ACKNOWLEDGEMENT

At the successful completion of my dissertation I would like to convey my sincere gratitude towards my professor and supervisor Dr Rafal Bogacz at the University of Bristol and Mr Edward Ross from RossTech. Without their guidance and support this dissertation would not have been completed.

I would like to thank my parents and sister for their constant encouragements in making this dissertation a success.

I would also like to thank all my friends and colleagues of the University of Bristol who guided me and supported me.

Also I would extent my gratitude towards the authors of all the literature that I used as reference to my project.

Summary

MatchMaker is a high performance fault-tolerant products search engine. Its fault-tolerant behaviour handles any minor spelling or the typographical errors made by the user while typing the query, such that it does not cause any big changes to the effectiveness of the search engine. When the user enters a part of the product name or its description, the MatchMaker Search engine performs an extensive search for the query words in all the product description it contains in its corpus collection. The products which the MatchMaker find to contain the description that matches with the query, are ranked based on their relevancy to the query and displayed to the user as search result.

The aim of the project is to improve the effectiveness of Exorbyte MatchMaker search engine. The relevance score of each product with respect to the query should be calculated in an efficient way to attain this significant improvement in performance.

The three different techniques which were used in this project to assign appropriate weight to the query words and to efficiently calculate the relevance score of each product are Term Frequency – Inverse Document Frequency (*tf-idf*), Weighted Inverse Document Frequency (*WIDF*) and Weighted Term frequency. There are few similarities and few differences among these techniques. All the three techniques were clearly studied and their performances were compared.

A few other minor improvisations were also incorporated to handle the co-occurrence of the query words in the product description. The product descriptions in which the query words co-occur are given higher scores than the product descriptions in which the query words appear far apart.

The results obtained from the project prove that a significant improvement was made to the effectiveness of the current implementation of the Exorbyte MatchMaker Search engine. Out of the three different techniques which were used in the project to calculate the relevance scores, Weighted Inverse Document Frequency method was found to perform better than the other two techniques.

TABLE OF CONTENTS

1. Introduction

Human mind cannot always remember everything that we want it to and sometimes we remember them but they get lost in a large crowd that makes it very hard to uniquely identify the one we are searching for. These are the scenario in which search engine comes in handy and assists in our search. When we enter a part of the name or some keywords from the description, the search engine will perform the search for us and displays the top results that match our need or query. This saves the tedious process of manually searching through the thousands or millions of total available data and also the huge amount of time that it takes to perform such manual search. Thus the importance and the usefulness of Search engines in our day-to-day life have become undeniable.

Exorbyte MatchMaker is a high performance fault-tolerant products search engine. Its fault-tolerant behaviour handles any minor spelling or the typographical errors made by the user while typing the query, such that it does not affect the effectiveness of the search engine by any significant amount. MatchMaker inherits this fault-tolerant behaviour by implementing Levenshtein algorithm to compare two strings and to find the distance between them. MatchMaker has in its database, the name of each product and its short description. So MatchMaker is essentially a search engine to find products. A product can be anything from electronic gadgets, clothing to grocery items. In the context of MatchMaker, a document pertains to a product name and its description and a document collection or corpus denotes a collection of product name and description pairs.

The aim of this project is to improve the effectiveness of the Exorbyte MatchMaker search engine by calculating and assigning appropriate weight for each term in the query and to effectively calculate the relevancy scores for each product in the case of a multi-word query. This project is done in collaboration with Mr Edward Ross of RossTech.

The system developed in this project calculates the relevance score of each word inside a document with respect to the query entered by the user in the MatchMaker search engine. The relevance score of a document is calculated depending on the relevancy of the words inside that document to the entered query. Also each word in the query is assigned different weight depending on the commonality of the word in the entire corpus collection. If a large number of documents have a particular word then it indicates that the word is not of much importance and that the presence of the word does not predict whether a document is relevant to the query or not. Such query words are given less weight. On the other hand, the query words which are not widespread but only present in few documents are given a higher weight as the mere presence of the word can give us valuable information about the relevancy of that document to the query.

The documents in which the query words co-occur as in the query are given higher scores than the documents in which the query words appear far apart. The order in which the query words occur in a document is not considered as it would not be of much help in the considered scenario.

Structure of the thesis

Chapter 2: It explains the literature review part. Here all the literatures related to the project are studied and reviewed. It also contains a brief description of the current implementation of the MatchMaker software.

Chapter 3: In this section, the design of the system to meet the aim of the project is discussed. Also the internal working of the system and the techniques which it uses are described.

Chapter 4: This section describes the implementation details of the project like the languages and tools used, the reason behind choosing them and also the MatchMaker interface is described.

Chapter 5: The structure of the test data, the process of collecting the test data, preparing it and the testing procedure used to test the project are explained in detail in this section.

Chapter 6: This chapter summarizes the results obtained, the reason behind them and also further discussion about the results.

Chapter 7: This section is about the future developments that could be made to the project so that the performance of the MatchMaker search engine improves even more.

Chapter 8: This section contains the conclusion part of the thesis. It summarizes the achievements made in the project and the objectives that were successfully met.

2. Literature review

2.1 Information Retrieval

Information retrieval (IR) can be defined as finding a document or material from a large collection of documents that matches the information need [1]. It is primarily a process of searching the collection of documents, databases, etc. When a user enters the query to be searched, the query is searched in the corpus collection and the document which is found to be most relevant to the query is returned to the user.

Evaluating the efficiency of an IR system

Let D be the set of documents in the collection and query $q = w_1, w_2,..., w_n$. An Information retrieval system aims at returning a subset of documents D^* of total document set D, such that for each $d \epsilon D^*$, the following probability is maximum [20].

$P(d \mid q, D)$ - (1)

The two most widely used metrics to calculate the correctness of an information retrieval system are *precision* and *recall*. The calculation of precision and recall requires the assigning of every single instance into one of the two categories, relevant and irrelevant. Precision is the calculation of the number of relevant matches from the total matches retrieved [2]. It is considered as the measurement of exactness of the information retrieval system.

$$\text{Precision} = \frac{Number\ \ of\ relevant\ \ documents\ \ \ retrieved}{Number\ \ of\ \ documents\ \ retrieved}$$

Recall is the fraction of total relevant documents that were retrieved by the information retrieval system [2]. It is the measurement of completeness of the information retrieval system.

$$\text{Recall} = \frac{Number\ \ of\ relevant\ \ documents\ \ retrieved}{Number\ \ of\ relevant\ \ documents}$$

In terms of true positives (*tp*), false negatives (*fn*) and false positives (*fp*), precision and recall are defined as [2]

$$\text{Precision} = \frac{tp}{tp\ + fp} \qquad \text{Recall} = \frac{tp}{tp\ + fn}$$

The various methods of calculating relevance scores are described in section 2.2. The relevant in context technique and the method to evaluate it are reviewed in section 3. The query expansion technique is briefly described in section 4 with its working procedure and uses. Section 5 describes various methods of combining scores from different information retrieval processes in order to improve the effectiveness of information retrieval. Section 6, being the final section, explains the current implementation of MatchMaker and the steps involved in it.

