BUILDING AN EFFICIENT INDEXING FOR CRAWLING THE WEBSITE WITH AN EFFICIENT SPIDER

G. Al-Gaphari, Ph.D.
Department of Computer Science
University of Sana'a, Yemen
email: drghalebh@yahoo.com

Abstract- With the present effort, we propose to investigate results of applying the Right-Truncated Index-Based Web Search Engine in order to determine its usefulness for storing and retrieving Arabic documents. The Right-Truncated Index-Based Web Search Engine, being a program for reading any set of Arabic documents accepts a query, and then processes both the documents and the query. Thus, it selects (predicts) those documents most relevant to the query which has been inserted. The program encompasses both a morphological component and a mathematical one. The morphological component allows the researcher to run either a stemming algorithm or a right-truncated algorithm. The chief advantage of the stemming algorithm is that it uses the least possible amount of storage for indexing by mapping the inflected and derived terms into a single, indexed-stem word. On the other hand, the right-truncated algorithm reduces the amount of storage to a lesser degree, but increases the probability of retrieving relevant (user-favorable) documents, compared to the stemming algorithm. One of the purposes of our investigation is to compare the efficiency of these two indexing mechanisms. The mathematical component of the algorithm accepts the output of the right truncation algorithm, and then employs both term-frequency and inverse document-frequency (TF-IDF) in order to establish the relative importance of each document, respective to the terms of the query. This paper also describes building a simple search engine based on a crawler or a spider. The clawer which indexes different types of documents is an algorithm to crawl the file systems from specified folder. A basic design and object model was developed to support single search word results as well as multiple search words results. It is capable of finding data to index by following (tracing) web links rather than searching directory listings in the file system. In this process files are downloaded through HTTP and HTML pages parsed in order to obtain more links without getting into a recursive loop. Also, this paper discusses how to improve indexing mechanism efficiency using a right truncated stemmer in terms of Arabic documents processing.

Keywords: Truncation, Indexing Efficiency, Web Search Engine, Spiders and Crawler.

THE PROBLEM OF ARABIC INDEXING AND QUERY-RETRIEVAL

Having become the occasion for a large amount of research as well as the development of a number of software applications, the task of indexing and retrieving information as

initiated by a user-issued query has emerged as a formidable, omnipresent difficulty. Yet, because of the universality of the problem, both researchers and developers will be obliged to continue their efforts until an optimal solution is finally achieved. In fact, many web-search engines have been developed for accommodating English and Arabic language topics. Nevertheless, those web-search engines designed for fielding English language queries might not be ideally effective in, also taking on those in Arabic language. The English method is to store multitudinous variants of a given word (i.e., its inflections and derivations) and when retrieving, to return a range of independently divergent variants, irrespective of their relative usefulness to the user's query. Such a procedure necessitates vast tracts of superfluous storage-space as well as a reasonably extended time-lapse for retrieval of information. Otherwise, web-search engines have thus far evolved for the purpose of accommodating Arabic-language indices based individually on a word, a stem, a root, or some combination of all three. Search engines which make use of root or stem-indexing offer a substantial reduction in the amount of storage required (thus favoring a reduction in the size of the indexing system). Those methods which employ some combination of stem- or root-indexing, as well as word-indexing, are able to find use exclusively for special purposes (e.g., the testing of indexing effectiveness, the science of linguistics, etc.). Conversely, they have not improved Arabic query-efficiency within the domain of retrieval of information, owing to the multitude of additional and irrelevant documents that are retrieved. For example, assume that a certain user-query will contain a word المذهبية, meaning "sectarianism", and that the method of indexing is root-retrieval: in this case, the search engine may map the above word into, e.g., the sector ذهب, which would signify the various concepts of "to go"; "to remove"; "to think"; "to gild" (i.e., "to apply a gold finish"). Hence, many of the documents retrieved might conceivably not be of any interest at all to the user and, moreover, such an indexing system might not be able to determine within an acceptable time-frame at which the documents thus retrieved are relevant. The main objective of the present experiment is to design and implement a Right Truncated Index algorithm that would affect a trade-off between storage requirements and the efficiency of query-retrieval. The algorithm is comprised of two components. The first is a morphological component (right truncated and stemmer algorithms): the right truncated algorithm maps out the respective variants of Arabic words, breaking them down into their original forms. The expedient is removing their right prefixes and retaining the words' remaining structure intact, for use as indices. On the other hand, the stemmer algorithm maps out the respective variants of Arabic words, breaks them down into their word-stems through the expedient of removing their prefixes and suffixes then returns the words' remaining structure, for use as indices.

The second is a mathematical component, known as the "vector space model",

which classifies documents in accordance with the user-query. Drawing upon this model as a basis, both the documents involved and the user-query are considered to be a pair of vectors in vector space. They are capable of encompassing as many dimensions as there are distinct indexing terms which remain after corpus-preprocessing. On such a foundation, the cosine-similarity which exists between query and document would be computed.

PRIOR RESEARCH

In this section, a survey will be presented of previous research conducted in the matter of Arabic stemming and Arabic information-retrieval, accompanied by a brief and informal explanation of the vector-space model.

- ARABIC STEMMER AND INFORMATION-RETRIEFAL

Most of the stemming algorithms thus far developed for Arabic information-retrieval have been based strictly upon root- or stem-models which provide users with divergent methods for locating morphological variants of the queries submitted; and these were able to gauge the effectiveness of Information-retrieval only on a basis of the linguistic relevancy concept, in place of learning-algorithms or a synthesis of both [9]. The essential researches conducted in this area were the following:

-- IMPROVING STEMMING FOR ARABIC INFORMATION-RETRIEVAL ALGORITHM

Having developed a number of stemmers, the researchers concluded that the light stemmer was more effective for cross-language retrieval than was the locating of the root for each word [3]. They also concluded that stemming was more agreeable[19], within the domain of Arabic information-retrieval, owing to the inflected nature of the language [12]. This conclusion may not hold for the stem or root, since there are other stems or roots in the assortment of word-senses which, being substantially unrelated, are user-irrelevant [15].

