6.1 INTRODUCTION

The users' current topic of interest can be assessed by maintaining a record of all their queries submitted during their recent sessions with a search engine. For example, during the time of parliamentary elections, most of the users sift through the information about the political parties and the candidates. During admission time the people surf through the information regarding universities and colleges for particular courses being offered by them. Similarly, in case of announcement of some new rules or new salary structure, user wants to track the latest information through search engine. These situations necessitate the need to pre fetch the relevant information by the crawler.

Infact, a critical look at the record of users' queries can help in estimating about their affiliation and possible future trend of their expectations in the sessions to come. Consequently, the crawler needs to be accordingly restrained to pre fetch information as per the user's interest. In this work a method for tracking the current user interest and downloading the relevant information is being proposed.

6.2 THE ARCHITECTURE

As discussed in chapter 2, generally the popular search engines like Google and Yahoo are used for gathering information from the web. However, with the explosive growth of the web, fetching information about a current topic is becoming an increasingly difficult task. Thus, much of the latest information relevant to the current topics may not have been gathered on time. In the past, focused crawlers have been developed that allow retrieval of the information from the WWW related to a specific topic. They generally rely on the topic specific seed URLs supplied by the user. The downloaded pages are checked for similarity to the topic and if the similarity is above a specified threshold, the
pages are indexed by the search engine and the links from the pages are registered. The
crawler continues to follow the out-links embedded in the documents up to a user-
specified depth. However, the drawback is that the system doesn’t takes into account the
changing interest and behavior of the users.

The mechanism, developed in this work, keeps track of users’ topics of interest. It parses
the queries supplied by the users’ to extract the keywords. To each keyword extracted,
temporal relevance is assigned by way of attaching a weight. On the basis of weight, top
N numbers of temporal relevant keywords are selected which are supposed to be
important, from users’ point of view. The set of keywords is used to explore the local
collection for identifying the documents, relevant to the users’ interest. For every
identified document a weight called $docRelWeight$ is computed and URLs of highly
$docRelWeighted$ documents are assigned to the crawl manager for frequent downloading.
Thus, the mechanism ensures that search engine collection contains temporal relevant
information as per users’ current topic of interest.

The layout of various components of temporal relevance improvement mechanism for
migrating crawler collection is shown in Fig. 6.1.
A brief discussion on various components and data structures is given below:

6.2.1 CREATE qryRecord

For assessing the current user interest, this process keeps track of all users’ queries supplied to a search engine during recent sessions. It stores all queries in a record called qryRec.

6.2.2 qryRec

This data structure is a record of all queries supplied to a search engine during a test period (say 12 hours). The process called create qryRec takes input queries from search engine and stores them in this data structure. This record is used by the create keyRecord process for extracting the keywords from user queries.

6.2.3 CREATE kwRecord

This process takes the query record as input and extracts keywords from it by removing the stop word and break points. Extracted keywords are stored in a record called kwRecord.

6.2.4 kwRec

This data structure is a list of all keywords extracted from qryRec. This list is used by the process called compute keyWeight and identify docs for computation of keyword weight and identifying corresponding documents from the search engine collection.

6.2.5 COMPUTE keyWeight AND IDENTIFY DOCS

This process takes top N keywords from the kwRec and performs two tasks: computes the keyword weight i.e. keyWeight and identifies the weight of the documents where these keywords occur in the collection.
6.2.5.1 Keyword Weight Computation

For every selected keyword a weight called *keyWeight* is computed. The top N keywords are selected as temporal relevant and of users’ interest. The weight of a keyword is computed as follows:

Let,

- \( kwFreq_i \) is the total number of occurrences of \( i^{th} \) keyword in the keyword record i.e. keyRec.
- \( totKw \) is total number of keywords in the keyRec.