2.2 Calculating the relevance score of a word to the query

The relevance of a document to a particular query or topic is dependent on the relevancy of the terms within the document and the term frequency of the document. Various methods for evaluating the relevance scores are described below.

2.2.1 Term frequency – Inverse document frequency

The concept of *tf-idf* in Information Retrieval is described below. The value *tf-idf* helps us to find the document which is most likely relevant to the query entered by the user. The *tf-idf* method is an easy way to calculate the weight of each term in the corpus. The definition of both *term frequency* and *inverse document frequency* are given below.

Term frequency (*tf*) is the number of times a string appears in a particular document.

Document frequency (*df*) is defined as the number of documents in which the string under consideration appears.

Inverse document frequency (*idf*) can be calculated as [20]

$$\text{idf}_t = \log \frac{N}{df_t}$$

Where N is the total number of documents in the collection and df_t is the document frequency of the term t (a term is a single word or a string).

tf-idf is the product of *tf* and *idf*. Higher the value of *tf-idf* of a term, more relevant the term is to the document.

Mathematical framework:

The *tf-idf* concept makes use of the fact that the words which are present more frequently in one document and are less frequent or totally absent in other documents, are the words which have high level of relevancy with the document. And if a word is present in high frequency in all the documents then the word is almost neglected. For example articles and other common words which are relevant to the document but are present in large numbers in all documents are spotted and neglected.

Consider D to be the document collection of N documents which is to be searched, w to be a word and d to be an individual document such that $d \epsilon D$. Then *tf-idf* is calculated as [20]

$$W_d = \text{tf}_{w,\,d} * \text{idf}_w \qquad \text{- (2)}$$

Where,

$tf_{w,\,d}$ is the term frequency of the word w in the document d
idf_w is the inverse document frequency of the word w
w_d is the weight of the document d.

Consider the case when $N \sim df_w$, that is, the word is present in almost all documents in the collection. In this case the value of w_d will be smaller than the value of $tf_{w,d}$ as the value of the

log part becomes less than 1. Thus the relevancy of the word to the document decreases in this case. This is exactly the case when an article is considered. Thus this case helps us eliminate from taking most common words into consideration. On the other hand consider the case when $tf_{w,d}$ is high and also that the word *w* is present only in very few documents ($N >> df_w$). It is in this case that the word has high relevancy to the document. This document *d* is returned as most relevant document if the query contains the word *w* [20]. When a query has *i* words in it is executed, then the document set D^* having documents *d* in it, are retrieved which maximize equation (3).

$\sum_i w_{i,d}$ - (3)

Comparing effectiveness of *tf-idf* against *tf* based IR system

The testing was done on the collection of 1400 documents of LDC's United Nations Parallel Text Corpus by Juan Ramos [20]. The selection of documents was done randomly from 1988 database of UN. The query chosen is given below.

Query: "*the trafficking of drugs in Columbia*"

Case-sensitivity was also taken into consideration in order to incorporate maximum noise into the system. The *tf-idf* for all the documents were calculated using equation (3) and the top 100 documents which maximize the equation are retrieved as the result. The result retrieved above and the top 100 matching documents retrieved based on only term frequency for the same query are compared in table 1and 2 [20]. The documents retrieved based on term frequency are the documents which had the articles and other unimportant words from the query in large numbers in them. It can be noted that the top document retrieved based on term frequency has *tf-idf* score of just 1.83 and the top document retrieved based on *tf-idf* has a term frequency score of just 24. So considering only the frequency of the words from the query that are present in the document is not a good measure, but some measure of relevancy of the word under consideration should be incorporated into the calculation. That is why *tf-idf* was found to be a very efficient technique.

2.2.2 Weighted inverse document frequency (WIDF)

WIDF is an extension to the original *idf* concept. In *idf*, the fraction of total documents that contain the query string in them is only considered, but it does not take into account the number of times the query string is repeated in the document. Hence this new term *WIDF* was introduced [3]. In the calculation of *WIDF*, term frequencies are used.

Return Position	Document number	Sum $tf_{w,d}$	Sum w_d
1	64	139	1.83
2	879	136	4.52
3	1037	121	2.08
4	324	107	0.91
5	710	98	7.22
6	161	93	3.95
7	1175	87	0.24
8	402	86	5.13

Table 1: Top 8 documents returned which have the maximum $tf_{w,d}$ for the query considered and the w_d scores of the corresponding documents. (taken from Ref. [20])

Return Position	Document number	Sum $tf_{w,d}$	Sum w_d
1	788	24	28.09
2	426	72	26.73
3	881	56	23.96
4	253	43	19.16
5	1007	37	16.50
6	362	29	15.42
7	520	33	12.34
8	23	58	10.79

Table 2: Top 8 documents returned which have the maximum w_d for the query considered and the $tf_{w,d}$ scores of the corresponding documents. (taken from Ref. [20])

WIDF is calculated as

$$WIDF\ (t, d) = \frac{tf\ (t,d)}{\sum_{i \in D} tf\ (t,i)} \qquad \text{- (4)}$$

Where,

tf (t,d) is the term frequency of the term *t* in document *d*

i represents each document present in the document collection *D,* the denominator is the sum of term frequencies of the term *t* for each document in the document collection.

WIDF(t, d) is the Weighted Inverse Document Frequency of the document *d* for the term *t*.

Equation (4) is an alternative for equation (3) as this calculation includes the participation of term frequency in itself. And also the accuracy of the information retrieval system was found to increase in most cases when *WIDF* is used instead of *idf* [3].

Comparison between the query and the stored document collection can be made in two ways.

1. *Text-to-Text Comparison (TTC):*
   Here each query string is compared in turn with each term in the document and the decision is made based on the number of query string that matches.
2. *Text-to-Category Comparison (TCC):*
   All the individual terms in the document joined as one single term and this term is compared with the query to calculate the similarity.