-- BUILDING A SHALLOW ARABIC MORPHOLOGICAL ANALYSER IN ONE DAY

K. Daruish [6] has devised a morphological analyzer which is grounded upon automatically-derived rules and statistics.

-- ON BI-DIRECTIONAL ENGLISH-ARABIC SEARCH

The researchers have devised three different methods for query-translation, making use of a bilingual dictionary for Arabic-English [13], Cross-Language Information Retrieval (CLIR) [2]. The researchers concluded that Machine Translation (MT)

systems were unable to achieve high-quality translation unless the system was usefully endowed with some level of semantic representation [1].

-- EMPIRICAL STUDIES IN STRATEGIES FOR ARABIC RETRIEVAL

This investigation shows similarity with the previous study wherein the authors presented an evaluation of some strategies for monolingual and cross-lingual Arabic retrieval [17]. The researchers formulated the conclusion that a strategy of spelling normalization improved retrieval performance [10]. However, such a normalization strategy confronts the absence of accommodation, in Arabic, of morphological rules, leading to the introduction of undesirable new letters which in turn, might interfere with, or change the concept of the word [16].

-- MORPHOLOGICAL ANALYSES AND GENERATION IN THE INSTANCE OF ARABIC DIALECTS

The researchers introduced a morphological analyzer and generator (MAGEAD) which addressed the necessity for processing the morphology of dialects [11]. They asserted that MAGEAD was equipped with morphological data and rules for MSA and certain Arabic dialects [8].

-- TOWORDS A UNIVERSAL DICTIONARY FOR INFORMATION RETRIEVAL APPLICATIONS

This paper reveals an investigation of the trade-off between the cost of tracking and the size of vocabulary for any given metric of term selection [18].

- SURVEY OF VECTOR-SPACE MODEL

Term-weights are definable simply as the multiplication of two vectors; term-frequency and inverse document-frequency [5].

-- TERM-FREEQUENCY (tf)

Term-frequency (tf (t)) is equal to the number of times (f(t)). Term t appears in document d; it can be normalized by dividing by the frequency of the most common term appearing in the document. Mathematically, it can be expressed as:

(1) $tf(t) = f(t)/\max\{f(t1) ..... f(tn)\}$

-- INVERSE DOCUMENT FREQUENCY (idf)

It can be expressed as: (2) $idf(t)=\log (D/df(t))$ Where df(t) is the number of documents containing a term t, and D is the number of documents in the corpus [14].

-- TERM-WEIGHTS (tf-idf)

The term-weights can be expressed as: (3) tf-idf=tf(t)*idf(t) Where idf(t)= log (number of document/document-frequency) [4].

-- INNER PRODUCT (COSINE-SIMILARITY)

The cosine-similarity between vectors for the document d and the query q can be calculated as the vector inner-product, as:

$$(4)\ \mathrm{sim}(d,q) = \frac{\vec{d_j} \cdot \vec{q}}{\left|\vec{d_j}\right| \cdot \left|\vec{q}\right|} = \frac{\sum_{i=1}^{t} (w_{ij} \cdot w_{iq})}{\sqrt{\sum_{i=1}^{t} w_{ij}^{2} \cdot \sum_{i=1}^{t} w_{iq}^{2}}}$$

where w (i, j) is the weight of term i in document j, and w (i, q) is the weight of term i in the query [7]. The above algorithm will be combined with a morphological stemmer (right truncation algorithm), and will be implemented and tested in the next section.

PROPOSED EXPERIMENT

- INTRODUCTION

Stemming algorithms determine morphologically similar indexing and search terms. They can be employed to provide end-users with different ways of locating morphological variants for inserted queries, and for reducing the size of index files. This experiment concentrates upon five main objectives:

- The design and implementation of stemming algorithms for reducing the size of the index file.
- The determination of which parts of Arabic words are prefixes or combinations of prefixes.
- The demonstration of the extreme importance of Arabic prefixes, and how they are commonly used in the various Arabic scripts.
- The comparison between the right-truncation and stemming algorithms performances.
- The combination of the new index file (the right truncated algorithm output index file) with a mathematical model called a “vector-space model” for computing cosine-similarity between the documents and the end-user query, and thus improvement of information-retrieval effectiveness.
- Building Spider.

- SETTING UP OF THE EXPERIMENT AND TRAINING IN THE STEMMING ALGORITHM

The present algorithms are right truncated and stemmer algorithms. The right truncated algorithm accepts Arabic terms, removes different prefixes from each term in the corpus, and returns the residue of each term as an index term. On the other hand, the stemmer algorithm has the ability to remove the prefixes and suffixes and to return the word stem as an index term. For purposes of training in and testing of this algorithm, the TREC-2001 Arabic corpus is used. Known also by the name AFP_ARB corpus, it contains more than 10,000 documents consisting of over fifteen million words, or over fifteen gigabytes in UTF encoding, as provided by the Linguistic Data Consortium. The amount of data mentioned had been input according to the right truncated algorithm which processes the corpus and then returns three classes of words; namely, truncated, detruncated, and invalid-truncated.

The truncated-word classification refers to that set of all words in the corpus which begin with either a prefix or prefix-combination. Having been "stripped" by the right truncated algorithm, these truncated words which are returned by the algorithm were ascertained in the corpus as making up a reasonable percentage thereof, i.e., 84% of the entirety. This would indicate 84% of Arabic words, in Arabic script endowed with either prefixes or a combination of prefixes. On the other hand, the remaining percentage, 16%, represents "prefix-less" Arabic words and those in the corpus which are attached to suffixes. The significance here is the fact that prefixes in Arabic script are of great importance since their use in the various Arabic scripts is a common occurrence as compared with the use of suffixes, and likewise, that the removal of such prefixes would improve indexing efficiency respective to 84% of the totality of terms in the corpus. The conclusion is that removing prefixes will reduce the corpus with respect to 84% of the corpus terms. Finally, the removal of Arabic suffixes is not a relevant necessity in the striving for enhancement of indexing efficiency.

The class of detruncated words is comprised of a set of all words in the corpus which begin with a character or character-combination not included in the set of prefixes or prefix-combinations, and this class represents 15% of the entire collection. The right truncated algorithm retains intact each word in this class, and returns each of them to the set of valid indexing terms thus far obtained, as being truncated terms. Including these terms could very well increase the percentage of valid indexing terms.

