Query Processing

1. Introduction

The query engine collects search terms from the user and retrieves pages that are likely to be relevant. The quality of search results is most important factor for the success of any search engine. We have tried to formulate a search strategy that will return the most relevant results in response to a user query within an acceptable time limit. We also need to design an elegant and easy to use inferface to accept the query from the user and to display the results.

2. Design Issues

The main design goals are listed below:

- **Fast Response Time** - It is essential that the results be presented to the user quickly. No user can be expected to adopt a search engine which takes tens of seconds even though it may return the most relevant results. This design goal limits the amount of computation we can perform before returning the result. We cannot hope to achieve this goal without an efficient indexer.
- **Relavent Results** - It is quite obvious that a search engine needs to display the most appropriate results in response to a query. Algorithms for measuring the relevance of a page may range from the very simple to the overly complicated. Factors such as storage space and computing capacity have to be taken into account before choosing an algorithm for implementation.

The major problem with the two listed goals is that they can be contradictory. To produce relevant results, we may need to perform elaborate computations which can increase the response time. The aim is to maintain a delicate balance between reasonably fast response times and result relevance.

3. Inteface to Other Modules

To calculate the relevance of a page, we require information that is mostly maintained by the Indexers. The Indexers maintain the data in a format that allows fast access. For our purposes, we assume the availability of the following logical entities all of which perform some form of mapping:

- A **Lexicon** that converts a Word into a WordID. The Lexicon should contan all words for which the user can query the search engine.
- A **Small Barrel** that contains a mapping of WordIDs to DocIDs, the PageRank and the number of hits corresponding to each DocID. A WordID is mapped to a DocID if another document uses that word in its anchor text to link to that document, or if that word occurs in the title of that document. Given a WordID, the small barrel should return the list of DocIDs, the PageRank and the total number of times that the word occurs in the anchors and title of each document. The results should be sorted on the DocIDs.
- A **Big Barrel** that conatins a mapping of WordIDs to DocIDs, and the PageRank and HitList for each listed DocID. A WordID is mapped to a DocID if the word occurs in that document. The HitList corresponding to each document contains details about each occurrence of the Word in that document. Information such as font-size, position, capitalization etc. are stored for each word occurrence. The big barrel will also contain the entries contained in the small barrel. Given a WordID, the big barrel should return the corresponding list of DocIDs, and the PageRank and HitList for each listed DocID. The results should be sorted on the DocIDs.

- A **Document Index** which maps each DocID to its corresponding URL and its Title. This information is required to display the search results.
- A **Repository** which contains the full-text of a document, in compressed form, and is indexed by the DocID. The full-text of the document is used to display relevant portions from the document when the results are displayed to the user

4. Algorithm

The steps required to evaluate a query are listed below:

1. Parse the query into its constituent words.

2. For each word, obtain the WordID corresponding that word from the Lexicon.
   Request: Word
   Response: WordID

3. Send the WordIDs to the small barrel to get the list of DocIDs, and the PageRank and number of hits corresponding to each DocID. The returned results will be sorted based on the DocIDs.
   Request: WordID
   Response: DocID1, PageRank, number of hits,
   DocID2, PageRank, number of hits,
   ...
   A request will be sent for each of the WordIDs that we obtained in step 2.

4. Get the DocID of the next document that contain all the words entered in the query. Since the results retured in step 3 are sorted by DocIDs, this task should be easy.

5. Compute the Final-Rank of the document returned in the previous step. The Final-Rank is based upon the PageRank and the total number of hits of all query terms

6. If "max" documents have been found, skip to step 11 else, if we have not come to the end of the small barrel, goto step 4.

7. Send the WordIDs to the big barrel to get the list of DocIDs, PageRank and the hit-list for each document. The returned results will be sorted based on the DocIDs.
   Request: WordID
   Response: DocID1, PageRank, hit-list,
   DocID2, PageRank, hit-list,
   ...
   The Hit-List corresponding to a DocID carries details (such as font size, position, capitalisation, etc.) about each occurrence of queried word in the document.
   A request will be sent to the big barrel for each of the WordIDs that were in step 2.