TTC requires minimum of *N* comparisons where *N* is the total number of terms in all documents in the collection. On the contrary TCC requires only *D* comparisons in minimum where *D* is the total number of documents in the collection. It is a general fact that *N* is always much greater when compared to *D*, so calculating TTC always takes more computation than TCC [3].

2.2.3 Weighting term frequencies for relevancy estimation

Harter [11] modelled term frequencies within documents using Poisson distribution. An improvement to the Harter's model by Robertson and Walker [12] is concentrated here [23]. It has various assumptions, the first assumption is that every term is related to a topic and a document may or may not be relevant to the topic. This in turn implies that there is a set of both relevant and non-relevant documents for every topic. And by considering just one term within a document it is not possible to ensure whether the document is relevant or non-relevant to the topic. Hence the distribution of within document term frequency is a mixture

of two Poisson distributions, one representing the relevant set and the other representing non-relevant set of documents. The usage of Poisson distribution only makes sense if all the documents have same length. For now it is considered that all documents have same length and the case where documents have varying length is discussed later.

To represent a Poisson distribution one parameter *mean*, is needed. But this mean is different in the case of the two sets (relevant and non-relevant) considered here and so two different parameters are required to represent these two values. Since a mixture of two Poisson distribution is taken, a third parameter is also required which gives some intuition about the contribution of the two distributions in the mixture. In total three parameters are required to represent this mixed distribution [23].

Eliteness (*E*) of a term is a property by which the term becomes relevant to the topic referred by it. E_i refers to the eliteness of a term and $\bar{E}_i$ refers to the non-eliteness of a term. tf_i denote the frequency of the term T_i within the document. The two probabilities $P(tf|E)$ and $P(tf|\ \bar{E})$ can be found easily with the help of the two Poison mean values found previously. In the same way the probability of eliteness given likedness $(P(E|L)$ and $P(E|\ \bar{L}))$ can also be found. But according to Robertson, Van Rijsbergen and Porter [12], *TF* has direct dependence only on eliteness, so *tf* could be related to likedness only through eliteness. This relation is described by two equations [23]

$$P(tf|L) = P(tf|E)\ P(E|L) + P(tf|\bar{E})\ P(\bar{E}|L)$$

$$P(tf|\bar{L}) = P(tf|E)\ P(E|\bar{L}) + P(tf|\bar{E})\ P(\bar{E}|\bar{L})$$

These equations are very complex and it is hard to compute the values of the parameters. Robertson and Walker studied these equations and came up with a simpler equation with similar behaviour.

$$\text{wtf}_i = \frac{tf_i(k_1+1)}{k_1+tf_i}\ W_i \qquad \text{- (5)}$$

Where W_i is the weight of the term t_i, k_1 is a constant and wtf_i is the weighted term frequency score for the term t_i. A few important characteristic of this equation are, it becomes zero when $tf_i = 0$, the value of the equation increases monotonically with the value of tf_i and the equation has an asymptotic limit. When $tf_i = 1$, the equation becomes equal to w_i and its value increases with increase of tf_i but there is a limit to this increase. The constant k_1 is used for this purpose. If $k_1 = 0$ then the value of the equation becomes equal to W_i. The ideal value of k_i in the case of TREC was found to be 1.2 to 2. (TREC stands for Text Retrieval Conference. It is a well known test bed collection framework released by National Institute of Standards and Technology. 8 versions of it were released between 1992 and 1999. In total these frameworks contain 1.89 million documents and relevance judgement for 450 information needs [25].)

The weight of a term, W_i can be calculated as [26]

$$W_i = \log \left(\frac{r_i}{R - r_i} \Big/ \frac{s_i}{I - s_i} \right)$$

Where,

r_i is the number of relevant products which contain the word t_i
s_i is the number of non-relevant products which contain the word t_i
R is the total number of relevant products
I is the total number of non-relevant products

Normalizing the score based on document length (DL):

The previous Poisson approach assumed that documents are of same length. But in reality this is not the case, document length varies widely. Consider a situation where the document d_1 has length twice to that of d_2 and also has the term frequency twice to that of d_2. In this case both the documents are equally relevant and d_1 should not be ranked high just because it has higher frequency of the query term. This ensures the necessity of a document length normalization method.

A simple document normalization method involves dividing *tf* by *DL*. But this approach is not very effective in all cases as sometimes the document with higher length may be more relevant to the topic. So a different improved normalization scheme was needed. The document length normalization should work in such a way that the document with average length should have no effect of the normalization. So a new simple normalization method was formulated which is given below [23]

$$\text{Normalization factor (NF)} = \frac{DL}{AVDL}$$

Where *AVDL* is the average document length.

In the case of mixed distribution, the mixed normalization factor is calculated as

$$NF = (1 - b) + b \frac{DL}{AVDL},$$

Where *b* is a tuning constant ($0 \leq b \leq 1$).

Applying this mixed normalization to equation (5), the resulting equation is

$$\text{wtf}_i = \frac{tf_i(k_1+1)}{k_1 \times \left((1-b)+b\left(\frac{DL}{AVDL}\right)\right)+tf_i} W_i$$

When the constant *b* is set to 1, the mixed normalization becomes simple normalization. Smaller the value of *b*, smaller becomes the effect of normalization on *tf*. A value of 0.75 has been found to be ideal for most cases.

2.3 Relevant in Context

Relevant in Context retrieval is the retrieval of the document with a twist, which means not only the relevant document is retrieved but also the relevant information from within the document is identified [21]. Thus Relevant in Context retrieval is one step more than the traditional Information Retrieval process as it reduces the work of the user by identifying the relevant information by itself from within the document that is retrieved.

2.3.1 Retrieving relevant in context

The Initiative for the Evaluation of XML Retrieval (INEX) is an international campaign comprising of various organizations worldwide. The main aim of INEX is to evaluate the retrieval systems which retrieve relevant XML elements from a collection of articles. INEX 2006 had the task of ad-hoc retrieval of XML elements. In response to a query, the static document collection is searched and the relevant XML elements are retrieved. In general Information Retrieval task, simply the documents that are relevant to the query are retrieved, but in the case of XML retrieval each document has various XML elements within them and the relevant XML elements are to be retrieved from the relevant document [7]. Relevant XML elements must satisfy these two conditions

- They must have relevant information in them
- They must not have non-relevant information in them.