The class of invalid truncated words is a set of all the words in the corpus which begin with a character or character-combination acting as prefixes or prefix-combinations, and which have been "stripped" by the stemmer to the effect that the returned word will not be found in the corpus. This class represents 0.007556% of the

entire corpus, which is quite small in comparison with the truncated and detruncated class percentages obtained in the same experiment. This problem can, however, be managed by resorting to the use of a small dictionary for accommodating all words and proper nouns, the beginning character or character-combination of which happens to be a prefix; therefore, the stemmer will begin to check for the word in the dictionary and, if it is found, then the stemmer will refrain from "stripping" it, but will retain it intact as being a valid indexing term.

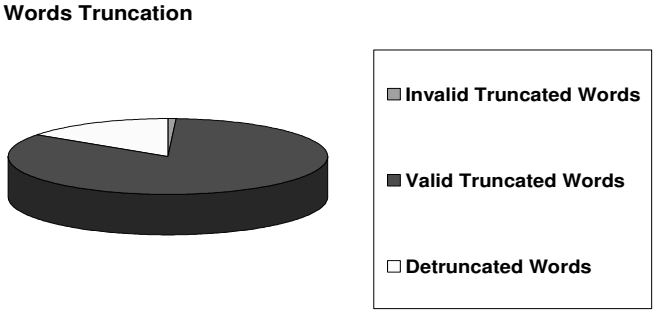

Figure 1: Valid truncated, invalid truncated and detruncated words.

-- ARABIC PREFIXES AND PREFIX-COMBINATIONS IDENTIFIED IN THE EXPRIMENT

The experiment result shows that certain Arabic prefixes or prefix-combinations enjoy priority for their being commonly used in Arabic script. These can be divided into four categories based on their rank according to the result of the experiment. Their categories are described as follows:

Category A embraces the prefixes **al**-, **wa**-, and **li**-, all of which occupy much higher percentages of the entire corpus, whereby the prefix **al**- is assigned 19%, the prefix **wa**- 16%, and the prefix **li**- 12.3% of the collection.

Category B embraces the prefixes **bi**-, **wal**-, **fa**-, and **bial**-, all of them enjoying a lesser percentage than those of Category A, and whereby the prefix **bi**- is assigned 9.5%, the prefix-combination **wal**- 8.7%, **fa**- 8.5%, and **bial**- 5.5% of the collection.

Category C embraces the prefixes **ka**-, **wala**-, **wabi**-, and **waka**-, which enjoy even smaller percentages than those in Category B, and whereby the prefix **ka**- enjoys 4.6%, the prefix-combinations **wala**- 3.6%, **wabi**- 3.3%, and **waka**- 2.16% of the collection.

Category D embraces the prefix-combinations **faka**-, **faal**-, **fala**-, **kaal**-, **fakaal**-, **wabial**-, and **wakaal**-, all enjoying percentages which vary respectively from 1.8% to 0.009%.

-- WORD-FREQUENCY IN THE COLLECTION

The right truncated algorithm computes the weight of the selected words in the collection by counting each of the word's occurrences in order to show that, the more frequent the occurrence, the likelier the word's importance in the corpus. In addition, the word's frequency will be saved for use in further processing.

In the first and second categories of word-frequencies, the highest frequency within the corpus was a result of conditions; propositions; conjunctions; and connective, affirmative, and demonstrative characters.

In summation, all these types of words can not be considered to be valid indexing terms within the framework of the quest for information-retrieval efficacy; thus, the algorithm is programmed to eliminate them during the second phase of pre-processing, using a stop-word list.

-- WORD-TYPES AND THE RATIO OF TOKENS

It may be of some importance to indicate the relationship between frequency of certain distinct words and that of tokens within the corpus, and how the rate of acceleration changes respective to the amount taken from the corpus for the experiment (Figure 3). For instance, whenever the amount taken from the corpus is increased, the number of distinct words likewise increases. Unfortunately, the rate of acceleration shown by the increment of distinct words begins at a definite, specific level (0.020831), and continues on a decrease all along the curve, in a very leisurely fashion, ending on a definite level (0.009568).

-- RATIO OF TRUNCATED WORDS AND TOKENS

The right truncated algorithm begins to read data from a given set of files, and then, during separating words with non-Arabic characters within the input-files, it computes Arabic word-types (distinct words) and Arabic tokens. Also, the right truncated algorithm reads a list of Arabic prefixes and matches their components with Arabic tokens. If there is a match, then the algorithm will return the truncated word; if not, then the algorithm will maintain the token intact. The right truncated algorithm has been evaluated through the use of 83,357,773 words of the Arabic corpus. During this evaluation procedure, 17 constituent runs were conducted.

In each run, the tokens and the word-types were based upon a pair of points within bi-dimensional space, and they were plotted as a curve in order to represent the relationship between the distinct words and the corpus as a whole. On the other hand, the truncated words and the entire corpus were envisioned as a pair of points within a bi-dimensional space, and plotted as a curve in order to represent the relationship between the truncated words and the entire corpus. Hence, a comparison between the

two curves was made possible. The area below the first curve represents the relationship between the corpus and the distinct words within the corpus; otherwise, the area below the second curve represents the relationship between the corpus and the truncated words. It was found that the area below the first curve is equal to approximately twice the area below the second curve. What this means is that the right truncated algorithm reduces the corpus by slightly less than half of the corpus (Figure 2).

In the first two runs, the algorithm accepted a set of files which contained respectively 147,455 and 650,135 tokens as their input, whereupon it returned 15,487 and 34,056 truncated words as the output, thus representing 10.5% and 5.3%, respectively, of the input tokens. In the ensuing 15 runs, for which the input tokens were increased, the percentage continued to decrease until it reached 1.1%.