For each keyword in the keyRec, the *keyWeight* is computed using the following formula:

\[
\text{keyWeight}_i = \left(1 + \ln(kwFreq_i)\right) \times \frac{(kwFreq_i)}{totKw}
\]

(6.1)

This *keyWeight* is used to select the top N keywords from the keyRec. The selected keywords are used for identifying the temporal relevant documents in the search engine collection.

6.2.5.2 Document Weight Computation

The top N keywords are used to explore the local collection for identifying the documents relevant to the users’ interest. For every document in the collection, where any of the top keyword occurs, a weight based on keywords occurrence in the document called *docWeight* is computed. The weight of \( i^{th} \) keyword in the document ‘\( j \)’ is computed as follows:

Let

- \( kwDocFreq_{ij} \) is the total number of occurrences of \( i^{th} \) keyword in the document \( j \).
- \( totWord_j \) is total number of words in the document \( j \).

The weight of \( i^{th} \) keyword in the document ‘\( j \)’ *docWeight* is computed as per the following formula:

\[
docWeight_{ij} = \left(\ln(1 + kwDocFreq_{ij})\right) \times \frac{(kwDocFreq_{ij})}{totWord_j}
\]

(6.2)
This document’s keyword weight is employed to compute the document relevance weight as discussed in following section.

6.2.6 COMPUTE DOCUMENT RELEVANCE

The document relevance computation is made on the premise that document relevance weight should increase for the keywords whose document frequency is less at the search engine’s local repository. Though, there may be several documents in the collection where the top N keywords occur. Therefore, it is suggested that instead of computing the document relevance for every document, the relevance should be computed for a category. Where, a category is made by counting the total number of documents falling in a specified range of docWeight. After determining the document frequency (i.e. number of documents) in the range intervals, a weightage called rangFactor is associated with all the ranges according to the highest docWeight in the interval.

Let,

\[ \text{docFreq}_{ik} \] is the total numbers of documents having docWeight falling in k\textsuperscript{th} docWeight range for i\textsuperscript{th} keyword (See Table 6.3)

\[ \text{totDocs} \] is total number of documents in the collection where any of the top keywords has occurred.

\[ \text{rangFactor}_k \] is the highest docWeight in the k\textsuperscript{th} docWeight range.

The document relevance weight for k\textsuperscript{th} collection of documents for corresponding range \( \text{docRelWeight}_k \) is computed using the following relation:

\[
\text{docRelWeight}_k = \frac{\text{totDocs}}{(1 + \text{docFreq}_k)} \times \text{rangFactor}_k
\]  

(6.3)

It may be noted that the \( \text{docRelWeight} \) increases with the decrease in document frequency in the range. It ensures that the documents belonging to a docWeight range, having less number of total documents gets a higher weightage. The URLs of the documents having higher \( \text{docRelWeight} \) are assumed to be temporal relevant and are stored in a separate record called list of URL-docRelweight pair. The top N URLs are supplied to the crawl manager for the purpose of downloading the documents and maintaining the temporal
relevance of the collection. The working of the mechanism is illustrated through the following example which is, test run conducted on a set of queries issued at www.google.com.

6.3 EXAMPLE

Step 1: The query record (qryRec) generated for the search queries fired by a user at google during a test period of 2 hours is shown in Fig. 6.2.

![Query Record](image)

These queries consist of stop words where stop words are the words that are not significant in a sentence and have no relevance and importance as far as search queries are concerned. For example ‘articles’, ‘prepositions’ or any other similar common words are stop words. A list of common stop words is maintained by the crawler, the contents of which are given in Fig. 6.3.

![Common Stop Words](image)
Step 2: The keywords are extracted from the query record and the extracted keywords are stored as a keyword record called keyRec as shown in Fig 6.4. It may be noted that the keyRec is free from stop words and it contain 33 keywords in total i.e. \( \text{totKw}=33 \).

```
"Gate result"
"Pay commission"
"Assembly election result"
"election"
"Pay commission"
"Pay commission India"
"Pay teachers"
"AICTE pay commission notification teachers"
"Gate result"
"AICTE technical teachers"
"Engineering college haryana"
"Haryana pay commission"
"Gate Score"
```

\( \text{totKw}=33 \).

