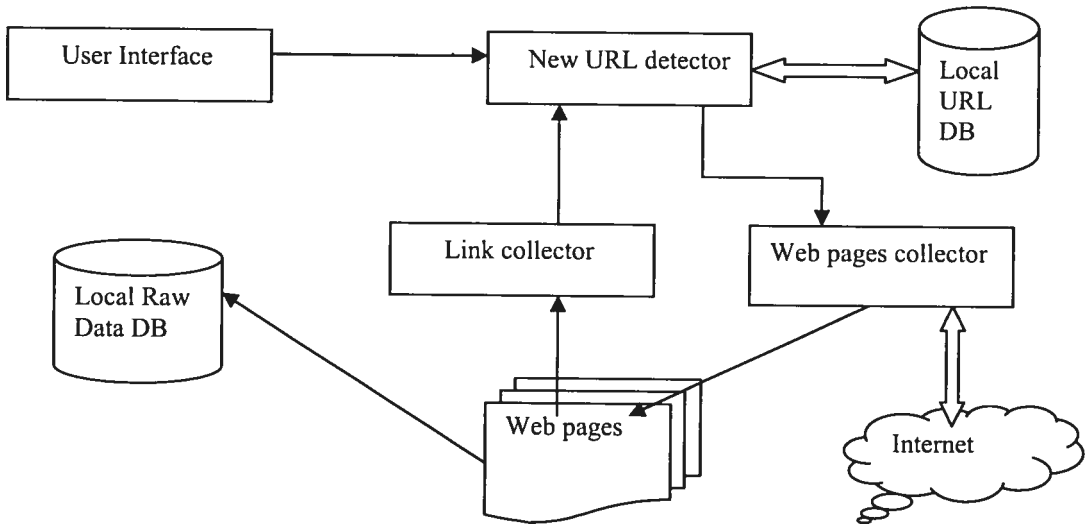


Figure 11. ERIC crawler module design

The details of the each module are presented below:

- User Interface

It provides interface between the crawler and the Webmaster. The Webmaster can submit the URL to start the crawling then monitor the downloaded results. The Webmaster can specify the depth of the exploration. When a depth (e.g. 2) is

specified, the crawler will stop following the links after the depth is reached. This depth limit is set to download only high level Web pages from marginally interesting sites. The Webmaster has the control on the depth. The interface also offers the options of "automatic crawling" and "monitored crawling". The automatic crawling will directly download documents automatically when the crawler discovers new Web pages. The monitored crawling is monitored by the Webmaster. It means the Webmaster has to approve the downloading of Web pages.

- New URL detector

This is a module which analyses the submitted URL to see if it has been downloaded. If the submitted URL is not in the local URL database, the crawler will start crawling its web pages according to this URL then add this URL into the local URL database.

- Web pages collector

It downloads the web pages corresponding to the URL provided.

Once the system downloaded documents from the Web, the next step is their indexing.

4.4 Integrated system architecture

As presented above, the ERIC system provides the following four main elements: indexing, retrieval, knowledge base, and crawler. The system is divided into two main parts: one performs retrieval; another performs data collection and indexing. Because both parts use the OKAPI for document indexing and retrieval, the following paragraphs will describe the integration of OKAPI in our system.

In ERIC system, the OKAPI system is a low level platform that supports ERIC. Each of the application levels of ERIC system is built upon the OKAPI platform, as illustrated in figure 12.

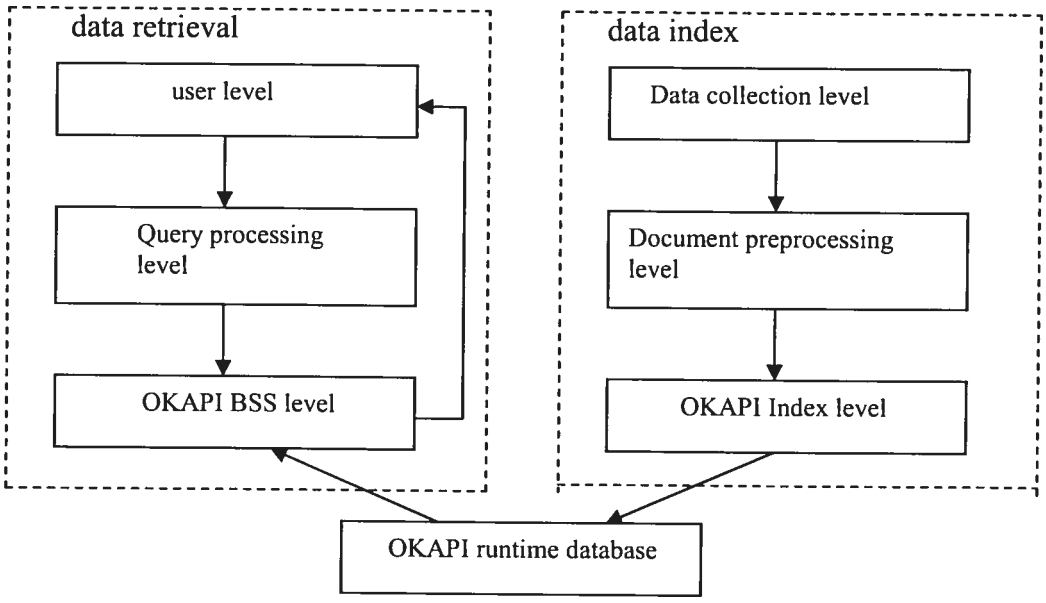


Figure 12. The ERIC system application level architecture

On indexing side, the collected documents go through its preprocessing and the extraction of compound terms. Then the OKAPI indexing function is used to construct the runtime database.

On the retrieval side, any query will pass user level (first level) and query processing application level (second level) to reach the OKAPI BSS level (third level). At the query processing level, the thesaurus is applied for extracting compound terms, and expanding the query by synonym terms. After the second level, the OKAPI BSS

will perform the retrieval operation with the new query. The retrieval results will be directly sent to the user interface.

The following figure 13 describes in more detail how OKAPI is integrated in ERIC system. It also presents how those OKAPI modules work within ERIC system.

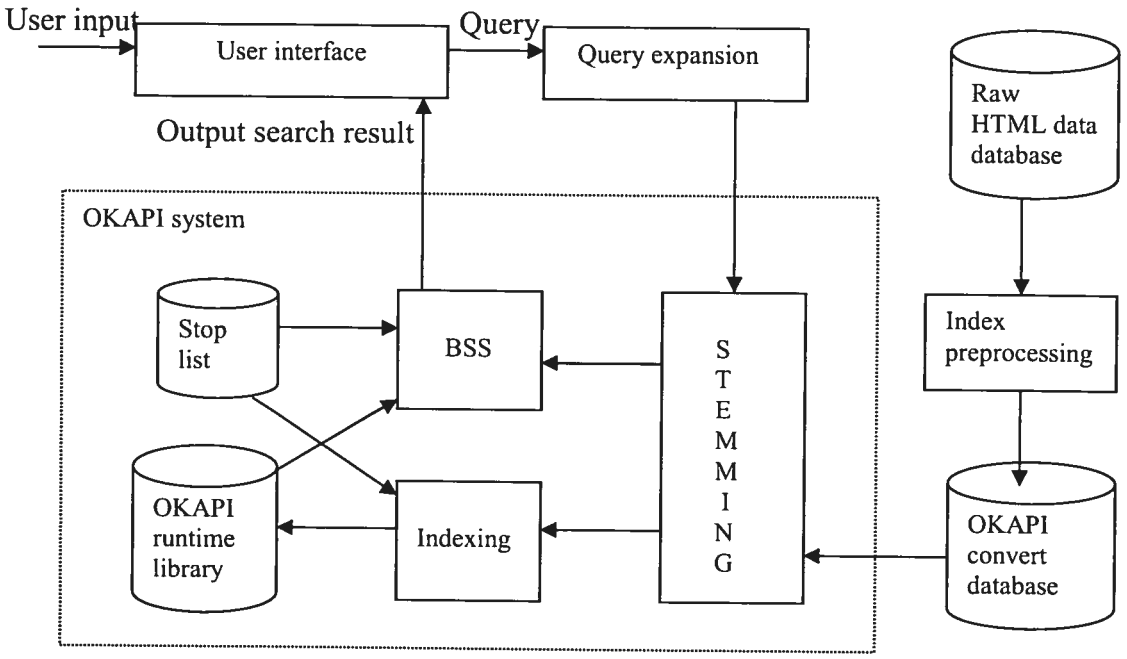


Figure 13. The integration OKAPI modules in ERIC system

To integrate each OKAPI modules in ERIC system, we use OKAPI's APIs. The whole architecture of the ERIC is shown in the following figure 14.