Consider a condition where two elements have same amount of relevant information about a topic. Here, the element retrieved would be the one which is smaller in length because the smaller length document will have less non-relevant information in it.

The ad-hoc XML retrieval task can be sub divided into four tasks which are explained below [7].

2.3.1.1 Thorough task

The main objective of this task is to return all the relevant elements from the document collection in the relevance order. When an element is judged to be relevant to a topic then its

parent XML element is also considered as relevant. So the relevant elements returned by this phase could contain various redundant elements. Ranking the relevant elements is the hardest part in this phase.

2.3.1.2 Focused task

The goal of this task is to return a ranked-list of elements for the query entered by the user, such that there is no overlapping between elements in the list. In this context overlapping means any text or information present redundantly in more than one element. In the context of XML elements, this stage ensures that no element present in the list is a child of some other element in the list.

2.3.1.3 Relevant in Context task

The ranked list of elements is now grouped based on the article which contains them. From the rank of the elements combined, the rank of an article is determined. This stage basically returns the ranked list of articles and the relevant information present inside them, in the logical order of their occurrence inside the article. Since the input to this stage contains no overlapping elements, ranked list of articles returned by this stage also do not contain any overlapping.

2.3.1.4 Best in Context task

This is the last stage in the Relevant of Context retrieval technique. In this stage the article are fetched in according to their relevancy rank and then the best entry point is found for every article. The best entry point refers to an element within the article which is most relevant to the query. Only one entry point per article is returned. No end point is mentioned.

2.3.2 Evaluating Relevant in Context:

Relevance in Context (RiC) combines the concept of document retrieval and XML element retrieval where not only the relevant document but also the XML element which contains the relevant information within that document is retrieved. Evaluation of RiC and the assignment of scores must be done based on two factors [21]

i) The score must correspond to the ranking of articles in the result list.
ii) The score must also correspond to the relevant information found within the document.

Score per article

This handles the second requirement mentioned above. It evaluates the relevancy of the information in elements within an article to the query given by the user. To find score per

article *precision* (the measure of how relevant is the information retrieved to the query entered by the user) and *recall* (the measure of fraction of relevant text that is retrieved) should be calculated first. After finding these two values, *F-Score* is calculated which ranges from [0, 1] for each article.

Let the function rel (.) represent the relevancy of the article or the element to the query given by the user and ret (.) represents the retrieved content. For each retrieved article *a*, precision and recall are calculated as

$$P(a) = \frac{|rel\,(a) \cap ret\,(a)|}{|ret\,(a)|}$$

$$R(a) = \frac{|rel\,(a) \cap ret\,(a)|}{|rel\,(a)|}$$

The F-Score of an article a is calculated as

$$F(a) = \frac{2\,.P(a).R(a)}{P(a) + R(a)}$$

The value of F-Score may vary from 0 (when article has no relevance to the context or none of relevance is retrieved) to 1 (when all the relevance components are retrieved).

Ranked list of articles

Consider a ranked list of articles with each having F-scores, F(a) ϵ [0, 1]. Now a generalized measure is needed and this is done directly by calculating generalized *Precision* and generalized *Recall*. Let *numrel* be the total number of relevant articles. Assume that the function *relart(a)* returns 1 if the article a contains some relevant information and returns 0 otherwise. Let r_{art} represent an article with rank *r* in the list. Then generalized Precision (*gP*) and generalized Recall (*gR*) are calculated as

$$gP(r_{art}) = \frac{\sum_{a=1\ldots.r\ art} F\,(a)}{r\ art}$$

$$gR(r_{art}) = \frac{\sum_{a=1\ldots.r\ art} relart\ \ (a)}{numrel}$$

Now, average generalized Precision (*AgP*) can be found by averaging *gP* at natural recall points where *gR* increases. Mean average generalized Precision (*MAgP*) is the mean of *AgP* scores per topic.

2.4 Query Expansion

In some cases the query entered may return very few relevant documents even though there may be a large number of actual relevant documents. This can be due to the reason that the terms in the query are not the exact words used in the documents but its synonyms. This problem is known as the word mismatch problem in information retrieval. Small length queries mostly face this problem. To overcome this problem the concept of Query expansion was introduced. The query is expanded by adding the words or phrases that have similar meaning to the words in the query. To find the words of similar meaning, a thesaurus is generally created and used. But the usage of a general thesaurus did not provide much improvement [13]. On the other hand, analyzing the corpus where the search is being done, provide an efficient way of creating a thesaurus.
One of the early algorithms to query expansion was devised by Sparck Jones [14]. He used clustering methods to find words that co-occurred with the query terms in the document and performed query expansion by adding these cluster of words to the query. A large number of new techniques were then found for query expansion. All of these techniques come under one of these two classifications, local or global analysis of documents in the corpus collection and are discussed here. Also a new method known as local context analysis is introduced which combines the features of both local and global analysis [17].

2.4.1 Global analysis:

It is one of the earliest techniques which provided improvement over the effectiveness of the retrieval system by using the idea of automatic expansion. This technique is used in INQUERY system in TREC evaluations [18] [19].
Before understanding the working of global analysis it is important to know the definition of two terms, *concepts* and *contexts*. In a simple way it can be said that all words in a document are called concepts and the words that co-occur in documents with concepts are called contexts. This method uses the contexts observed globally for a concept to find the concepts that are interrelated. The returned related concepts are stored in a database and called as concept base. Thus in global analysis method, the occurrences of words and the relationship between them are analyzed in the entire corpus as a whole and this knowledge is used to expand the query.

2.4.2 Local analysis:

The reason behind the name local analysis is that these techniques were modifications of the local feedback method [15] [16]. Local analysis observes only the top ranks that are returned by the retrieval system for a particular query. From this knowledge gained it expands the query. It is not appropriate to use local analysis in the case of a query which returns less relevant documents because most of the concepts in the context would be from non-relevant documents and so adding these words to the query would only deform it more.

2.4.3 Local context analysis:

Both the global and local analysis methods have their own advantages and disadvantages. Global analysis takes more computation time than the other but has the advantage of returning higher number of related concepts. But this large concept database requires additional memory space and there is also a high probability that the context returned may contain many low-relevant words. Both these factors reduce the effectiveness and efficiency of global analysis.

To overcome these disadvantages a new method called local context analysis was proposed [17]. This method copies some behaviour of global analysis and applies them to the locally retrieved document set. Not all the information from the retrieved documents is used by this method to construct the concept database, but only relevant passages inside these documents are considered.