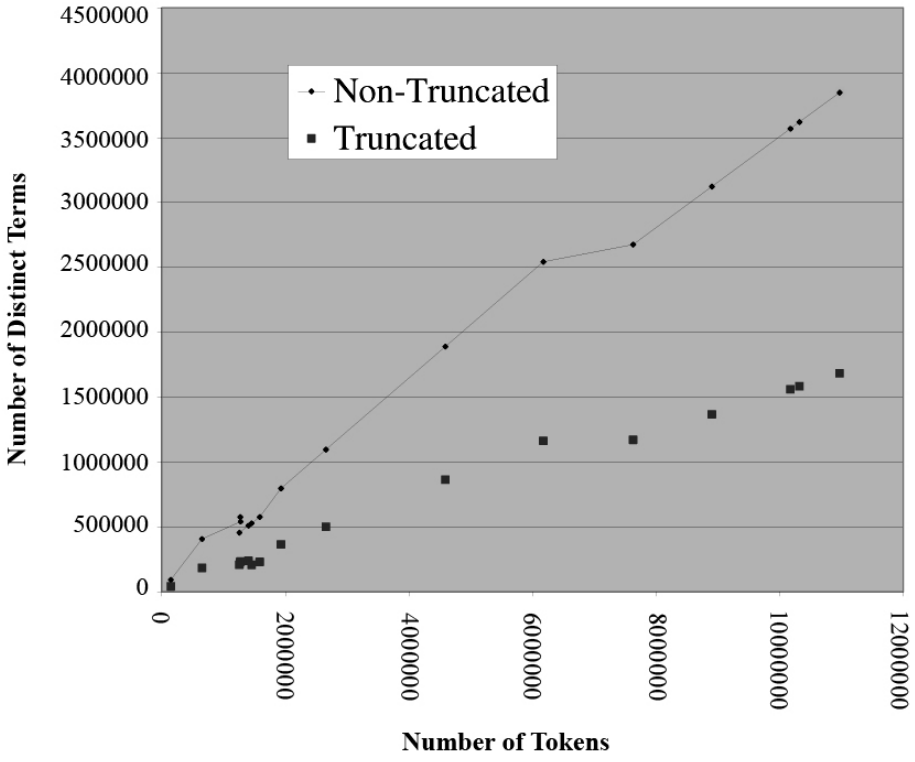

Figure 2: Relationship between token-frequencies and truncated-word-frequencies.

- STEMMERS EFFECTIVENES

The TREC-2001 Arabic corpus provided by LDC was used for documents-retrieval evaluation. A set of 25 queries with relevance-judgments was also applied in the collection. The stemmers (right truncated and stemming algorithms) were trained through use on the corpus. Two lists of terms were drawn up. The first contained word-stems wherein the stemmer algorithm removed the prefixes and suffixes of each word. The second contained right-truncated terms by which the right truncated algorithm was

programmed to remove only the prefixes of each word. In addition, the queries were processed using the same algorithm as the indexing system being tested. Stop-words and terms of one character length were removed. LEMUR, which is the source-code for indexing and search-applications, was used for retrievals. In this experiment, two runs were conducted. In the first, the corpus was indexed using word-stems. In the second, the corpus was indexed using right-truncated words. Eleven recall-points were calculated with maximal precision for both word-stems and right-truncated words, and computed and plotted as shown in Figure 3.

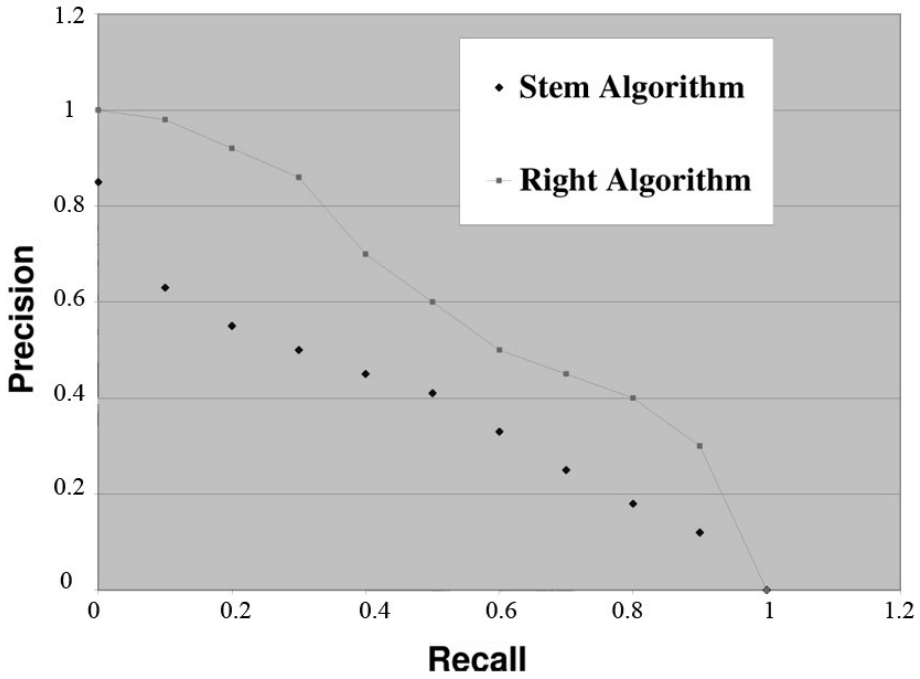

Figure 3: Relative precision of stem-word and right-truncated indexing.

The average precision of the stem-indexing mechanism was 39%. But the average precision of the right-indexing mechanism amounted to 60%. The t-test was used, and the significance was concluded at the level of less than 0.05. These measures suggested that the truncating of term-prefixes and the indexing of the rest of the term are more effective than stem-indexing techniques for information-retrieval.

-- THE VECTOR SPACE-MODEL, OR TF-IDF MODEL

In the second part of this experiment, the corpus is pre-processed by the right truncated algorithm which begins to remove stop-words and truncating-term prefixes, after which the vector space-model is engaged for the purpose of improving information-retrieval efficiency. The collection of the documents was represented by a term-document matrix wherein each entry in the matrix indicates the weight of a term in the document. If the weight of any term is equal to zero, then this signifies that the term

is without significance in the document, or simply does not exist therein.

-- TERM-FREQUENCY (tf)

The more frequent occurrence of terms in a document is significant as being more predictive of the document-topic; i.e., a document containing multiple occurrences of a query-term is probably more relevant to a query than a document containing only one occurrence of the term in question. Thus, the program takes this fact into account and begins to compute the "tf", for the purpose of exhibiting important terms within that document. The program in this case also normalizes such term-frequency by dividing it by the frequency of the most common term in the document. Figure 4 shows the relationship between the normalized term-frequency and the document.