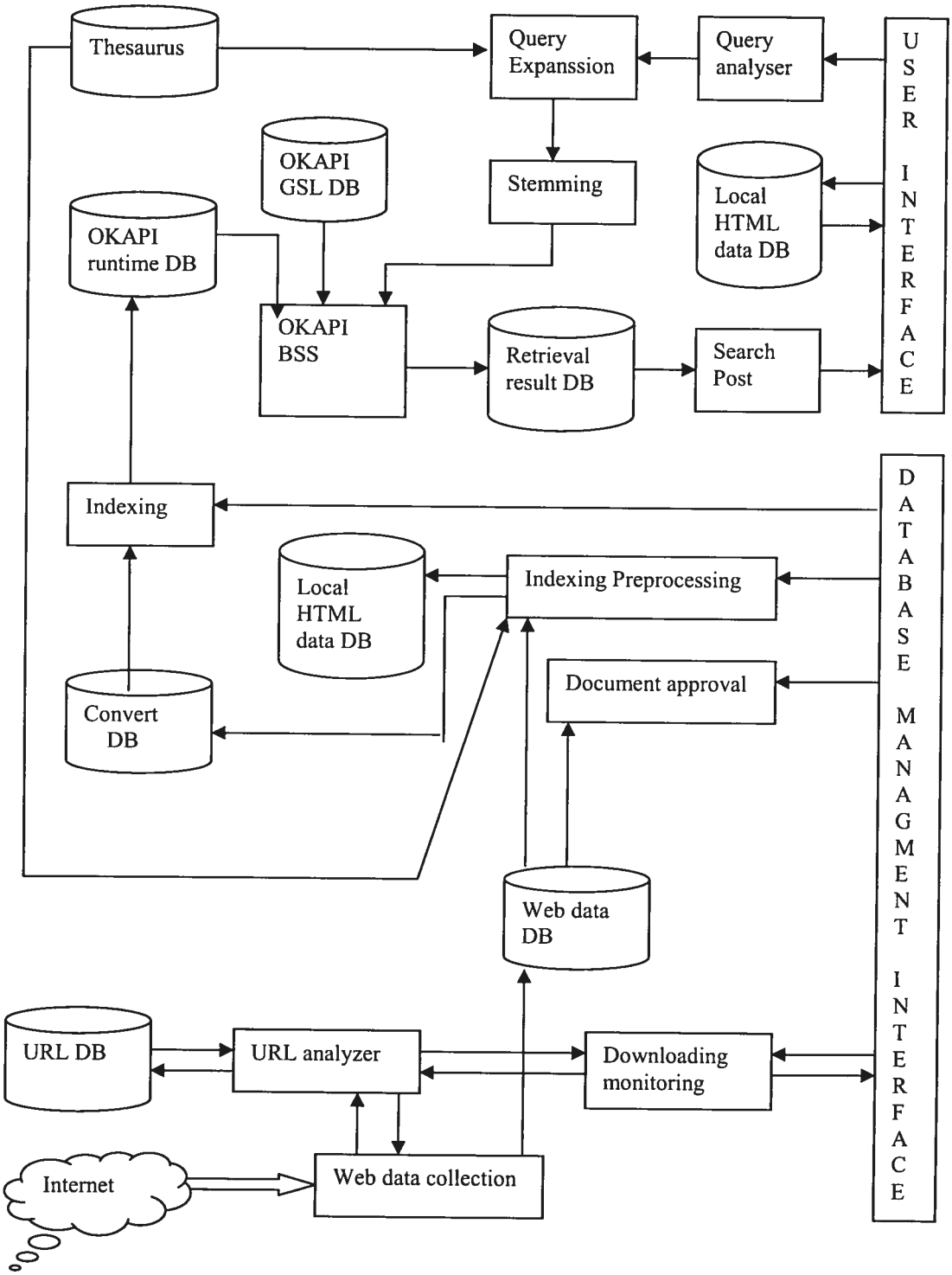


Figure 14. The system design modules

The components illustrated in figure 14 are briefly explained below:

1. In the data index part:

- Database index interface – it is an interface to accept the user commands and monitor data downloading from the Internet.
- Indexing – to index the passages and make the inverted file by calling the OKAPI index APIs.
- Indexing pre-processing – it is a processing of raw documents in order to recognize compound term, cut passages in each documents of collection, and set path between the cut passages and original documents.
- Convert DB – it is a function to convert raw data into OKAPI's format reading.
- Document approval– to display the downloaded raw documents for database administrator to check up.
- Web document DB – it holds the download web pages.
- URL acceptor – to check the connection and the existence of the submitted URL.
- URL analyzer – to analyzer and collect each URL link from the downloaded web pages, and then compare those collection with URL DB to detect new pages to crawl.
- URL DB – it holds all the downloaded web page's URLs.
- Web downloading – it downloads the web pages based on the URL provided.

2. In the data retrieval part:

- User interface – it is the interface between the user and the system which accepts requests from the user and returns the answer.

- Query analyzer – to analyze the submitted query and extract keyword from the query.
- Query expansion – to add the related terms of the original query.
- Stemming – removing suffixes from words.
- OKAPI GSL DB – it is the OKAPI system "stop words" database.
- OKAPI runtime DB – OKAPI inverted file handler (OKAPI runtime library).
- OKAPI BSS – OKAPI basic search system.
- Retrieval result DB – it is temporary file to hold the system's original retrieval results.
- Local HTML DB – it is a local resource to respond to the user. When user selects the passage of the retrieval list, the system has to display the document and point to the passage.
- Search post – to browse the user search results.

4.5 Related OKAPI APIs

Three parts of OKAPI are integrated in the ERIC system. They are stemming module, OKAPI indexing module, and OKAPI Basic Search System (BSS). The use of OKAPI in our system is made through calls to OKAPI's APIs. So let us describe the related APIs. Then we will describe how they are used in our system.

4.5.1 OKAPI indexing APIs

In order to index a set of documents for OKAPI to search and retrieve, the following steps have to be followed:

- Set up the database parameter files;
- Set up OKAPI exchange format file;
- Convert documents into OKAPI's database;
- OKAPI indexing.

We give some details on these steps in the following subsections.

4.5.1.1 Set up the database parameter files

Before performing indexing, we have to prepare a set of related parameters. These parameters are contained in following three files:

1. "db_name" file;
2. "db_name.field_types" file;
3. "db_name.search_groups" file.

The "*db_name*" file contains the name of the database that will be recognized by OKAPI system together with a set of parameters related to the database. The main parameters in "*db_name*" file are shown in the following table:

Entry in "db_name" file	Comment
name=<db_name>	database name
Bibsize=<in kilobytes>	An overestimate of the available space for the corresponding database.
real_bibsize=<size of corresponding volume>	This value is filled in by convert_runtime program.
Display_name=<info_display_name>	The field to be displayed by the BSS "info databases" command.
nr=<number of records>	The number of records in the database.
nf=<number of fields per record>	The number of fields in the database.
db_type=<database_type>	Type of the data which may be text or others
ix_stem=<pathname of index files + prefix>	This parameter specifies the path to find the documents to be indexed
ix_volsize=<size in MB>	Space available (MB) for index file.
ix_type=<index_type>	8 for <db_type>=text; 9 for others.

Table 3. Some parameters to be specified in "bd_name" file

When OKAPI indexer starts to index the documents in the local database, this "*db_name*" file will be first read. When OKAPI BSS starts to search and retrieve documents, this file will also be called first so that the APIs can select the corresponding database to search.

The second file "*db_name.field_types*" contains parameters to set up the field types in each document. In ERIC search database, we will also use it for the passage type. The OKAPI indexer will read this file before indexing to recognize different fields. Below is some of the parameter of field types:

Field Type	Comment	Field Type	Comment
NAMES	Presents a person or company mane etc. in this field	TITLE	Indicate a title in this field
MAIN_TITLE	Indicate a main title in this field	SUBTITLE	Indicate a subtitle in this field
TEXT	Indicate if this field contains text document.	SH	Indicate a subject heading in this filed
LITERAL_NC	It is lowercase in this field	NUMBERS	Indicate some numbers in this field

Table 4. Some parameters in <db_name>.field_type file

In the ERIC system, the document passage field types are specified as LITERAL_NC and TEXT i.e. the raw documents are lowercase texts.

The third file "*db_name.search_groups*" contains one entry per index. Each entry consists of several fields. Some of them are shown in the following table.

Field	Comment
<index_name> [|<index_name>]	The name of the index file.
<stem function name>	There are three options in OKAPI: wstem (weak stemming), sstem (strength stemming), and nostem (no stemming)
<GSL filename>	stoplist filename and path. "GSL" means General Stop-List.

Table 5. Some parameters in <db_name>.search_group file

This file indicates the stopword file name (GSL) and the word stemmer to be used. When the name of stopword file is indicated, OKAPI indexer will filter those stopwords before indexing. When a term stemmer is selected, OKAPI indexer will stem each term before indexing. In ERIC system, we use "*sstem*" provided by OKAPI. It is a stemmer in OKAPI .

4.5.1.2 Defining OKAPI exchange format

Documents have to be formatted into OKAPI exchange format in which records and fields in records are clearly separated. Passage segmentation is actually performed during the transformation of documents into this format. In ERIC system, this is done by inserting a special character "*0x1E*" to mark the beginning of each passage (record). In each passage, each field is also separated by a special character "*0x1D*". Once these markers are inserted, OKAPI can index each passage and each field according those special character marks. In ERIC system, we use one text field in each record (passage).

4.5.1.3 Converting into OKAPI database

Once the data collection has been processed into exchange format, it has been converted into OKAPI runtime format which is usable by OKAPI. This is the indexing process because making convert database is a condition to use the OKAPI indexing APIs and the convert commands are in the OKAPI APIs. An example of using OKAPI APIs for converting is as follows:

convert_runtime –c <BSS_PARMPATH> <db_name> <exchange format file>

the "*convert_runtime*" is one command in OKAPI indexing APIs. This call means to start the process for converting the original database <*db-name*> into the exchange format in <*exchange format file*>. The <*BSS-PARMPATH*> indicates the path of the BSS parameter files.