Fig. 6.4: Keyword Record (keyRec)

Step 3: For every keyword existing in keyRec, the keyword weight i.e. keyWeight is obtained by applying equation 6.1 and the result is listed in Table 6.1 in the descending order of keyWeight.

Table 6.1: Keyword Weight (keyWeight)

<table>
<thead>
<tr>
<th>S No</th>
<th>Keywords</th>
<th>kwFreqi</th>
<th>keyWeight</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Pay</td>
<td>6</td>
<td>0.507</td>
</tr>
<tr>
<td>2</td>
<td>Commission</td>
<td>5</td>
<td>0.395</td>
</tr>
<tr>
<td>3</td>
<td>Gate</td>
<td>3</td>
<td>0.191</td>
</tr>
<tr>
<td>4</td>
<td>Result</td>
<td>3</td>
<td>0.191</td>
</tr>
<tr>
<td>5</td>
<td>Teachers</td>
<td>3</td>
<td>0.191</td>
</tr>
<tr>
<td>6</td>
<td>Election</td>
<td>2</td>
<td>0.103</td>
</tr>
<tr>
<td>7</td>
<td>AICTE</td>
<td>2</td>
<td>0.103</td>
</tr>
<tr>
<td>8</td>
<td>haryana</td>
<td>2</td>
<td>0.103</td>
</tr>
<tr>
<td>9</td>
<td>Assembly</td>
<td>1</td>
<td>0.03</td>
</tr>
<tr>
<td>10</td>
<td>India</td>
<td>1</td>
<td>0.03</td>
</tr>
<tr>
<td>11</td>
<td>Notification</td>
<td>1</td>
<td>0.03</td>
</tr>
<tr>
<td>12</td>
<td>Technical</td>
<td>1</td>
<td>0.03</td>
</tr>
<tr>
<td>13</td>
<td>Engineering</td>
<td>1</td>
<td>0.03</td>
</tr>
<tr>
<td>14</td>
<td>College</td>
<td>1</td>
<td>0.03</td>
</tr>
<tr>
<td>15</td>
<td>Score</td>
<td>1</td>
<td>0.03</td>
</tr>
</tbody>
</table>
Step 4: In the current test the following 5 keywords \((keyWeight>=0.15)\) were selected.

"pay", "commission", "gate", "result", "teachers"

These keywords were tested on a collection of documents containing approx 70800 documents, for the purpose of computing document weight \((docWeight)\). For instance, the keyword "pay" occurs 8 times in a document consisting of 107 words in total. Applying the relation 6.2 we get:

\[
docWeight = (\ln(1+8))^8/100 = 0.176.
\]

The computed \(docWeight\) is taken as the basis of classifying the documents belonging to a particular category i.e. range of \(docWeights\). For each category, the document frequency \((docFreq)\) is computed and the values of \(rangFactor\) corresponding to \(docWeight Range\) are given in Table 6.2.

<table>
<thead>
<tr>
<th>S No</th>
<th>(docWeight Range)</th>
<th>(rangFactor)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(0.0 &lt; docWeight &lt; 0.1)</td>
<td>0.1</td>
</tr>
<tr>
<td>2</td>
<td>(0.1 &lt; docWeight &lt; 0.2)</td>
<td>0.2</td>
</tr>
<tr>
<td>3</td>
<td>(0.2 &lt; docWeight &lt; 0.3)</td>
<td>0.3</td>
</tr>
<tr>
<td>4</td>
<td>(0.3 &lt; docWeight &lt; 0.4)</td>
<td>0.4</td>
</tr>
<tr>
<td>5</td>
<td>(0.4 &lt; docWeight &lt; 0.5)</td>
<td>0.5</td>
</tr>
<tr>
<td>6</td>
<td>(0.5 &lt; docWeight &lt; 0.6)</td>
<td>0.6</td>
</tr>
<tr>
<td>7</td>
<td>(0.6 &lt; docWeight &lt; 0.7)</td>
<td>0.7</td>
</tr>
</tbody>
</table>

In the current test the total documents where any of the selected keyword occurred i.e. \(totDocs\) comes out \(4363\). The values of \(totDocs\) and the \(docFreq\)ies are used to compute the \(docRelWeight\) by applying equation 6.3.