The various steps involved in local context analysis are detailed below.

(1) A standard retrieval system is used to obtain the top passages that are relevant to the query. A passage is a text window which contains a certain predefined number of words in it. The main reason behind considering passages instead of the document as a whole is that normally a passage will concentrate on a single topic whereas a document may concentrate on more than one topic.
(2) The concepts from these retrieved passage list are ranked using the formula

$$bel(Q, c) = \prod_{t_i \in Q} (\delta + \log(af(c, t_i)) \; idf_c / \log(n))^{idf_i}$$

where

$af(c, t_i) = \sum_{j=1}^{j=n} ft_{ij} \; f_{c_j}$
$idf_i = max(1.0, \log10 \; (N/N_i) / 5.0)$
$idf_c = max(1.0, \log10 \; (N/N_c) / 5.0)$
Q is the query

c is a concept
ft_{ij} is the frequency of t_i in p_j
fc_j is the frequency of c in p_j
N is the total number of passages in the collection
N_i is the total number of passages containing t_i
N_c is the total number of passages containing c
δ is chosen here as 0.1 to avoid *bel* becoming zero.

The above formula can be used as a replacement for *tf-idf* measurement. The *af* part in the formula favours the concepts that occur more frequently with the query terms. The

idf_c part is used to find and neglect the concepts that occur too frequently in the corpus collection. Similarly idf_i addresses terms.

(3) Finally the *m* top ranked concepts are added to the query.

Local context analysis overcomes most of the shortcomings of global and local analyses. It expands a query in few seconds, but still it takes longer time than Phrasefinder, which is the global analysis procedure of INQUERY retrieval system. The concepts returned by all the three analysis methods for TREC4 query 214, are shown in table 3, 4 and 5. Stemming is performed on the concepts from each retrieved passage using the Porter Stemmer algorithm.

Query 214 of TREC4: "*What are the different techniques used to create self induced hypnosis*"

hypnosis	meditation	practitioners
dentists	antibodies	disorders
psychiatry	immunodeficiency-virus	anesthesia
susceptibility	therapists	dearth
atoms	van-dyke	self
confession	stare	proteins
katie	johns-hopkins-university	growing-acceptance
reflexes	voltage	ad-hoc
correlation	conde-nast	dynamics
ike	illnesses	hoffman

Table 3: The top 30 concepts returned by Phrasefinder for TREC4 query 214. (taken from Ref. [17])

In the step 1 of the query expansion process the relevant passages could be retrieved by relevant in context technique which is described earlier in section 3. Then the concepts in these passages are ranked using the formula in step 2. When a passage has more number of query words in it then the relevancy of that passage to the query is high.

From *tables 3, 4 and 5* it can be observed that the concepts returned by local context analysis are more coherent and relevant to the query terms.

hypnot	hypnotiz	19960500
psychosomat	psychiatr	immun
mesmer	franz	suscept
austrian	dyck	psychiatrist
shesaid	tranc	professor
hallucin	18th	centur
hilgard	11th	unaccept
19820902	syndrom	exper
physician	told	patient
hemophiliac	strang	cortic
ol	defic	muncie
spiegel	diseas	imagin
suggest	dyke	feburar
immunoglobulin	reseach	fresco
person	numb	katie
psorias	treatment	medicin
17150000	ms	franz-mesmer
austrian-physician	psychosomat-medicin	
hypnot-state	fight-immun	intern-congress
late-18th	diseas-fight	hypnotiz-peopl
ms-ol		

Table 4: The concepts returned by local feedback analysis on TREC4 query 214. (taken from Ref. [17])

hypnosis	brain-wave	ms.-burns
technique	pulse	reed
brain	ms.-olness	trance
hallucination	process	circuit
van-dyck	behavior	suggestion
case	spiegel	finding
hypnotizables	subject	van-dyke
patient	memory	application
katie	muncie	approach
study	point	contrast

Table 5: The concepts returned by local context analysis on TREC4 query 214. (taken from Ref. [17])

2.5 Combining results from multiple search agents

The technique which combines the ranked result of the queries from various search engines into a single improved ranked list is called *metasearch* [22]. This technique has been found to improve the effectiveness of an information retrieval system.

2.5.1 Relevance Score Normalization for Metasearch

The major problems in *metasearch* are to combine efficiently the ranking of documents produced by various search engines and to provide an optimal combined ranking. This problem usually consists of three steps [22]

a) *Normalization of scores*: In this step the relevance scores produced by different search engines are normalized and made comparable to each other.
b) *Estimation unknown scores*: In this step the relevance scores of documents which were not retrieved by the search engines.
c) *Combination the scores*: In this step the scores obtained from the previous two steps are combined to give the final score for a document.

These steps are later explained in detail.

Scores and ranks

A search engine searches for the query in the collection of documents and returns a ranked relevancy list which is in more-relevant to less-relevant order. This ranking is done based on the score calculated by the search engine. These scores given by various search engines for a document can be widely different and incompatible. To make the scores compatible for combining, the normalization methods are used. It is assumed that the normalization methods have access to the intermediate scores calculated by each search engine. The condition in which the scores are not available is discussed in [4, 5].

2.5.1.1 Normalization methods

The relevance scores given by various search engines are commonly real numbers in the interval $(-\infty, \infty)$ but they could be in different scales, different ranges and maybe widely distributed [22]. For each query given to the search engine S_i, it returns n ranked documents which are most relevant to the query along with their scores. Let $scr_S\ (a)$ denote the score assigned to the document a by the search engine S. And if $scr_S\ (a) > scr_S\ (b)$ then it means that the document a is more relevant than the document b.

The three desirable qualities of a normalization method are discussed below [22].

1) *Shift invariant:* Let R be the set of relevance scores returned by a search engine and R_c be the same relevance scores shifted with some additive constant c. Let the score of a be denoted by $scr(a) \epsilon R$, $scr(a_c) \epsilon R$ be the same score shifted by a additive constant c and $scr'(a)$ denote the normalized score of the document a. By shift invariant it is mean that $scr'(a)$ must be same as $scr'(a_c)$, in other words both the shifted and unshifted score must have the same normalized score.
2) *Scale invariant:* This is similar to shift invariant but in this case the normalization should be invariant of any scaling to the score with multiplicative constant.
3) *Outlier insensitive:* The normalization should not be over-sensitive on the score of a single document, if so then a single outlier can cause wide variation to the normalized scores.