4.5.1.4 OKAPI Indexing

OKAPI provides two API commands to perform the final indexing – *ix1* and *ix*f. The *ix1* is for each index specified in "*db_name.search_group*" file. It reads each field specified by the appropriate indexing parameter, splits it into "indexing units" determined by the field type, and extracts or generates keys from it in accordance with the specified indexing method, stemming function and stopword file. When term extraction has finished, those extracted index terms will be input to the indexing program ixf for final index process. The *ixf* is final index production program. Below is the way of using the *ix1* and *ixf* APIs:

ix1 –c <BSS_PARMPATH> <db_name> <index_no> | ixf –c <BSS_PARMPATH> <db_name> <index_no>

where: index_no is the rang in the database parameter.

Once OKAPI runtime library is ready, the OKAPI BSS can perform document retrieval.

4.5.1.5 Using stemming API for query processing

The OKAPI stemming API command is included in the BSS APIs. However, it is an independent function that the user can use in a separate task. In ERIC system, the

indexing stemming will be automatically called in indexing with "ix1" program. In addition, in query processing, the OKAPI stemming API is also used to stem user query and expanded new query. The call to this function is as follows:

stem t=<term>

The result of this call is the stemmed term.

4.5.2 BSS API commands

The BSS APIs are used to access the OKAPI basic search system. A search process goes through the following steps:

1. it first chooses OKAPI runtime local search database which it will use to search document;
2. it parses the query into a suitable list of terms;
3. it matches each query term with search database index.
4. it weights the retrieval results; and
5. finally, it returns the ranked search results to the user.

To run OKAPI BSS, five BSS APIs are used to perform these five steps.

- `choose`

  It chooses a local database that the system follows to retrieve information from this search database. If there are several independent databases in the system, OKAPI has to use this API command to decide which database to use.

- `parse`

  It parses the query into individual terms in a form suitable for looking up in the database.

- `find`

  This API aims to find the records containing one particular terms. It is a lookup into the inverted file. For example the argument "*f t=construct*" would match the records that contain "*construct*" term.

- `weight`

  This API determines a weight for each of the query term, according to the number of documents the "find" command returns for it. In ERIC, the calculation of this weight is made according to BM25 formula, which is described in chapter 3, to rank the retrieval results for the user.

- `show`

  It displays discovered documents to the user.

4.6 Workflows

In this section, we will show the workflow of the indexing and the retrieval process. Figure 15 below is flowchart of ERIC system database index processing. It displays the involved actions in building the local OKAPI runtime library to search and retrieval. It starts from data crawling with a submitted URL and ends with the generation of the OKAPI runtime library. Figure 16 shows the workflow of the ERIC search process. It presents the involved processes and resources in query search and retrieval.

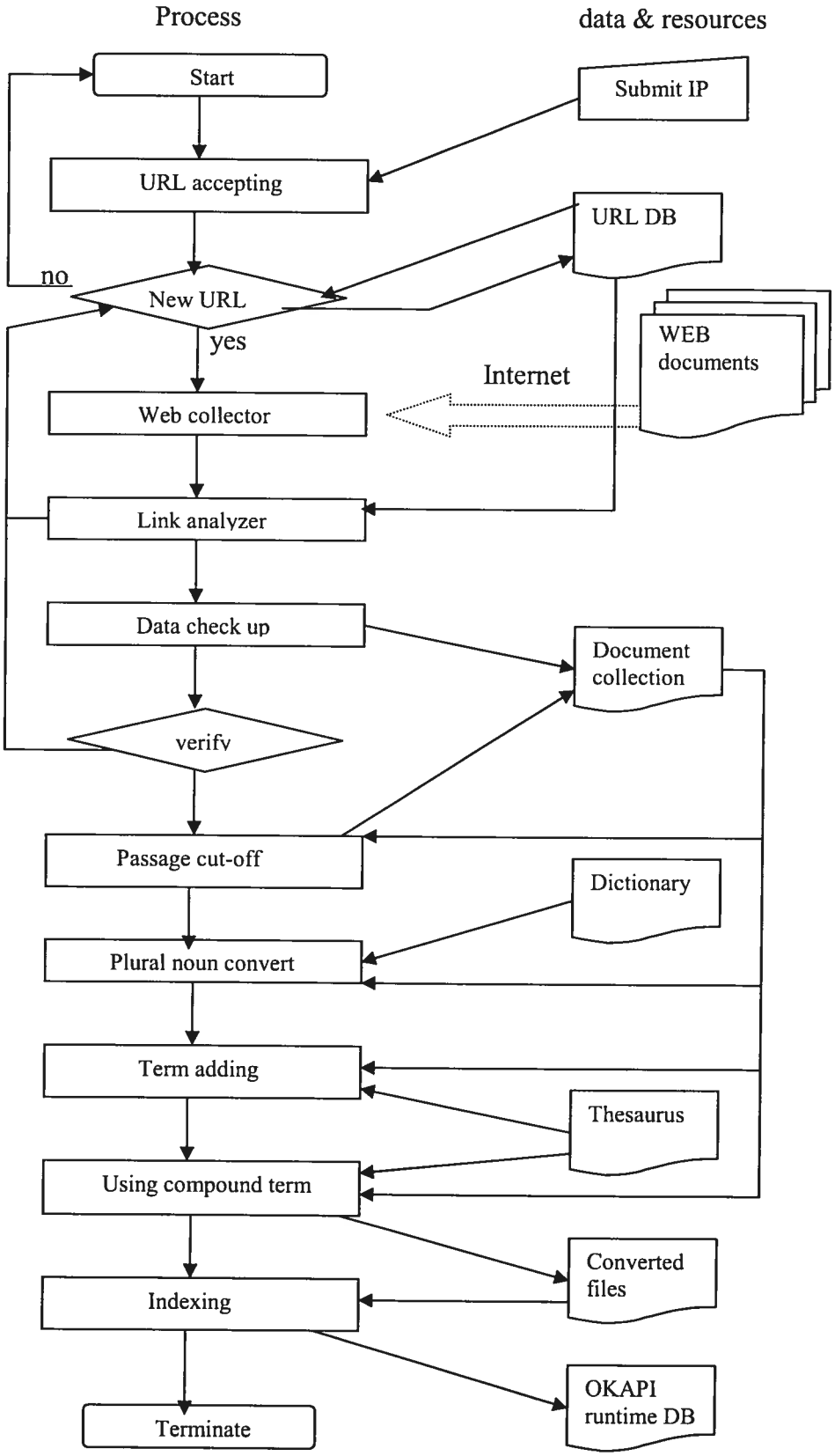


Figure 15. The workflow of the document collection and indexing process

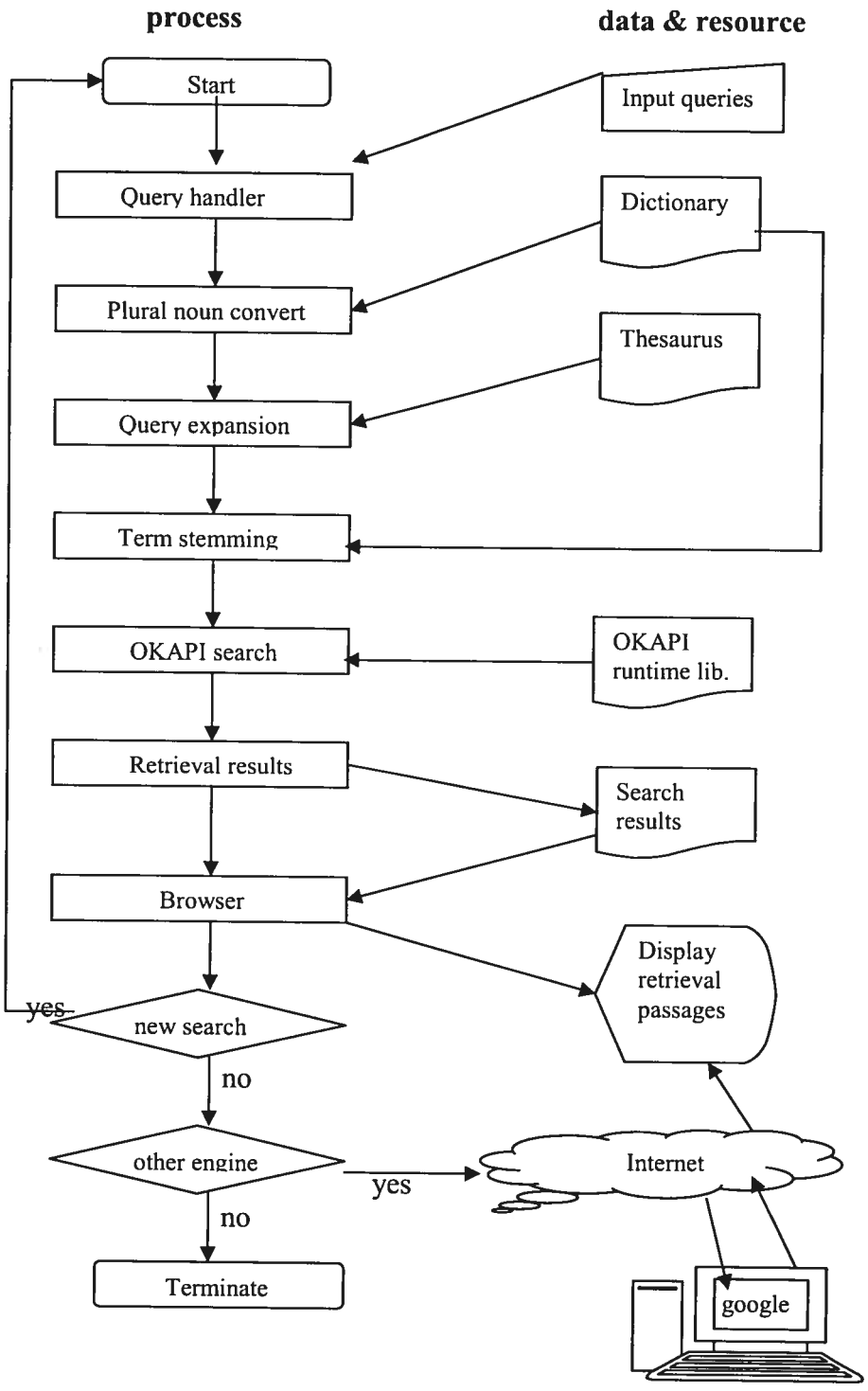


Figure 16. The workflow search process

4.7 The system interface

We will show some snapshots of the ERIC interfaces to illustrate the functionalities implemented. The first user interface is shown in figure 17 below.