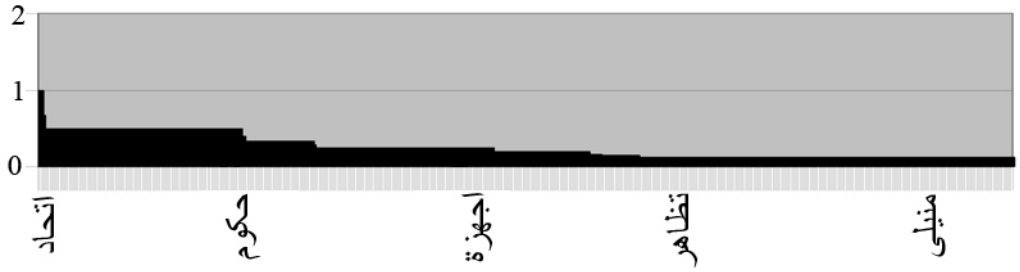

Figure 4: Term-frequency in a document.

-- INVERSE DOCUMENT-FREQUENCY (idf)

Document-frequency, "df", is defined by the number of documents which contains a specific term and plays an important role in the computation of terms-weight within specific documents.

A disadvantage here is the fact that terms which are spread over many different documents are less indicative of overall topics. The main concept here is that any term which seldom appears within different documents should possess a high weight as compared with those terms which appear frequently within different documents in the collection. Here, there is the notion that a term such as الأقتصادي (economist) may have greater meaning, for its appearing less often than does another term, such as الأتجاه (direction). By implementing "idf", a query concerning الأتجاه الأقتصادي (economic direction) will have a greater similarity to a document which contains only الأقتصادي (economist) than to one that contains only الأتجاه (direction).

Therefore, the algorithm computes the "idf" measure in order to increase the weight of the terms that appear but rarely within different documents, and to decrease the weight of those terms which occur frequently within different documents of the collection. Figure 5 shows the relationship between document-frequency and inverse document-frequency, as evidenced in the experiment.

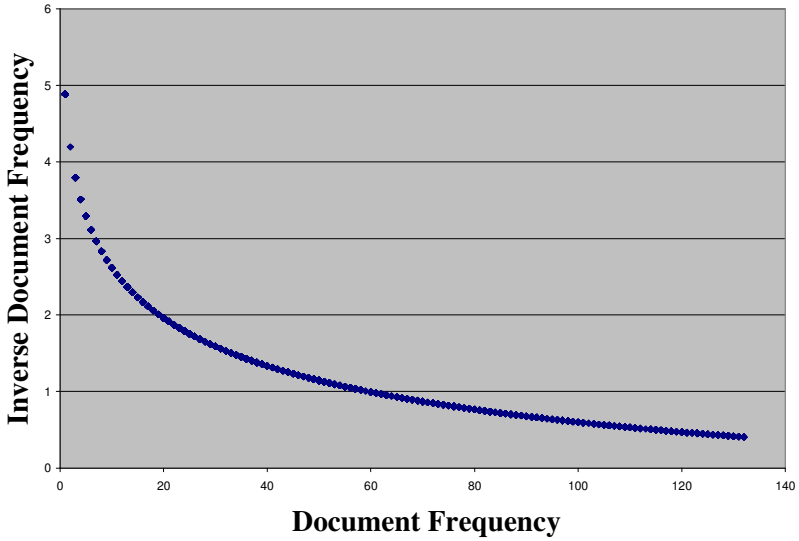

Figure 5: The document-frequency and inverse document-frequency relationship.

-- TERM-WEIGHT (tf-idf)

A most important indicator is "tf-idf" weighting, which combines the impact of term-frequency with that of inverse document-frequency. This type of indicator enables a term to be assigned a high weight on the basis of the qualification that occurs frequently in the document, but only rarely in the rest of the collection. Thus, the program computes the term-weights of different terms throughout the document collection, keeping a running sum of term- and document-frequency. Figure 6 shows the relationship between documentary and term-weight summations.

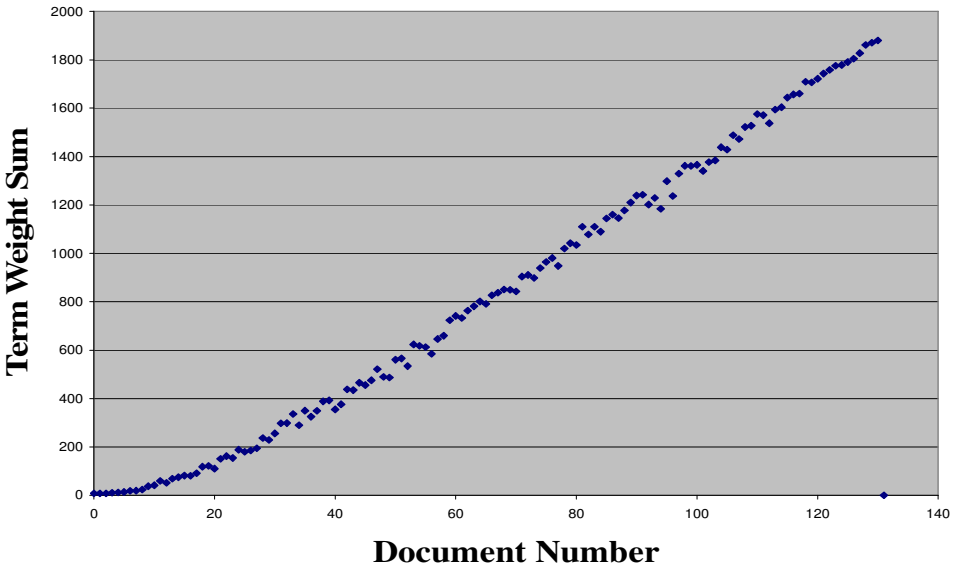

Figure 6: Documentary and term-weight summation.