Three commonly used normalization schemes are described below with respect to the qualities which were previously defined.

a) *Standard norm:* In this method, minimum is shifted to 0 and the maximum is scaled to 1. This method is shift and scale invariant but it is highly outlier sensitive.
b) *Sum norm:* In this method the minimum is shifted to 0 and the sum of scores is scaled to 1. This method is scale and shift invariant and also invariant of the maximum score given for a query. Though it is sensitive to the minimum score returned for a query, this does not affect the normalization process much because the ranking only returns a certain number of documents which scored high compared to other documents. So it can almost be said that sum norm is outlier insensitive.
c) *ZMUV norm:* In this method the mean is shifted to 0 and the variance is scaled to 1. This method is shift and scale invariant and also outlier insensitive.

2.5.1.2 Score estimation

In the standard and sum normalization methods, unretrieved documents are given a score of zero [6]. In the case of ZMUV normalization a score of 0 pertains to the mean score and so a score of -2 is applied to the unretrieved documents [22].

2.5.1.3 Score combination

Some of the most effective and simple combination algorithms were devised by Fox and Shaw [6]. A few among them are CombMIN, CombMED, CombMAX, CombSUM, CombANZ and CombMNZ. Among these algorithms the most effective ones are CombMNZ and CombSUM [22].

2.5.2 Analyses of multiple evidence combination

It has been shown that by combining the information retrieved by the information retrieval system for different representation of the same information need improves the effectiveness of the system. But the reason behind this phenomenon is not fully studied.

Turtle and Croft [8] combined the results of probabilistic and Boolean version of queries and also analyzed the result. The combination resulted in a significant improvement in the effectiveness of the information retrieval system. The conclusion deduced by them was that the belief about the combination returning more relevant information than the separate queries, is because each query returns different set of relevant documents was wrong and also that the result retrieved by a Boolean query is a subset of the result retrieved by the corresponding probabilistic query. Saracevic and Kantor [9] executed various different Boolean queries which were formulated for the same description of an information retrieval problem. They observed different documents being retrieved for different queries describing the same problem. An important deduction they made was that the probability of a document being classified as relevant, monotonically increased with the number of times the document appeared in the retrieved lists. JH Lee further studied this phenomenon and came up with a rationale that similar set of relevant documents might be retrieved in different runs but the non-relevant documents retrieved in different runs are not similar [10]. A run denotes the information retrieval process performed once with the given query. They justified this rationale with the help of two overlap coefficients, $R_{overlap}$ and $N_{overlap}$ which represent the degree of overlap among the document that were relevant and non-relevant in the retrievals obtained from two runs of an IR system. These coefficients are calculated as

$$R_{overlap} = \frac{\text{num\_of\_common\_rel\_docs} \times 2}{\text{num\_of\_rel\_docs\_in\_run 1} + \text{num\_of\_rel\_docs\_in\_run 2}}$$

$$N_{overlap} = \frac{\text{num\_of\_common\_nonrel\_docs} \times 2}{\text{num\_of\_nonrel\_docs\_in\_run 1} + \text{num\_of\_nonrel\_docs\_in\_run 2}}$$

$R_{overlap}$ becomes 1 if both runs retrieve the same relevant documents and 0 if they do not retrieve any relevant document in common. In a similar way $N_{overlap}$ gets a value of 1 if both runs retrieve the same non-relevant documents and 0 if they do not retrieve any non-relevant document in common.
Six retrieval results (*westp1, pircs1, vtc5s2, brkly6, eth001 and nyuir1*) were chosen from the TREC3 ad-hoc track and overlap coefficients were calculated for pairwise combinations of these six results and the values are tabulated in *table 6 [10]*. From the table it can be easily noted that degree of overlap between the relevant documents ($R_{overlap}$) is always very much greater than the degree of overlap between the non-relevant documents ($N_{overlap}$). This is known as the *unequal overlap property*. This property proves both the rationale of JH Lee and the deduction of Saracevic and Kantor stated earlier.

Fox and Shah devised several methods for combining evidence from multiple sources. The five methods devised by them are shown in table 7 [6]. Various analyses and experiments were performed and it has been found that CombSum worked well for TREC sub-collections like AP-1, AP-2, WSJ-1 and WSJ-2 [10]. But according to Saracevic and Kantor the combining methods should assign a higher rank to a document when it is retrieved in many runs as a relevant document. According to this fact CombMNZ should be more effective than the CombSum combining method, as it is biased towards the document which is returned by larger number of runs. So the pairwise combination of the same six retrieval sets from TREC3 ad-hoc track were again considered by JH Lee [10] but the difference between experimental setup of Fox and Shah [6] and JH Lee was that the former applied combination in the retrieval time and so no normalization techniques were applied. But JH Lee applied normalization to the scores obtained in each run and only then the combination was made. The result obtained by JH Lee is shown in the table 8. This table clearly shows that CombMNZ works more effectively when compared to other combination functions.

		westp1	***pircs1***	***vtv5s2***	***brkly6***	***eth001***
pircs1	$R_{overlap}$	0.7970				
	$N_{overlap}$	0.3620				
vtc5s2	$R_{overlap}$	0.7712	0.7562			
	$N_{overlap}$	0.3009	0.3035			
brkly6	$R_{overlap}$	0.7846	0.7813	0.7846		
	$N_{overlap}$	0.3522	0.3649	0.3272		
eth001	$R_{overlap}$	0.7706	0.7927	0.7686	0.8253	
	$N_{overlap}$	0.3260	0.3869	0.2936	0.4179	
nyuir1	$R_{overlap}$	0.7902	0.8210	0.7457	0.7562	0.7882
	$N_{overlap}$	0.3517	0.4360	0.3303	0.3238	0.4009

Table 6: Degree of overlap between relevant and non-relevant documents over the six retrieval results from TREC3 ad-hoc track. (taken from Ref. [10]).