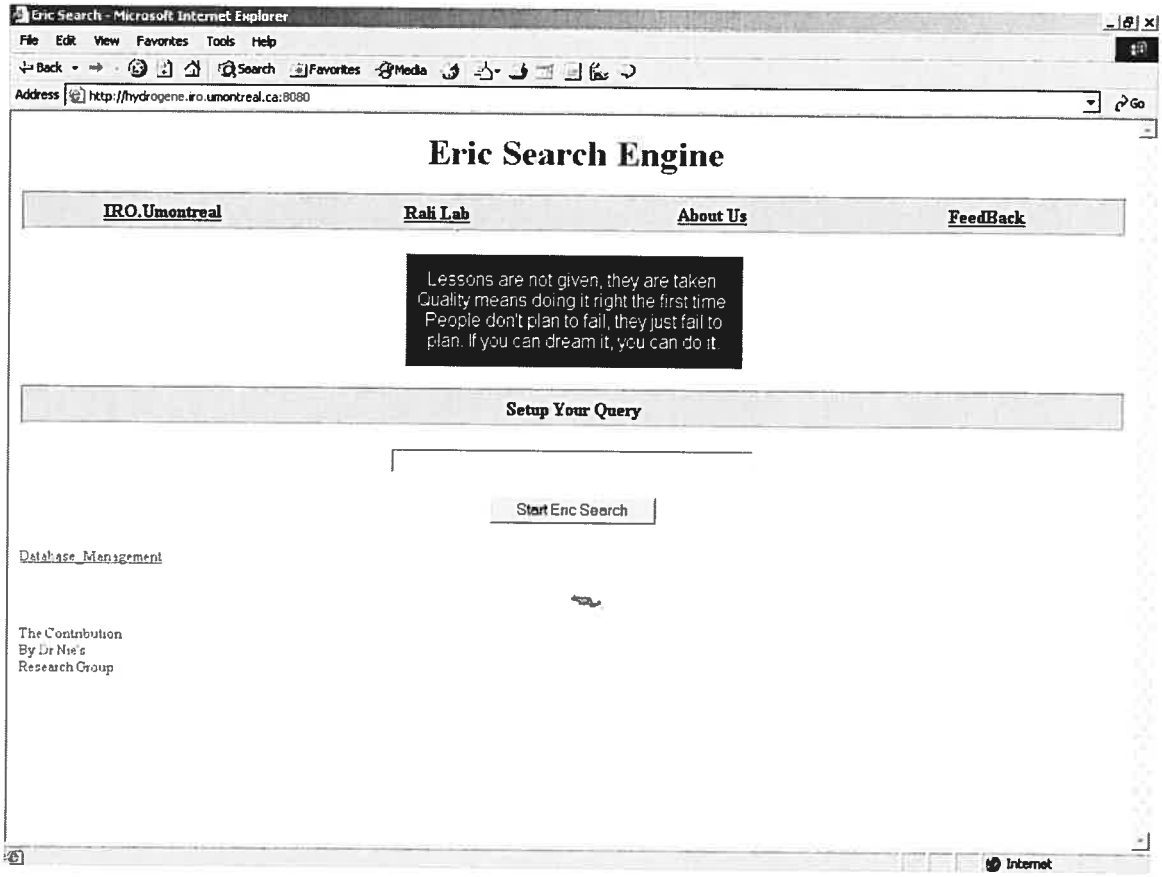


Figure 17. The system query acceptance interface

This user interface allows the user to input a query. It is a simple online Web interface programmed in java script. When the user submits a query in the query-submitting box, the system starts to retrieve passages for it.

This interface is implemented by using Perl, C and HTML languages. A Web server – Apache Web server is also integrated in ERIC for handling multi-processing on Internet. When a query is input by the user, it is transmitted to ERIC through Web server, and Perl program.

When search results are produced, the system interface will show the results in the display window (figure 18). For each retrieved passage, the matched terms are highlighted. The user can choose to read the complete document (view document), or read the Web page at its original site (e.g. http://www.arc.ca/irc/cbd/cbdauth-e.html).

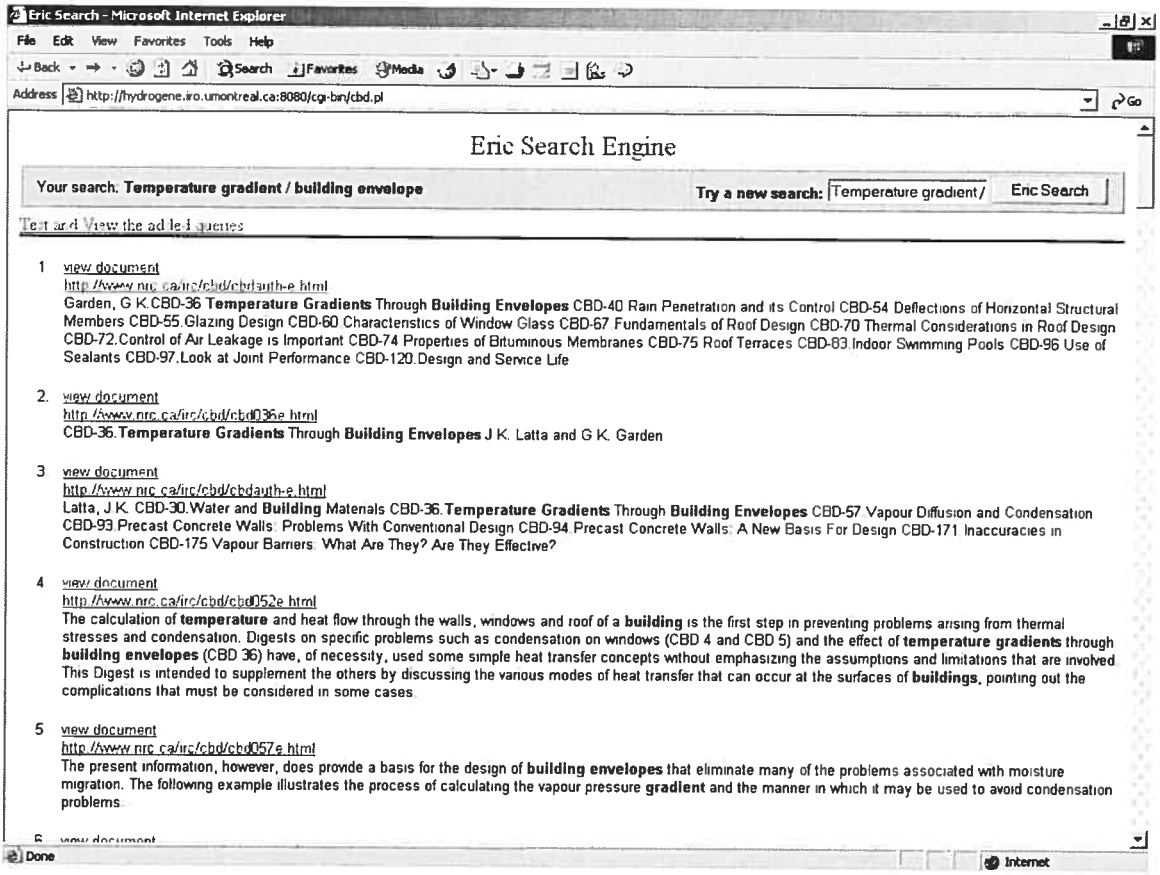


Figure 18. The search result interface

Figure 19 below shows the interface for managing document database (downloading and indexing). This interface includes the following main functions as:

- search function to discover documents with a user query,
- local URL database display function to display the collected URLs, as well as the corresponding document,
- index function to perform OKAPI indexing of all the passages.
- Local disk clear up function to interrupt all of the crawling process and clear up the downloaded files from local machine disk.
- Crawler to submit URL in the URL submitted box to start system crawling.