8. Get the DocID of the next document that contain all the words entered in the query. This is made easier by the fact that the results are presorted based on DocIDs.

9. Compute the Final-Rank of the document returned in step 8. The Final-Rank is based upon the PageRank, the proximity of occurrence of queried words in the document, the font-size and the capitalization of the occurrences.

Depending on the words entered by the user, we may get anywhere from a few tens to several million matching pages. Calculating the Total-Rank of several million pages may take tens of seconds. This delay is totally unacceptable. To put a limit on the response time, we will not be calculating the Total-Rank of all documents that have the queried words. We will limit ourselves to, say, the first "max" (say 25000) documents (i.e., skip to step 11 as soon as "max" matching documents have been found). The limit specified can be varied to get the desired response time, although it must not be made too low.

It may seem that the limit imposed above will severly affect the relevence of the results returned by us but infact it won't. Since we will be searching the small barrel first, the most relevant results should not be missed due to this imposed limit. Even then, there is a minor possiblity that a few relevant documents may not be returned to the user. For sake of usable response times, this tradeoff will have to be made.

The information contained in the small barrel is different from the information contained in the big barrel. The small barrel only contains the number of hits while the big barrel contains detailed information about each hit. We will require separate algorithms for calculating the Total-Rank of each document. The proposed method that can be used to calculate the Total-Rank is described below:

1. **Total-Rank calculation for small barrel**
   After we have finished with step 4 given above, we will have the list of all DocIDs that have all the words entered in the query. For each DocID, we have the total number of occurrences of each queried word. We have a function that converts counts to points. The function is such that the points increase linearly with counts at first but quickly taper off so that more than a certain count will not help. To see why such behaviour is required, we take an example.

   > Suppose the user queries for two words, **WordA** and **WordB**. **Doc1** conatins 500 instances of WordA but only 1 instance of WordB. **Doc2** on the other hand contains 200 instances each of WordA and WordB. Obviously, Doc2 is more relevant to the query posted by the user. The function described above will ensure that the total points of Doc2 are more than Doc1.

   To calculate the Total-Rank of a page, we add the points for each queried word to the normalized PageRank. The PageRank is normalized by multiplying it by some weight. This allows us to control the effect that the PageRank has on the Total-Rank. The formula for calculating the Total-Rank is given below:

   ```
   Total Rank = (Wt x PageRank) + Pts(Word1) + Pts(Word2) + ...
   ```

   `Pts(Word1)` calculates the points corresponding to the occurrences of Word1. `Wt` is the weight that is multiplied with the PageRank to normalize it.

2. **Total-Rank calculation for big barrel**
   The big barrel provides us with information about each occurrence the word in the document. This information is used in much more complicated Total-Rank calculation function. For multiword queries, we calculate the proximity of words occuring nearby. The proximity can be calculated because we have the position of each occurrence of the words. The proximity is classified into few classes such as occuring in same phrase (1 to 3 words seperating them), seperated by 3 to 10 words, seperated by 10 to 50 words and seperated by more than 50 words. Each class has a corresponding function that can be used to calculate the points depending on the number of elements in that class.

We will also classify the occurrence of each word into classes depending upon its font size, capitalization, whether it occurs in the title/anchor text etc. Each class will have function to calculate the points depending on the number of element in that class, and the points thus obtained for each class will be added together. The best function for each class can be decided by experimenting and selecting the ones which give the most relevant results. The points obtained will be added to the PageRank of the document to obtain the Total-Rank of the page corresponding to the set of queried words.

3. `Total Rank = (Wt x PageRank) + Pts. from proximity + Pts. due to font size, title/anchor etc.`

5. Future Work

- **Searching all Pages** - Due to time constraints, currently we are only searching a subset of all available pages before returning the results. Though our policy tries to ensure that the most relevant pages are not skipped, there is a possiblity that we may return sub-optimal results. To remedy this, we could search all pages if faster hardware and bigger memories and caches are available in the future.
- **Caching** - To decrease the response time, we may cache the results corresponding to most frequently queried terms in memory. Reducing the response time has the added benefit of allowing the use of more complex algorithms which would result in better quality results.