CombMIN	The minimum of the individual similarities is taken
CombMAX	The maximum of the individual similarities is taken
CombSUM	The summation of the individual similarities is taken
CombANZ	(CombSUM / number of non-zero similarities is taken
CombMNZ	(CombSUM × number of non-zero similarities is taken

Table 7: The Combining functions proposed by Fox and Shaw. (taken from Ref. [10])

	CombMIN	CombMAX	CombSUM	CombANZ	CombMNZ
$westp1$ & $pircs1$	0.2780	0.3377	0.3586	0.3280	0.3604
$westp1$ & $vtc5s2$	0.2524	0.3355	0.3690	0.3100	0.3737
$westp1$ & $brkly6$	0.2641	0.3238	0.3490	0.3111	0.3525
$westp1$ & $eth001$	0.2560	0.3258	0.3502	0.3112	0.3522
$westp1$ & $nyuir1$	0.2566	0.3205	0.3505	0.3054	0.3559
$pircs1$ & $vtc5s2$	0.2469	0.3221	0.3463	0.2994	0.3520
$pircs1$ & $brkly6$	0.2595	0.3083	0.3275	0.2979	0.3305
$pircs1$ & $eth001$	0.2637	0.3115	0.3287	0.3045	0.3293
$pircs1$ & $nyuir1$	0.2663	0.3062	0.3259	0.3014	0.3273
$vtc5s2$ & $brkly6$	0.2456	0.3106	0.3390	0.2911	0.3431
$vtc5s2$ & $eth001$	0.2295	0.3108	0.3468	0.2876	0.3543
$vtc5s2$ & $nyuir1$	0.2532	0.3150	0.3341	0.3027	0.3334
$brkly6$ & $eth001$	0.2554	0.2900	0.3142	0.2876	0.3170
$brkly6$ & $nyuir1$	0.2378	0.3011	0.3354	0.2834	0.3402
$eth001$ & $nyuir1$	0.2564	0.2877	0.3214	0.2963	0.3230
average	0.2548	0.3144	0.3398	0.3012	0.3430

Table 8: Average precision of the five different combination functions over the pairwise combination of six retrieval sets from TREC3 with normalization done after each run. (taken from Ref. [10])

2.6 Current implementation of the search engine

Exorbyte MatchMaker implements Levenshtein Algorithm to compare the strings in query with the string in documents. This algorithm is used to match the words in the query to the words in the document even if the word is slightly misspelt in one of them.

2.6.1 Levenshtein algorithm

Levenshtein algotithm (also known as Edit distance algorithm) was devised to find the difference between any two strings. This algorithm was found by Vladimir Levenshtein in the year 1965 and so it was named after him. The difference between two strings is calculated in terms of the number of edit operations to be done on a string to transform it to the other string it is compared with. The number found is termed as the edit distance or Levenshtein distance between the two strings. Allowed edit operations are insertion, deletion and substitution.

(1) *Insertion*: Inserting a new character at a given position in the string.
(2) *Deletion*: Deleting the character at a given position in the string.
(3) *Substitution*: Deleting the character at a given position and inserting another character at the same position in the string.

		m	e	i	l	e	n	s	t	e	i	n
	0	1	2	3	4	5	6	7	8	9	10	11
l	1	1	2	3	3	4	5	6	7	8	9	10
e	2	2	1	2	3	3	4	5	6	7	8	9
v	3	3	2	2	3	4	4	5	6	7	8	9
e	4	4	3	3	3	3	4	5	6	6	7	8
n	5	5	4	4	4	4	3	4	5	6	7	7
s	6	6	5	5	5	5	4	3	4	5	6	7
h	7	7	6	6	6	6	5	4	4	5	6	7
t	8	8	7	7	7	7	6	5	4	5	6	7
e	9	9	8	8	8	7	7	6	5	4	5	6
i	10	10	9	8	9	8	8	7	6	5	4	5
n	11	11	10	9	9	9	8	8	7	6	5	4

Figure 2: The two strings 'meilenstein' and 'levenshtein' are compared with the Levenshtein algorithm. (taken from Ref. [24]).

Same cost is given to all the above three operations in the algorithm. The working of the algorithm is explained below with an example.

The steps in constructing the figure 2 are given here [24]. First an $m \times n$ matrix is drawn where m and n are the length of the two strings being compared. Filling of the matrix cells is done from top-left to bottom-right. A horizontal jump is made between cells when a character is deleted from that position in the string and a vertical jump is made from one cell to its neighbour when a character is inserted at that position in the string. A diagonal jump is made when a character is being substituted by another character at that position in the string. During each of these three operations, insertion, deletion and substitution, the cost is

increased by one and the new cost in filled in the cell to which the jump is made. But when the two characters being compared are similar then a diagonal jump is made without any increment to the score. The highlighted path in figure (2) is the least cost path to convert the string ‘meilenstein’ to ‘levenshtein’. This least total cost to perform the conversion in this case is 4.

There are two possible paths through the matrix from which one can attain the least total cost.

l	e		v	e	n	s	h	t	e	i	n
o	=	+	o	=	=	=	-	=	=	=	=
m	e	i	l	e	n	s		t	e	i	n

or

l	e	v		e	n	s	h	t	e	i	n
o	=	o	+	=	=	=	-	=	=	=	=
m	e	i	l	e	n	s		t	e	i	n

Figure 3: Levenshtein operations. (taken from Ref. [24]).

Where,
= denotes a match of character
o denotes the occurrence of substitution
+ denotes that a character has been inserted at that position
- denotes that a character has been deleted from that position

When a match occurs, one simply jumps to the next diagonal cell without changing the score. When a string is inserted at that position, a vertical jump from the current cell to the next cell is made, increasing the score by 1. When a string is deleted from that position, a horizontal jump from the current cell to the next cell is made with increase in score by 1. And when the current character is being substituted for a different character, a jump to the next diagonal cell is done with the same increase in score by 1. Thus finally one reaches the bottom-right cell of the matrix. According to the least total cost obtained from the Levenshtein algorithm a score between 0 and 255 is assigned to the word, where a score of 0 represents no similarity between the two words and 255 represents that the two words are the same.

2.6.2 Tokenization and indexing

Initially the document which contains a product name and its description is broken down into tokens by a process called tokenization. The separation character using for tokenization can be a space, comma or any other punctuation mark. Then an index is created representing each distinct token from every document in the corpus collection. Each entry in the index is of the form

{TOKEN, (Products which they describe)}

The query entered by the user is searched for in the documents collection and the matching documents are returned by the MatchMaker product in the following steps. Firstly, the query entered by the user is broken down into *tokens* by the process of tokenization which is described earlier. Secondly, these tokens from the query are compared with the token in each entry of the index file using the Levenshtein algorithm. If they match then the entry is extracted to a *result set*. The second step is repeated for each token in the query. Let n be the number of tokens in the query, then for every $token_i$ $(1 \leq i \leq n)$ a $resultset_i$ is obtained. Finally an intersection of all the n result sets is made and the scores are assigned according to the result of intersection.