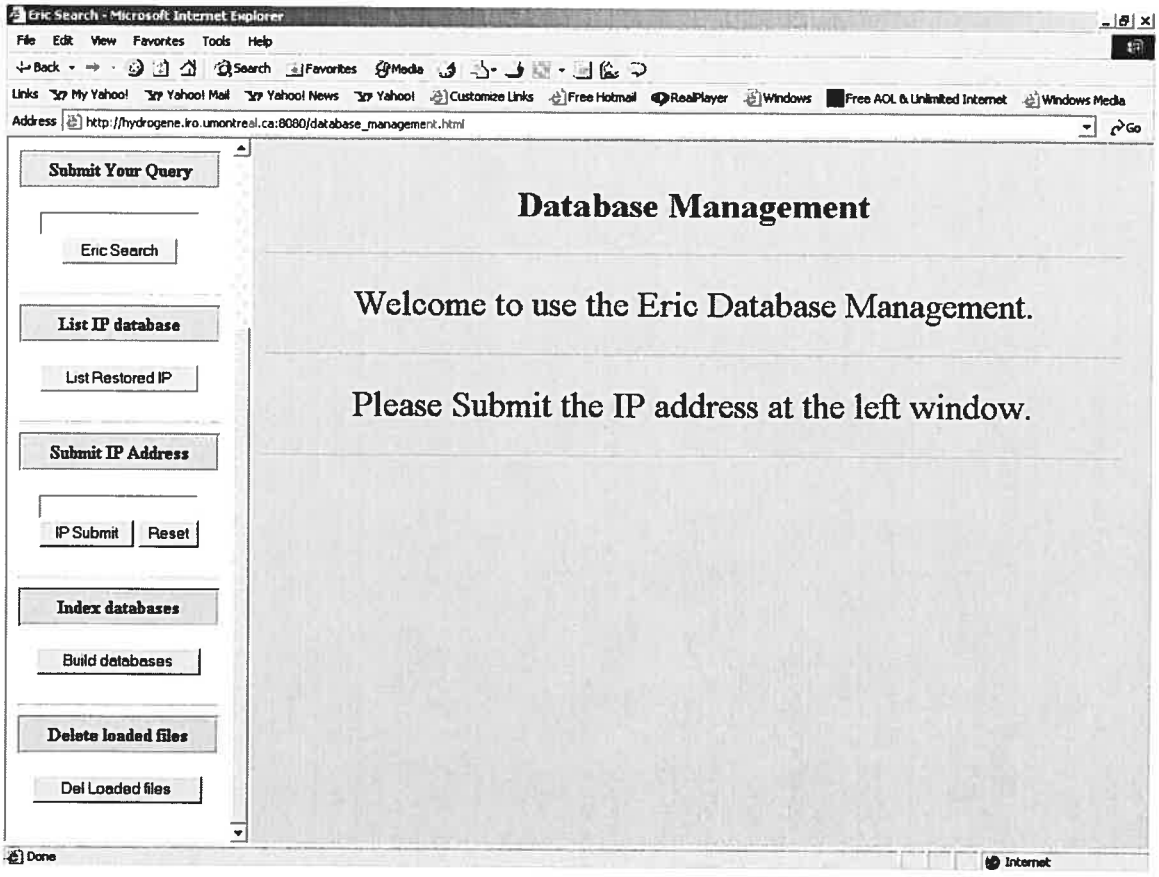


Figure 19. The system database management-processing interface

Figure 20 below shows an example of the crawling process. The system administrator or Webmaster can monitor the crawling processing by check up the loaded data through this interface. The interface includes four buttons on the bottom of this page allowing downloading the document, ignoring the document, crawling the Web site, and starting to index the local database immediately.

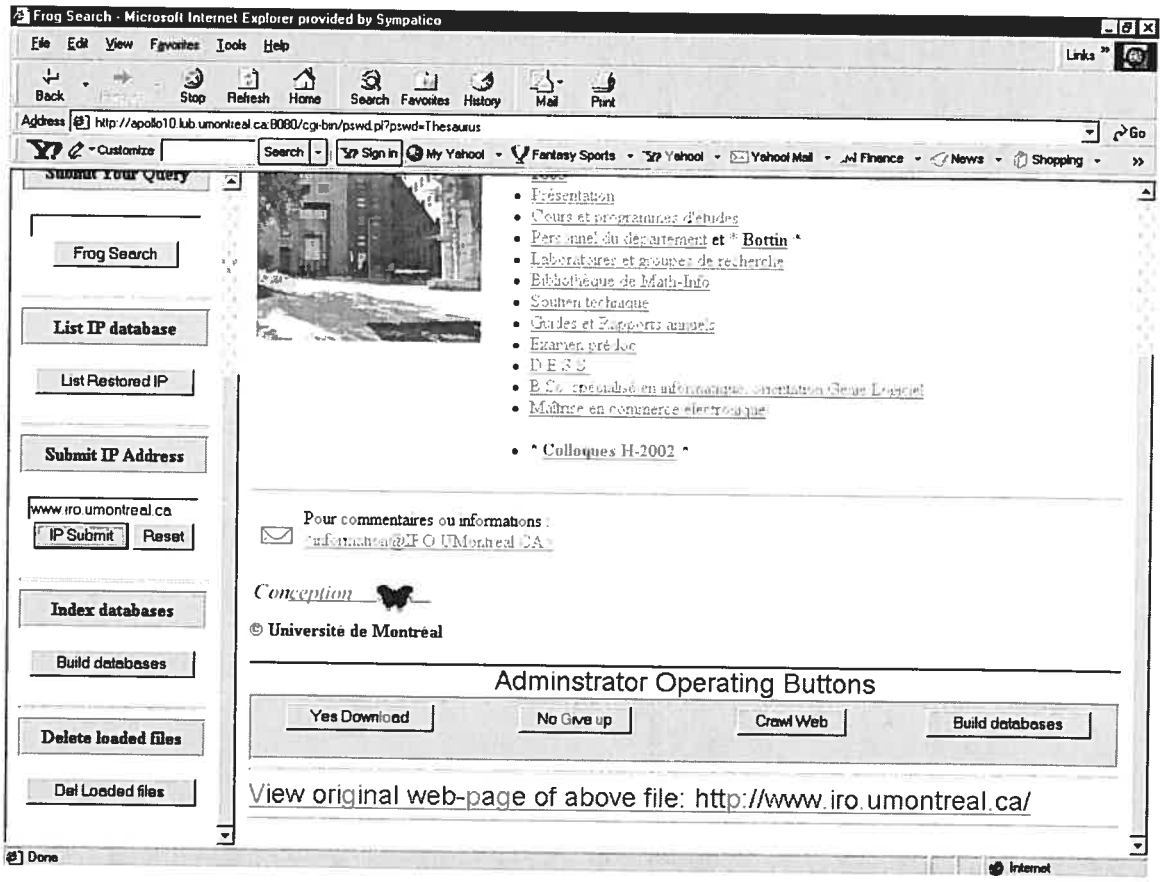


Figure 20. The crawled web page displaying

4.8 Summary

In this chapter, we described the implementation of ERIC system. It went through the system requirement, system architecture and design, and system

implementation of the ERIC system what we have implemented is the first step for a new domain-specific search engine in the construction sector.

Our system is based on the OKAPI platform by calling its APIs. Through adding some new functions (i.e. compound term and query expansion), the ERIC system performs domain-specific search upon OKAPI functionalities. In this chapter, we also described how ERIC integrated the OKAPI's functionalities in its own system and how ERIC calls the OKAPI's APIs.

In addition, a simple crawler is implemented and presented in this chapter. It is a simple Web crawling tool which provides a way to create and extend the local search database by adding new documents.

Because ERIC is an online search engine on Internet, the Web interfaces for ERIC have been created for user to use the ERIC through the Internet. These interfaces support multi-users.

However, OKAPI is not an open source software. We cannot change any source code of it. This imposed some limitations on our implementation. For example, we have to recognize compound terms through a separate pre-processing and cannot modify OKAPI's source code to add this function in the OKAPI. In addition, OKAPI only operates under Linux/Unix. Therefore. ERIC cannot run in the windows environment.

Chapter 5. Experiments

In this chapter, we will present our evaluation of the ERIC system for IR in the construction sector. The whole system is functional. We have been able to construct a small collection of documents on which user can search. The system is ready to extend the document collection to a much large one. In this chapter, we will examine three main aspects:

1. The construction of a test collection;
2. The impact of compound terms on the retrieval performance;
3. The impact of query expansion on the retrieval performance.

5.1 Constructing document collection

There is no existing test collection available in the construction sector. Therefore, we have to construct our test collection. This includes: a set of documents (document collection), a set of queries, and the relevance judgments for each of there queries (i.e. we know all the desired relevant documents for each query). In the following sections, we will describe how these are constructed.

5.1.1 Canadian building digest — first document collection

Canadian building digest (CBD) is the main document resource for our tests. This collection contains high quality documents describing problems in construction.

It is also one of the most consulted document collection in Canada. We use this collection as our primary test collection.

The Canada building digest is published online. The web site URL is http://www.nrc.ca/irc/cbd/ in the Canadian National Research Centre (NRC). It is an authority resource for knowledge and information in the building industry. The Web site contains around 250 documents. The characteristics of those building digest documents are described below:

First, the published CBD is in HTML format. There are some links in the documents. A CBD document generally has between 5 and 15 hyperlinks (more than eight links on average) and most of them are local (that is, they point to pages on their own web site). There also is a document index web page, whose URL address is http://www.nrc.ca/irc/cbd/cbd-e.html, It displays the sorted titles of the whole CBD documents, and some links to some additional document.

Second, the documents do not only contain text on the document pages but also images. Sometimes, the image is more important in building industry professional document because it can present more clearly in intuition. Typically, those image links are also local links.

Finally, the CBD documents have a reasonable length. On average, each document contains 2900 words. The paragraphs are around 250 words long. So it is reasonable to use paragraphs as passages in ERIC. There are in total 9786 paragraphs in the CBD collection.