In the Table 6.3 the computed \(docFreq\) and the average \(docRelWeight\) for all selected keywords are given.
Table 6.3: *docFreq* and the Average *docRelWeight*

<table>
<thead>
<tr>
<th>Cg</th>
<th><em>docWeight Range</em></th>
<th>rF</th>
<th>pay</th>
<th>commission</th>
<th>gate</th>
<th>result</th>
<th>teachers</th>
<th>Average dRW</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>dF</td>
<td>dRW</td>
<td>dF</td>
<td>dRW</td>
<td>dF</td>
<td>dRW</td>
</tr>
<tr>
<td>1</td>
<td>0.0&lt; <em>docWeight</em> &lt;0.1</td>
<td>0.1</td>
<td>185</td>
<td>2.35</td>
<td>282</td>
<td>1.55</td>
<td>17</td>
<td>25.66</td>
</tr>
<tr>
<td>2</td>
<td>0.1&lt; <em>docWeight</em> &lt;0.2</td>
<td>0.2</td>
<td>220</td>
<td>3.94</td>
<td>74</td>
<td>11.79</td>
<td>98</td>
<td>8.91</td>
</tr>
<tr>
<td>3</td>
<td>0.2&lt; <em>docWeight</em> &lt;0.3</td>
<td>0.3</td>
<td>150</td>
<td>8.66</td>
<td>273</td>
<td>4.79</td>
<td>135</td>
<td>9.69</td>
</tr>
<tr>
<td>4</td>
<td>0.3&lt; <em>docWeight</em> &lt;0.4</td>
<td>0.4</td>
<td>127</td>
<td>13.63</td>
<td>157</td>
<td>11.12</td>
<td>80</td>
<td>21.82</td>
</tr>
<tr>
<td>5</td>
<td>0.4&lt; <em>docWeight</em> &lt;0.5</td>
<td>0.5</td>
<td>70</td>
<td>31.16</td>
<td>46</td>
<td>47.42</td>
<td>158</td>
<td>13.81</td>
</tr>
<tr>
<td>6</td>
<td>0.5&lt; <em>docWeight</em> &lt;0.6</td>
<td>0.6</td>
<td>53</td>
<td>49.39</td>
<td>65</td>
<td>40.27</td>
<td>105</td>
<td>24.93</td>
</tr>
<tr>
<td>7</td>
<td>0.6&lt; <em>docWeight</em> &lt;0.7</td>
<td>0.7</td>
<td>115</td>
<td>26.55</td>
<td>81</td>
<td>37.71</td>
<td>306</td>
<td>9.98</td>
</tr>
</tbody>
</table>

Cg: Category
rF: *rangFactor*
dF: *docFreq*
dRW: *docRelWeight*
It may be observed from Table 6.3 that the average docRelWeight is largest for category 6 indicating that documents belonging to this category are more relevant as compared to other categories. Hence, these documents need to be refreshed more frequently. Therefore, the revisit frequency field of URL record corresponding to these documents is modified with the upper threshold value of revisit frequency. Consequently, these documents are placed at the top of resolved URL-IP queue and accordingly they get crawled more frequently.

Thus, the mechanism developed in this work renders the collection rich and relevant by taking into consideration the queries fired by the user from time to time.

Next chapter concludes the outcome of the work proposed in this thesis. Future possibilities of research work based on the proposed design are also suggested.