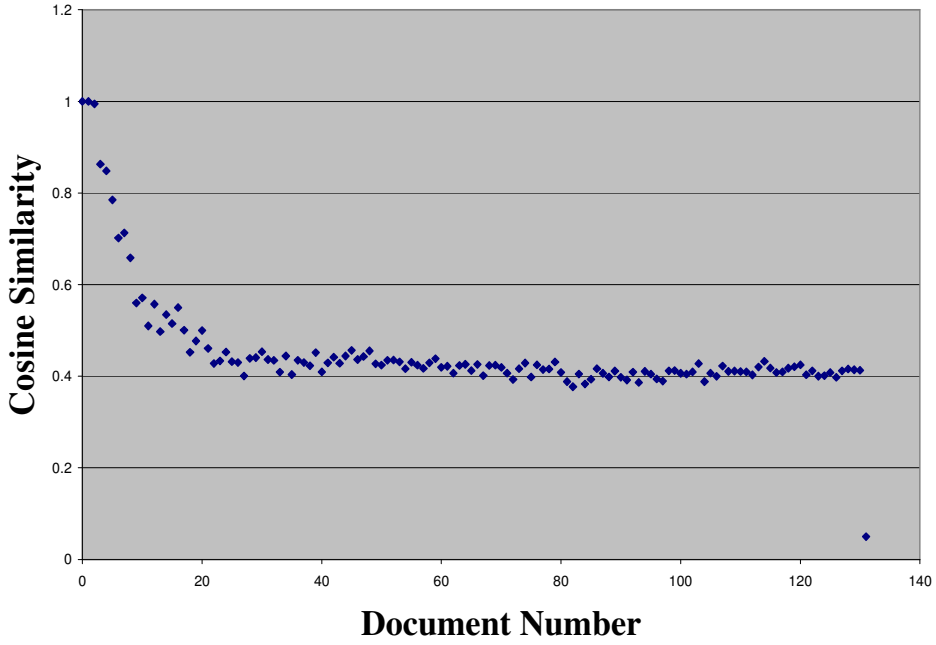

Figure 7: Similarity involving documents and cosines.

With the completion of all term-weight computations, and after having saved them in a bi-dimensional matrix that is, by definition, a documentary-term matrix, the program then computes the cosine similarity between vectors for the documents "di" and the query "q" as the vector inner-product, using the formula (4), as stated in the section entitled “Prior Research”. The document's title and its description were used for representing the query, which was processed in the same manner as were the documents. Hence, the program returns the first 131 documents which possess maximal correlation, as contra distinct from the other documents in the collection. The algorithm ranks the retrieved documents in the order of presumed relevance. Moreover, the algorithm imposes a certain threshold which enables the size of the retrieved set to be controlled.

SPIDER ARCHITECTURE

In this section, the spider software architecture will be described in details. The spider software is partitioned into two major components: spider system and spider application as shown in Figure 8. The spider system consists of several specialized components, in particular: spider administrator (Preferences) and spider application component (Spider).

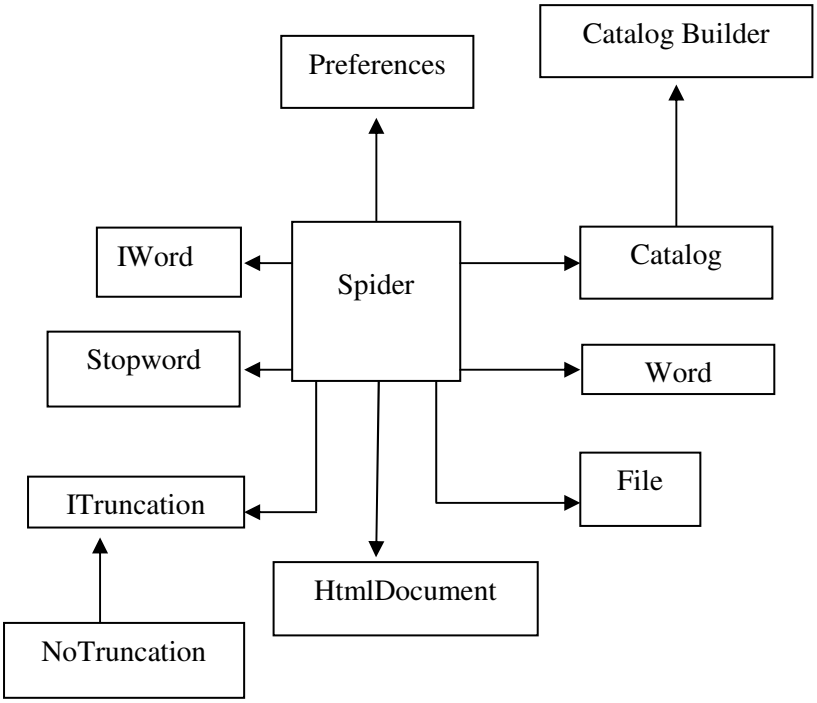

Figure 8: Spider software architecture.

- SPIDER ADMINISTRATOR

The spider administrator is the first module that starts up. Later, other modules start and register with the administrator to send or accept required services. It starts reading settings then checking the cache for the catalog to use. If there is no cache for that catalog, then it checks the file system for a serialized cache. If there is no serialized cache either, then the administrator transfers the control mechanism (Server. Transfer) to the spider application to build a new cache.

Moreover, the spider administrator is in charge of checking the user query string for search arguments. If there is a query string, then it analyzes the query string into search arguments, computes terms weights and then performs the search. Otherwise, it just displays the html of this page as a blank search form.

- SPIDER APPLICATION

The spider application starts out at a set of seed URLs, in our experiment the URLs of the main pages of some Yemen universities. Such a set is initially obtained through the web configuration setting. The spider application consists of nine main components as depicted in Figure 8. Such components are Spider, HtmlDocument, ProgressEventArgs, IWord, NoWord, IStopper, NoStopper, ITruncation, NoTrancation, SerializeCatalog, SerializeWord , SerializeFile and RankFile. These components collaborate and interact

in order to process such a set of Uri's. The starting Uri's passed into another subcomponent that is a BuildCatalog (startUri) which in turn is partitioned into five sub methods: CreateCatalog (), CreateHtmlDocument (startUri), Download(htmldoc), Parse (htmldoc) and Catalog (htmldoc).