5.1.2 Crawling and collection

The documents of CBD have been downloaded using our crawling and downloading tool. The procedure of collecting document by the crawler follows the following steps:

1. Submit URL

The index page of the Canada building digest (i.e. www.nrc.ca/irc/cbd-e.html) is submitted as the entrance point.

2. Crawling

The crawler follows the links to explore the linked documents (the 250 documents). To avoid the crawler exploring too deeply in a Web site, we can set a depth limit. For the CBD, we set it at 1, i.e. we only explore the directly linked documents from the submitted home page.

During the crawling process of CBD, we monitor the downloading of documents so that the test collection only contains high-level domain-specific documents. The downloaded 250 documents are stored in 250 files in their original HTML format. They constitute our document collection.

5.2 Test queries

To be significant, the test queries should be at least 25 or 50. In TREC, 50 queries are used each year. For our experiments we also use 50 queries. We asked an experts in construction - a PHD student in architecture to determine 50 questions that

people in construction can likely ask to the CBD collection. Below are some of the questions, as well as their correct answer.

1. According to the National Building Code, what is the snow load that has to be considered for roofs in Canada?

"Answer: Since for many roofs in Canada the snow load is the greatest load that has to be sustained, its design value takes on great importance with regard to the safety and economy of these structures. Climatic variations across Canada are reflected by the corresponding variations in design snow loads, such as those used in the National Building Code of Canada which vary from 30 to 60 psf in the more populated areas. Even greater loads are required in other areas. The snow loads for parts of Northern Quebec for example reach 90 psf, and loads of up to 100 and 200 psf are found in some of the mountain areas of British Columbia."

2. Temperature gradient / building envelope

"Answer: Determination of the thermal gradient throughout a building element that separates two environments that have different properties is the first step toward designing problem free walls. Information pertaining to thermal bridges, psychrometry, moisture migration, rain penetration and differential air pressures is also necessary for optimum design. Perfection in buildings is not readily achieved and the quest for it is often hampered, for financial reasons."

3. What is corrosion?

"Answer: Corrosion of metals is an electrochemical process in which the deteriorating area of the metal is the anode, the positively charged electrode of the galvanic cell. Positive potential of the metal indicates corrosion activity, i.e., the metal in this region is converting from the metallic to the ionic state. The value of the potential depends on the tendency of the metal to go into solution and, based on the concentration of ions around the electrode, is a good measure of the corrosion that has taken place."

The complete list of the queries is given in an appendix.

For all the queries, only one correct answer is identified for each.

5.3 Metrics

The performance of IR systems is usually measured by recall and precision. Recall measures the ability of the system to retrieve all useful documents, while precision measures the ability to retrieve only the useful documents. The recall and precision are defined as follows:

$$\text{Recall} = \frac{\text{Number of relevant items retrieved}}{\text{Total number of relevant items in the collection}} \quad (1)$$

$$\text{Precision} = \frac{\text{Number of relevant items retrieved}}{\text{Total number of relevant items retrieved}} \quad (2)$$

The measures of precision and recall vary according to the length of the result list considered. The longer the list, the lower the precision, and the higher the recall. For our experiments, the purpose is not so much to know the absolute values of precision and recall. Rather, we want to compare the performance of different methods. Therefore, we can fix a length of the result list and compare the methods on the same basis.

In our testing, we consider the top 50 retrieved documents. It is shown in [Buckley00] that using the top n (n = 50, 100) retrieved documents are a valid evaluation method in IR.

As the number of relevant answers is very limited, which is only one, it is not reasonable to use the traditional precision-recall curve to measure the retrieval

performance. The precision-recall curve would be very flat. This situation is similar to that in the Question Answering (QA) track of TREC, for which only a few relevant answers exist. In the QA track, a score is defined as follows:

$$\text{Score}_i = \frac{1}{n_i} \Sigma_j \frac{1}{\text{Rank}_{ij}} \qquad (3)$$

$$\text{and Average-Score} = \frac{1}{N} \Sigma_i \text{Score}_i \qquad (4)$$

where:

$Score_i$ is the score for a query (i-th query);

n_i is the number of relevant answers found in the top retrieved results for query i;

Rank_{ij} is the rank of a relevant answer for query i in the retrieved results;

N is the number of test queries.

For example, given the following retrieved results (limited to the top 5), in which the elements marked with * are relevant:

answer 1	
answer 2	*
answer 3	
answer 4	*
answer 5	

Table 6. Example of related answer

The score for this query is:

$$Score_i = \frac{1}{2}\left(\frac{1}{2} + \frac{1}{4}\right) = \frac{3}{8} = 0.375$$

The average score used for QA is more appropriate for us. We will use this measure in our experiments.

5.4 Using basic OKAPI system

The OKAPI system is used as a reference system. Its performance is used to compare with the other enhanced methods. By OKAPI system, we mean to use OKAPI's indexing and retrieval process without any modification. These methods use single words as indexes. A standard stop-list is provided with the OKAPI package. The stemming process is also a standard one which cuts suffix of the words. The weighting is BM25 – described in chapter 2. OKAPI is set in such a way that it can perform passage retrieval (here, passage is paragraph). The top 50 retrieved results

(paragraphs) are compared with the standard answers. The following table shows the results obtained with OKAPI.

Rank	Number queries	Score
1	14	1
2	6	1/2 (0.5)
3	4	1/3 (0.333)
4	3	1/4 (0.25)
5	1	1/5 (0.2)
7	2	1/7 (0.143)
8	1	1/8 (0.125)
13	2	1/13 (0.077)
14	1	1/14 (0.071)
15	1	1/15 (0.067)
16	1	1/16 (0.063)
22	1	1/22 (0.045)
24	1	1/24 (0.042)
47	1	1/47 (0.021)
0	11	0
Average Score = 40.3%		

Table 7. Basic OKAPI system test data

Table 7 shows the ranks of the correct answers in the list, for different numbers of questions, as well as the scores for these questions. From table 7, we see that 14

questions have their correct answers at the first place in the retrieval list and 11 queries do not have their correct answers among the top 50 answers. 25 other questions have the results in the top of 50 answers but not at the first place. Finally, the average score for all the 50 queries is 40.3%. This is not a bad result in comparison with what we obtain in general in IR which is usually around 30% in average precision. However, this performance measure should not be considered in absolute because it is strongly influenced by the test collection. In our case, an important factor is that our documents are domain-specific, contributing in increasing the retrieval effectiveness. Nevertheless, the above measure can serve as a reference to compare with the other methods we developed, which we will evaluate in the following sections.

5.5 Integrating compound terms

One specific processing in our project is the utilization of the compound terms in indexing and searching. To identify compound terms, a thesaurus on construction is used. For a query or document passage, the system scans it for compound terms. If a compound term in the thesaurus is found, the system will add this compound term into the passage or the query. The retrieval result using compound terms is shown in table 8 below:

Rank	Number queries	Score
1	14	1
2	6	1/2 (0.5)
3	6	1/3 (0.333)
4	3	1/4 (0.25)
7	2	1/7 (0.143)
13	3	1/13 (0.077)
14	2	1/14 (0.071)
15	2	1/15 (0.067)
22	1	1/22 (0.045)
24	1	1/24 (0.042)
26	1	1/26 (0.038)
36	1	1/36 (0.027)
0	8	0
Average Score = 41.3%		

Table 8. Compound term utility test data

In table 8 above, we see that the same 4 queries have retrieval results at the first place with precision 100%, as in the OKAPI test result. The number of the queries for which no relevant passage appears in the top 50 results is reduced to 8, compared to 11 in our reference test with OKAPI. The global average score using compound terms is 41.3%. This is slightly higher than OKAPI average precision rate (40.3%).

From the result above, it is clearly indicated that there is an improvement by adding the compound term in indexing and searching. However, this is not an important improvement. The main reason is that the test queries contain compound

terms. We can only identify compound terms in 25 of the 50 queries. For these queries, we observed an increase of average score from 40.3% to 41.3%.

Compound terms have to be used together with single words. When we use compound terms together with single words, it is important to assign reasonable weights to compound terms V.S. single keywords. If compound terms are assigned a too high weight, the top ranked search results usually are dominated by compound terms. This may over stress the importance of compound terms and neglect single words. As a result, the recall may be decreased. In order to solve this problem, we have to reduce the weights of the compound terms at a comparable level to the average weight single terms. Through several tests, we found that cutting the original weights of compound terms to half will be a good solution. The result shown in Table 8 corresponds to the setting.

5.7 Using synonyms for query expansion

Query expansion aims to add synonym words to original query. The synonyms are determined with the thesaurus. When a synonym term is found, it is appended to the query. Below is an example of the expanded query.

```
Query: According to the National Building Code, what is the snow load
that has to be considered for roofs in Canada?
```

Single terms after stemming are:

```
t=according, t=national, t=building, t=code, t=snow, t=load,
t=consider, t=roof, t=canada
```

Expended terms are:

build $\Rightarrow_{expended}$ civil_engineering_work, construction, facility, landscape_feature, land_development, construction.

code $\Rightarrow_{expended}$ labour_law, lighting_technology, leisure_economics, electricity.

snow $\Rightarrow_{expended}$ snowstorm, snowstorm, attic.

load $\Rightarrow_{expended}$ load_bearing_element

roof $\Rightarrow_{expended}$ roof_framing, roofing_product, structural_member, waterproofing_work, wood.