- SOFTWARE MODULARITY

As it was mentioned earlier, the main objective of this paper is to design and implement a web search engine with better performance. In order to develop such software, it should be distributable, scalable and extendable. First, the spider algorithm can be distributed across some servers for better performance. Second, it also can be scaled by adding some servers to the spider cluster. Moreover, it can reduce the amount of the memory size used for processing websites regardless of the huge size of such websites. Third, ideally, a spider should be designed in a modular manner, whereas new functionality can be added by third party. Spider algorithm in this paper obtains this feature through a component-based development technique. Each one of the main components has some abstract interfaces. Some of them were implemented while others can be implemented by the third party in the way he or she requires.

-- SOFTWARE DETAILED IMPLEMENTATION

In this section, some important methods within spider application would be described in more details as follows:

Building Catalog Method

- Creates a catalog object that uses a word object.
- Creates a root document object to start the search.
- Performs all the complicated iterative parsing in which the document object is pushed into a queue.
- While (linkq.count > 0)

```
{
   Process (linkq.Dequeue())
}
```

- When the document object processing is over, then the catalog object is added into Cache.

Html Document (Starturi) Method: As soon as the process of a document parsing is over, then HtmlDocument object is used to move and store information about documents. This object encapsulates some properties and methods such as ContentType property and SetRobotDirective (st) method. They index html pages including: Title, Meta tags and Keywords. They return and set the page contents as soon as the page is

downloaded from the server. They set the mime type and encoding whenever such contents are blanks.

Download (Htmldoc): The Download functionality is a function returns a boolean value. If that value is true then, document parsing takes place. Otherwise, the blank page will be shown.

Such a function could be outlined as follows:

- Opens the requested Url.
- Initializes request cookie containiar.
- Adds the document path into the container.
- Gets the stream from the returned web response.
- Checks if the web response is not empty then handles cookies as long as there is any session cookie.
- foreach(cookie rc in cookies)

```
{
 foreach(cookie oldc in c.GetCookies(u)
  {
   if(rc.Name.Equals(oldc.Name))
   {
     Oldc.value==rc.value;
    cookieFound ==true;
   }
   if(!cookieFound)
    {
          c.Add(rc);
    }
  }
}
```

- Parses out mime type and char set.

Parse (Htmldoc): The Parse method extracts information from downloaded file and populates the passed in document object's properties. It accepts a parameter of type HtmlDocument which is called htmlDoc. The method performs the following main tasks:

1. Declaring three variables of type string: htmlData, metaKey and metaValue.

2. Assigning htmlDoc attribute ALL into the varible htmlData, hence, Iterating through html document tag to perform the following sub tasks:

2.1. Initializing metaKey and metaValue variables to blank.

2.2. Starting a new iterative loop that iterates through the attribute/value pair inside the tag of the same document in order to fill both metakey and metavalue with

attributes: http-equiv name and content respectively.

2.3. Checking if metakey variable contains description, keyword or robot then the method assigns metavalue variable into htmlDoc attributes: Description and keyword respectively.

3. Declaring three variables of type ArrayList and string: linkLocal, linkExternal and link respectively.

4. Iterating through each html page in the variable htmlData to parse ALL attributes from within tags. In addition to the 'href' attribute, there might also be 'alt', 'class', 'style', 'area', etc..., there might also be 'space', between the attributes and they may be ',' or unquted. Such sub tasks could be performed as follows:

4.1. Starting a new iterative loop that iterates through href attribute in each page. It is only interested in the href attribute.

4.2. Striping off internal links to prevent indexing same page over again.

4.3. Populating the linkExternal Array for possible future use, under assumption that external links not to be proceesed at all.

4.4. Sending the query string to the default page in a directory in case the query is /? query.

4.5. Adding the link into the linkLocal Array.

Catalog (Htmldoc): Catalog is a function of type integer. It recieves a parameter of type HtmlDocument and returns number of words contained in that document. The function Catalog performs the following main tasks:

1. Declaring five variables filedesc, wordsonly, wordsonly A, fileSize and infile of type string, long and file respectively.

2. Calling another function of type string to stript the parameter htmldoc. It is called StripHtml. It performs the following sub tasks:

- Striping the <script>, <style> and <html> tags.

- Replacing all html tags matches with empty string and all <end> with < and &gt.

3. Assigning wordsonly to filedesc.

4. Updating wordsonly into Keyword, Description attributes of html document in addition to formar wordsonly contents.

5. Spliting wordsonly contents and assigning such contents into wordsonlyA array.

6. Passing url, title, filedesc, DateTime and length attributes of the htmldoc into the File constructor to initialize the object infile.

7. Iterating through wordsonlyA array performing the following sub tasks:

- Triming each word and assigning such a word into key variable.

- Adding each word into the m_catalog object in specific file object in specific position i.

8. Returning the last position i which denotes the size of the catalog as a final result of the function value.

Process (Htmldoc): Process (htmldoc) accepts an argument of type HtmlDocument and performs the following tasks:

1. Declaring five variables: filedesc, url, filesize, wordcount and links of type string, long, integer, and ArrayList respectively.

2. Initializing url variable into htmldoc. Uri. AbsoluteUri attribute.

3. Checking if the url is existed in the hash table contains urls visited so far.

4. Storing the url into the hash table in case it has not stored yet.

5. Checking if the variable htmldoc is downloaded, in case it is true then the process performs the following sub tasks:

- Parsing the htldoc argument.

- Computing number of words in the htmldoc.

6. Checking htmldoc LocalLinks attribute in case it is not null, then the process adds htmldoc LocalLinks attribute into links.

7. Checking htmldoc ExternalLink attribute in case it is not null, then the process adds htmldoc ExternalLink attribute into links as well.

8. Checking links in case it is not null and htmldoc RobotFollowOk, then the process iterates through the links in order to add each link contained in links into the linked queue.

CONCLUSIONS AND FUTURE RESEARCH ASSIGNMENTS

It has been thus far established that the initial portion of the present experiment succeeded in proving that the right truncated stemmer reduces the corpus to at least 50% of the original entire corpus. It also removes the stop-words from the corpus, resulting in a reduction in the size of the indexing system. It determines which parts of the words are prefixes, or combinations of prefixes. Finally, it demonstrates the extreme importance of Arabic prefixes, and how they are commonly used in the various Arabic scripts. Otherwise, it has proven its worth in the process of retrieving Arabic documents by using the right truncated indexing mechanism, with an average success-rate of 60%, which is greater than that (39%) obtained through use of the stem-indexing technique. The second portion of the present experiment makes use of the vector space-algorithm which employs the right truncated algorithm output in order to map out the terms in a given query, involving each of the related documents in the collection under discussion. This algorithm is capable of ranking, i.e., classifying, the retrieved documents in order of cosine similarity existing between the query and the relevant documents: the closer the document is relevant to the query, the closer to the top it will appear in the retrieved

list.

The algorithm is also capable of imposing a certain threshold, thus limiting the size of the number of documents retrieved, and increasing the number of relevant documents for improvement of efficiency. The combined or synthesized algorithm (i.e., the right truncated algorithm and the vector space) possesses a very high level of efficiency, especially as compared to those algorithms which are based only on the stemming aspect. In the present case, the average level of precision can reach 100%, once the threshold is adjusted. However, the algorithm does have its limitation respective to the documentary-terms matrix: when the corpus is rather expansive, the computation process requires time for computing term-frequency, inverse-document frequency, term-weight, and cosine similarity between query-terms and documentary-terms peculiar to the collection. To meet this problem, the collection invites pre-computation which may be comprised of: right-truncation of terms; stop-word removal; indexing of the right-truncated portion; computation of term-weight; and finally, saving the term-weights on to a hard disk or distributed servers accessible by a background thread, whereupon the query-weights must be computed with the use of the same algorithm as a front thread. That is, computing the cosine similarity between the query-terms' weight vector and each documentary-term vector. The measurement thus computed would be usable for retrieving the relevant documents.

The initial portion of this algorithm (the right truncated algorithm) could be expanded in order to process Arabic dialects. It could likewise be combined (synthesized) with some model for the purpose of processing both structured and unstructured data. This same problem could be solved through combination of the initial portion of the algorithm and the neural-networks algorithm.

This paper also describes the design and implementation details of the spider software. It presents a preliminary experiment result. Such result shows that the achievable crawling speed is 160 pages per second. It is a reasonable result compared to other result recorded by other researchers where they recorded 140 pages per second. The stemmer convolves both Arabic and English words into their original forms that reduces the required size of the catalog. On the other hand, the stemmer is able to retrieve the most relevant document presents to the query.

REFERENCES

[1] Aljlayl, M., et al., "On Bidirectional English-Arabic Search." *Journal of the American Society for Information Science and Technology*, Vol. 53, No. 13, pp. 1139-1151, 2002.

[2] Altinel, M. and Franklin, M., "Efficient Filtering of XML Documents for Selective

Dissemination of Information." *Proceedings of the 26th VLDB Conference Cairo,* Egypt, 2000.

[3] Baeza-Yates, R. and Castillo, C., "Web Search." *Proceedings of the third Workshop on Web Graphs (WAW)*, Vol. 3243 of Lecture Notes in Computer Science, pp. 156-167, Rome, Italy, Spring, 2005.

[4] Broglio, J., et al., "Inquery System Overview." *TREC-9 Gaithersburg*, Maryland, 2000.

[5] Buckwalter, T., "Buckwalter Arabic Morphological Analyzer Version 2.0." Available: at <www.idc.upenn.edu/catalog/catalogEntry.jsp?catalogld=LD2004 Lo2>, 2004.

[6] Darwish, K., "Building a Shallow Arabic Morphological Analyzer in One Day." *in ACL02 Workshop on Computational Approaches to Semitic Languages*, Philadelphia, PA, Association for Computational Linguistics, 2003.

[7] Grossman, D. A. and Frieder, O., *Information Retrieval Algorithm and Heuristics*. Second Edition, Springer, 2004.

[8] Habash, N., et al., "Morphological Analysis and Generation for Arabic Dialects." *Proceedings of the ACL Workshop on Computational Approaches to Semitic Languages*, pp. 17-24, Ann Arbor, June 2005.

[9] Habash, N. and Rambow, O., "Arabic Tokenization, Part-of-Speech Tagging and Morphological Disambiguation in One Fell Swoop." *Proceedings of the 43rd Annual Meeting of the ACL*, Association for Computational Linguistics, pp. 573-580, Ann Arbor, June 2005.

[10] Jinxi, X., et al., "Empirical Studies in Strategies for Arabic Retrieval." *Proceeding of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retreival,* Tampere, Filand, Augest 2002,

[11] Leah, S. L. and Connell, M. E., "Arabic Information Retrieval at UMass in TREK-10." *TRECK 2001*, Gaithersburg: NIST, 2001.

[12] Leah, S., et al., "Improving Stemming for Arabic Information Retrieval: Light Stemming and Co-Occurrence Analysis." *SIGR'02*, Tampere Finald, Augest 2002.

[13] Levy, A. Y., et al., "Querying Heterogeneous Information Sources Using Source Descriptions." *Proceedings of the 22nd VLDB Conference Mumbai (Bombay)*, India, 1996.

[14] Moody, K. and Palomino, M., *Spidering the Web through Web Services*. University of Cambridge press, 2003.

[15] Moukdad, H., "Lost in Cyberspace: How Do Search Engine Handle Arabic Queries?" Available: at <http://www.morfix.com/arabicSearch/arabic/help.html>, 2004.

[16] Najork, M. and Heydon, A., *High-Performance Web Crawling*. Kluwer Academic Publishers Inc., 2001.

[17] Oard, D. W., "The TRECK-2002 Arabic/English CLIR Track." *SIGIR'02*, Tampere, Filand, Augest 2002.

[18] Schultz, J. M. and Liberman, Y. M., "Towards a Universal Dictionary for Multi-Language Information Retrieval Applications." Available: at <http://www.ldc.upenn.edu>, May 2000.

[19] Weeds, J. and Weir, D. "Co-Occurrence Retrieval: A Flexible Framework for Lexical Distributional Similarity." *Association for Computational Linguistics*, 2006.