The new query with expanded terms is:

<expanded terms> civil_engineering_work, construction, facility, landscape_feature, land_development, construction, labour_law, lighting_technology, leisure_economics, electricity, snowstorm, snowstorm, attic, load_bearing_element, roof_framing, roofing_product, structural_member, waterproofing_work, wood.

The test result with query expansion is shown in Table 9 below.

Rank	Number queries	Score
1	14	1
2	6	1/2 (0.5)
3	6	1/3 (0.333)
4	2	1/4 (0.25)
5	2	1/5 (0.2)
6	1	1/6 (0.167)
7	2	1/7 (0.143)
10	1	1/10 (0.1)
12	1	1/12 (0.083)
13	3	1/13 (0.076)
14	2	1/14 (0.071)
20	1	1/20 (0.05)
24	1	1/24 (0.042)
26	1	1/26 (0.038)
36	1	1/36 (0.028)
0	6	0
Average Score = 42.1 %		

Table 9. Query expansion integration test data

In table 9, we see that the number of queries whose relevant answer is out of the top 50 results is reduced to 6. The global average score with query expansion and compound term performance is 42.1%. It is about 4.49% ((new_average_score – old_average_score) / old_average_score) better than the original OKAPI result, and

2% better than the run with compound terms only. It shows that compound terms and query expansion are suitable techniques for our application area.

However, as for compound terms, we also have to assign reasonable weights to the expansion terms. As we can see from the example, many of the expansion terms are not strongly related to the query. So assigning a too strong weight to them would reduce the retrieval effectiveness because more irrelevant results would be retrieved. On the other hand, the expansion terms can also help enhance the original query. Therefore. An appropriate weight has to be assigned to an expansion term. In the above table, the results are obtained with an 80% reduction of the original weights of the expanded terms.

5.8 Conclusion

In this chapter, we tested and compared several indexing and retrieval methods with a test collection. The methods compared are the basic OKAPI method, OKAPI with compound terms, and OKAPI with query expansion and compound terms. We obtained an average score of 40.3% for the basic OKAPI, 41.3% for OKAPI with compound terms, and 42.1% for OKAPI with compound terms and query expansion. The results indicate that adding compound terms and doing query expansion is beneficial for our application.

However, there are some problems with compound terms and query expansion.

- Compound terms:

  1. Many queries do not contain compound terms;

2. The thesaurus does not contain all the compound terms. For example, the term "*Canada Building Digest*" would be a compound term, but in our thesaurus, this compound term is not included. A possible solution is to use an automatic analysis to extract new compound term.

- Query expansion:

  The thesaurus does not always suggest strongly related expansion terms. As we can see in example we showed earlier for query expansion. A possible solution is to find automatically related terms. This approach has been tested in and interesting results have been obtained.

Chapter 6. Conclusions and Future work

To solve the problems for professional users who look for precise information in the construction sector, we started our project on domain-specific information retrieval. Our project aims to build a new domain-specific search engine – ERIC system for the construction sector. The final goal of our project is to perform question answering (QA). Our study described in this thesis is the first step of it, i.e. passage retrieval.

To construct this new domain-specific search engine, we chose OKAPI to be basic search tool in our system. However, OKAPI does not perform a domain-specific search. To solve this problem, we focused on three aspects in our study: Web document collection, indexing and retrieval with OKAPI (passage retrieval), and enhancement with a thesaurus for compound term and query expansion.

- Web document collection

In our project, a simple crawler is used to download Web documents. It starts from manually select related Web sites, and automatic downloading documents from these sites. The reason of manual selection is that we only want high quality domain-specific documents to be downloaded and stored in our local search database.

- Indexing and passage retrieval with OKAPI

OKAPI is a classical IR system with probabilistic model. The reason to choose OKAPI is its high effectiveness, which has proven in previous IR experiments. However, it is not an open source system. We use OKAPI by calling its APIs. In

ERIC, we use OKAPI BSS for basic retrieval and OKAPI index module for document indexing.

In order to enhance document indexing with our approaches, we have to carry out a document preprocessing to segment passages of the documents, and to add compound terms in the passages.

- Enhancement with a thesaurus for compound terms and query expansion

In order to enhance OKAPI basic retrieval, we created a query preprocessing for identifying compound terms in the query as well as determine the related terms to add in the query expansion. The generated new query will contain compound terms if it has, and some synonym words of the original query through the query expansion. Because more words have been added in the new query, the OKAPI could exploit more candidates rather than limited candidates with original query retrieval.

The above three aspects have been integrated into the ERIC system. To test effectiveness of ERIC with our approaches, Chapter 5 presented our experiences with ERIC and comparison between ERIC and the original OKAPI system. We obtained an improvement of 2% by integrating compound terms over the basic OKAPI method, and an improvement of 4.9% by integrating both compound terms and query expansion in ERIC. It shows that compound terms and query expansion are useful techniques for our application area. In addition, the thesaurus is also a necessary tool for us to identify compound terms and synonym words for query expansion.

However, there are still some problems remaining in the system. For example, the thesaurus presents a source of problem. To be a knowledge base, we want the thesaurus to cover all the specialized terms, which are related to the construction

sector. However, the thesaurus we used is far from complete: many important terms (either single words or compound terms are missing from the thesaurus. As a consequence, the effect of compound term detection and query expansion is limited. To solve this problem, it is possible to use statistical methods to find compound terms and expansion terms. Statistical methods can be used as complementary tool to the thesaurus. It is a future work on the ERIC system.

In addition, the OKAPI is not open source software. We do not have the possibility to change the codes of OKAPI for our need. For example, we would like to integrate compound term detection as pat of the OKAPI functions, but it is not possible. On the other hand, we would also like to change OKAPI common search tool to be more domain-specific search by incorporating specialized terms in the retrieval process. For example, we may want to change the weighting scheme for specialized terms. A possible way to do it would to change the OKAPI BMxx (best match) such as BM25 to the one below:

BM25: $w = w_{(bm25)} + w_{(t,thesaurus)}$

However, this has not been possible. We had to artificially decrease by half the weights assigned by OKAPI to compound terms. This is not the most reasonable weighting method. So the modification of OKAPI's weighting is another interesting future work.

Despite all these limitations, we have been able to construct a basic passage retrieval system which integrates some domain knowledge. This system is used as the basic retrieval system for further developments on question answering.

References

[Agichtein01] Agichtein, Eugene, Lawrence, Steve, Gravano, Luis, Learning Search Engine Specific Query Transformations For Question Answering, *Tenth International World Wide Web Conference*, 2001, pp: 169 – 178, http:// citeseer.nj.nec.com/agichtein01learning.html

[Buckley00] Buckley, Chris, Voorhees, Ellen M., Evaluating Evaluation Measure Stability, *proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2000, pp: 33 – 40, http://citeseer.nj.nec.com/buckley00evaluating.html

[Ballerini96] Ballerini, J.P., Buche, M., Domenig, l, R., Knaus, D., Mateev, B., Mittendorf, E., Schauble, P., Sheridan, P., Wechsler, M., SPIDER Retrieval System at TREC-5, *proceedings of TREC – 5*, 1996, National Institute of Standards and Technology (NIST) special publication pp: 271 – 229, http://trec.nist.gov/pubs/trec5/t5_proceedings.html

[Cormack97] Cormack, G.V., Palmer C.R., Passage-based refinement (multi-text experiments for TREC-6), *proceedings of TREC – 6*, 1997, National Institute of Standards and Technology (NIST) special publication pp: 303 – 321, http://trec.nist.gov/pubs/trec6/t6_proceedings.html

[Callan94] Callan, James P., Passage-Level Evidence in Document Retrieval, *Proceedings of the Seventeenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 1994, pp: 302 – 309, http://www.ai.mit.edu/people/jimmylin/papers/Callan94.pdf

[DeLoupy98] De Loupy, C., Bellot, P., EI-Beze, M., Marteau, P.-F., Query Expansion and Classification of Retrieved Documents, *proceedings of TREC – 7*, 1998, National Institute of Standards and Technology (NIST) special publication pp: 443 – 451, http://trec.nist.gov/pubs/trec7/t7_proceedings.html

[Fuhr92] Fuhr, Norbert, Probabilistic Models in Information Retrieval, *The Computer Journal (volume 35)*, 1992, pp: 243 – 255, http://citeseer.nj.nec.com/fuhr92 probabilistic.html

[Girill96] Girill, T. R., Luk, Clement H., Fuzzy Matching as a Retrieval-Enabling Technique for Digital Libraries, *1996 ASIS Mid-Year meeting*, 1996, http:// www.asis.org/midyear-96/girillpaper.html

[Ghani00] Ghani, Rayid, Jones, Rosie, Mladenic, Dunja, Nigam, Kamal, Slattery, Sean, Data Mining on Symbolic Knowledge Extracted from the WEB, *Proceedings of the Sixth International Conference on Knowledge Discovery and Data Mining (KDD-2000) Workshop on Text Mining*, 2000, pp: 29 – 36 http://citeseer.nj.nec.com/ghani00data.html

[Hammer94] Hammer, Sebastian, Favaro, John, Sistemi, Intecs, Z39.50 and the World Wide Web, *the magazine of the digital library research*, march 1994, ISSN 1082 – 9873, http://www.dlib.org/dlib/march96/03contents.html

[Holmes98] Holmes, D., McCabe, C., Grossman, D., Chowdhury, A., Frieder, O., Use of Query Concepts and Information Extraction to Improve Information Retrieval Effectiveness, *proceedings of TREC – 7*, 1998, National Institute of Standards and Technology (NIST), special publication pp: 399 – 409, http://trec.nist.gov/pubs/trec7/t7_proceedings.html

[Ittycheriah01] Ittycheriah, A., Franz, M., Roukos, S., IBM's Statistical Question Answering System – TREC-10, *proceedings of the TREC – 10*, 2001, National Institute of Standards and Technology (NIST), special publication pp: 258 – 265, http://trec.nist.gov/pubs/trec10/t10_proceedings.html

[Ishikawa97] Ishikawa, K., Satoh, K., Okumura, A., Query Term Expansion based on Paragraphs of the Relevant Documents, *proceedings of TREC – 6*, 1997, National Institute of Standards and Technology (NIST), special publication pp: 577 – 585, http: //trec.nist.gov/pubs/trec6/t6_proceedings.html

[Katz97] Katz, Boris, From Sentence Processing to Information Access on the World Wide Web, *Proceedings of the AAAI Spring Symposium on Natural Language Processing for the World Wide Web*, 1997, http://www.ai.mit.edu/people/boris /webaccess/

[Kleinberg98] Kleinberg, Jon, Authoritative Sources in a Hyperlinked Environment, *proceedings of the 9th ACM-SIAM Symposium on Discrete Algorithms,* 1998, http://www.cs.cornell.edu/home/kleinber/auth.pdf

[Llopis01] Llopis, Ferando, y, Vicedo, Jose, Luis, IR-n a passage retrieval system from University of Alicante, *CLEF 2001 Cross-Language System Evaluation Campaign,* 2001, http://www.ercim.org/publication/ws-proceedings/CLEF2/ llopis.pdf

[McDonald97] McDonald, J., Ogden, W., Foltz, P., interactive information retrieval using term relationship networks, *proceedings of TREC 6*, 1997, National Institute of Standards and Technology (NIST), special publication pp: 379 – 385, http://trec.nist.gov/pubs/trec6/papers/index.alpha.html

[Mitra98] Mitra, Mandar, Singhal, Amit, Buckley, Chris, Improving Automatic Query Expansion, *Research and Development in Information Retrieval*, 1998, pp: 206 – 214, http://citeseer.nj.nec.com/121460.html

[Nie01] Nie, J.Y., A general Logical approach to inferential Information retrieval, *Encyclopedia of Computer Science and Technology, eds. Kent E. A. and Williams J.G., Vol. 44*, 2001, pp : 203 – 226, http://www.iro.umontreal.ca/~nie/publication.html

[Nie02] Nie, J.Y., Jin, Fuman, Integrating Logical Operators in Query Expansion in VSM, *Workshop on Mathematical/Formal Methods in Information Retrieval, 25th ACM-SIGIR,* 2002, http://www.iro.umontreal.ca/~nie/publication.html

[Nie02] Nie, Jian-Yun, Dufort, Jean-François, Combining Words and Compound Terms for Monolingual and Cross-Language Information Retrieval, *Information 2002*, 2002, http://www.iro.umontreal.ca/~nie/publication.html

[Nie99] Nie, J.Y., Simard, M., Isabelle, P., Durand, R., Cross-Language Information Retrieval based on Parallel Texts and Automatic Mining of Parallel Texts in the Web, *22nd ACM-SIGIR*, Berkeley, 1999, pp. 74-81, http://www.iro.umontreal.ca/~nie/publication.html

[Nie98] Nie, J. Y., TREC-7 CLIR using a Probabilistic Translation Model, *proceedings of TREC 7*, 1998, National Institute of Standards and Technology (NIST), special publication page: pp: 547 – 555, http://trec.nist.gov/pubs/trec7/t7_proceedings.html

[Nie96] Nie, J.Y., Brisebois, M., An inferential approach to information retrieval and its implementation using a manual thesaurus, *Artificial Intelligence Review*, 10: 409-439, 1996, http://www.iro.umontreal.ca/~nie/publication.html

[Newby98] Newby, Gregory B., An Information Access Model with a Unified Approach to Data Type, Retrieval Mechanism and Information Need, ASIS 98, 1998, http://www.ils.unc.edu/~gbnewby/papers/asis98-theory-final.html

[Page98] Page, Lawrence, Brin, Sergey, Motwani, Rajeev, Winograd, Terry, The PageRank Citation Ranking: Bringing Order to the Web, *Stanford Digital Library Technologies Project*, 1998, http://citeseer.nj.nec.com/page98pagerank.html

[Piwowarski00] Piwowarski Benjamin, Learning in Infromation Retrieval: a Probabilistic Differential Approach, *proceedings of the BCS-IRSG, 22nd Annual Colloquium on Information Retrieval Research*, 2000, http://citeseer.nj.nec.com/piwowarski00learning.html

[Ricardo99] Ricardo Baeza-Yates, Berthier Ribeiro-Neto, Modern Information Retrieval, *Addison Wesley Longman Publishing Co. Inc.*, 1999

[Robertson92] Robertson Stephen E., Walker Stephen, Micheline Hancock-Beaulieu, Gull Aarron, Lau Marianna, OKAPI at TREC, *proceedings of TREC – 1*, 1992, National Institute of Standards and Technology (NIST), special publication pp: 21-31, http://trec.nist.gov/pubs/trec1/t1_proceedings.html

[Robertson93] Robertson, S E, Jones, S Walker, S, Hancock-beaulieu, M M, Gatford, M, OKAPI at TREC-2, *proceedings of TREC – 2*, National Institute of

Standards and Technology (NIST), special publication pp: 21-35, http://trec.nist.gov/pubs/trec2/t2_proceedings.html

[Robertson94] Robertson, S E, Walker, S, Jones, S, Hancock-beaulieu, M M, Gatford, M, OKAPI at TREC-3, *proceedings of TREC – 4*, 1994, National Institute of Standards and Technology (NIST), special publication pp: 109-127, http://trec.nist.gov/pubs/trec3/t3_proceedings.html

[Robertson95] Robertson S E, Walker S, Beaulieu M M, Gatford M, Payne A, OKAPI at TREC-4, *proceedings of TREC – 4*, 1995, National Institute of Standards and Technology (NIST), special publication pp: 73 – 108, http://trec.nist.gov/pubs/trec4/t4_proceedings.html

[Srihari99] Srihari, Rohini, Li, Wei, Information Extraction Supported Question Answering, *proceedings of TREC – 8*, 1999, National Institute of Standards and Technology (NIST), special publication pp: 185 – 197, http://trec.nist.gov/pubs/trec8/t8_proceedings.html

[Salton83] Salton Gerard, Michael J. McGill, *Introduction to modern Information Retrieval*, McGraw-Hill Book Company, New York, 1983

[Savoy99] Savoy J., Picard J., Report on the TREC-8 Experiment: Searching on the Web and in Distributed Collections, *proceedings of TREC – 8*, 1999, National Institute of Standards and Technology (NIST), special publication pp: 229 – 241, http://citeseer.nj.nec.com/364516.html

Appendix: Test Questions

1. According to the National Building Code, what is the snow load that has to be considered for roofs in Canada?

2. What is corrosion?

3. Where can I find the thermal resistance of building materials?

4. Design of exit signs

5. Temperature gradient / building envelope

6. Soil / permeability

7. Issues about the location of drains

8. Aspects related with the chemical resistance of pipes.

9. Is it possible to use glass-fibre reinforced cement in structural elements?

10. Which norms of the building code have to be considered in the renovation of an existing building?

11. Where can I find information about the influence of radon in human health?

12. I am looking for information regarding the use of computers in the industry

13. How to prevent wood from decaying under the influence of water?

14. Drainage / erosion / filters

15 Design considerations for roofs in cold regions

16. Research about shadow angles and solar shading in façades

17. Doors insulation

18. Which trees should I use to reduce water demand in the soil?

19. Reducing rain penetration in prefabricated walls

20. In soil testing, what does swelling mean?

21. Glazing design / rain penetration / construction details

22. What are the silts?

23. Considering sound transmission, what are the specifications recommended for a party wall in between two apartments?

24. The selection of the type of foundation

25. What is the stack effect in buildings?

26. How to establish the air supply rate in buildings?

27. Established dimensions for the access of wheelchairs

28. Comment construire un abri d'hiver au Canada?

28b. How to build a winter shelter for construction sites in Canada?

29. What is polymer concrete?

30. Does the National Building Code accept the construction of wood frame foundations?

31. How to reduce the corrosion of the reinforcing steel in garages?

32. Where can I find a map of Canada with the seismic risk regions?

33. The address of the Standards Council of Canada

34. The Building Research Library

35. What causes air pressure differences in windows?

36. Waterproofing the Basement

37. How to find information about solar radiation on walls for the particular case of Canada?

38. What is the loss of noise transmission recommended for adjacent rooms in apartments?

39. What is efflorescence?

40. The effect of color in the temperature of roofs

41. What is the recommended temperature for the water of an indoor pool?

42. Volume changes in concrete structures due to moisture changes

43. The Canadian Building Digests

44. Do the clear urethanes perform well to the influence of UV radiation?

45. What is the recommended mortar for laying reclaimed bricks?

46. Rock formations and pyrite

47. Central control and monitoring systems

48. Does it exist a relation between condensation and roof forms?

49. What is the maximum tolerable noise level accepted in apartments?

50. Degree of comfort of ground-level winds