2.6.3 Calculation of score

The score of a document for a particular query entered by the user is calculated using equation (6) below.

$$S_{d,\,q} = \frac{\sum_{w=1}^{w=n} S_w . l_w}{\sum_{w=1}^{w=n} l_w} \qquad \text{- (6)}$$

Where,

n is the total number of words in the query

S_w is the score of an individual word in the query which is assigned by using Levenshtein algorithm and comparing each query word to the word in the query. The value of S_w of a word ranges from 0 to 255 based on its similarity to the word being compared with.

l_w is the length of the word

$S_{d,\,q}$ is the score of a document d for a particular query q.

The top documents which receive highest score of $S_{d,\,q}$ are then retrieved and displayed to the user in response to the query.

3. Design

This section describes the solution that is proposed for the problem addressed in this project after reviewing all the related literature. The main criterion to be considered while choosing the appropriate technique to be used for this project is the execution time taken for the technique because the search result must be displayed to the user within milliseconds after the submission of the query. So a trade-off has to be made between effectiveness and execution time. Though using many different techniques discussed earlier like *tf-idf, WIDF* and *Weighted Term Frequency* in combination would increase the effectiveness of the search engine, it would also increase the response time of the search engine by a great amount. So a comparison has to be made in order to find out which one among the three techniques works best and retrieves the best result.

3.1 Proposed solution

The sequence of steps that is to be carried out in order to solve the problem is described below.

a) The query is received from the user. The query can be a single word or multi-word. Tokenization is then performed on the query to retrieve each separate word from it.

b) For each such word in the query, the following scores are calculated for every product in the corpus collection.

(i) Tf-idf score

The *tf-idf* score of each product with respect to the query word under consideration must be calculated. In other words the *tf-idf* score of a document is calculated assuming that the query only consists of the one word which is considered for this run.

Section 2.2.1 describes *Term Frequency* and *Inverse Document Frequency* in detail [20] and the formula to calculate *tf-idf* score [20] in the context of the our project is obtained from equation (2) as

$W_d = tf_{w,d} * idf_w$

Where W_d is the *tf-idf* score of a product description *d* with respect to the word *w* in the query.

$tf_{w,d}$ is the term frequency of the query word *w* in product description *d*, and

$$idf_w = log \frac{N}{df_w}$$

Where N is the total number of products in the collection and df_w is the document frequency of the query word w.

An intuition about the significance of various terms in the above equation is given below.

Significance of $tf_{w,d}$ part

The relevancy of a product to the word under consideration can be found by calculating the term frequency of that word in the product. If the number of time a word appears in the product is high then it indicates that the product is highly relevant to that word. Thus the $tf_{w,d}$ part gives us a direct intuition of the relevancy of a product to the word under consideration. This value should be calculated once for every product in each run.

Significance of idf_w part

The inverse document frequency value of a word becomes lower if the word is present wide-spread in large number of product descriptions in the corpus collection. So the idf_w part gives us a direct measurement of the weight to be assigned to a word in the query. This value should be calculated once for every run, i.e. it is calculated once for each query word. This can be more clearly explained with the help of an example.

Consider a query "The Continental Inn". Here the word 'The' is a common word which would be present in a large number of products and similar is the case for the word 'Inn'. But the word 'Continental' is not very common and it would be present in only a few products. So the inverse document frequency value of the word 'The' and 'Inn' would be very low and that of 'Continental' would be higher.

(ii) WIDF

The Weighted Inverse Document Frequency [3] score of each product with respect to the query word under consideration is calculated using the equation (4) as described in section 2.2.2.

$$WIDF\ (w, d) = \frac{tf\ (w,d)}{\sum_{i \in D} tf\ (w,i)} \qquad \text{- (7)}$$

Where,

tf (w,d) is the term frequency of the query word *w* in product description *d*

i represents each product present in the product collection *D*, the denominator is the sum of term frequencies of the query word considered for each product in the product collection.

WIDF(w, d) is the Weighted Inverse Document Frequency of the product *d* for the word *w*.

The significance of various terms in the calculation of *WIDF* score from the above equation is given below.

Significance of the numerator

The numerator of the *equation (7)* is the normal term frequency value of the query word in a product description. This significance of *tf(w, d)* has been already discussed before. This value differs for each product and for each word in the query.

Significance of the denominator

The denominator part of the *equation (7)* gives us a measure of how a word is widely spread in the product corpus collection. Its significance is similar to that of the inverse document frequency from the *tf-idf* technique. This value remains the same for a run, i.e. it is a constant for every word.

(iii) Weighted Term Frequency

The Weighted Term Frequency [23] score of a product is calculated using equation (5) described in section 2.2.3

$$\text{wtf}_i = \frac{tf_i(k_1+1)}{k_1+tf_i} \text{W}_i$$

Where,

W_i is the weight of the word w_i and
k_1 is a constant
tf_i is the term frequency for the word w_i.
wtf_i is weighted term frequency of the word w_i *for a product description*

The weight of a word, W_i can be calculated as [23]

$$W_i = \log \left(\frac{r_i}{R-r_i} \Big/ \frac{s_i}{I-s_i}\right)$$

Where,

r_i is the number of relevant products which contain the word t_i
s_i is the number of non-relevant products which contain the word t_i
R is the total number of relevant products
I is the total number of non-relevant products

A product is said to be relevant to a query when the product is present in the result set calculated by the Search engine which contains the top ranked products matching the query entered by the user. All the remaining products which were not retrieved as result to a query are deemed to be non-relevant product to that query.

The calculation of this weight of a word requires a training data which contains the products which were earlier displayed as a search result and was chosen by the user for some query. This makes the weighted term frequency technique a tedious process.

The weight of a word, W_i calculates the importance of a word based on the number of relevant and non-relevant products which contain the word [23]. This is based on the assumption that if a word is present in a large number of product descriptions which were earlier classified to be relevant, then it is most likely that this word is of high significance and so the word is assigned a higher weight. When a query word appears in large numbers in non-relevant product descriptions, the word is given a lower weight. Since this measurement takes as training data the real products which were chosen by the user from the search result displayed by the search engine, this is expected to outperform the other two techniques which were discussed earlier.

c) The total *tf-idf*, *WIDF* and weighted term frequency score of a product is calculated by cumulating the score obtained for the product by that technique for each query word.

d) The score assigned by the original MatchMaker for the products with respect to the query entered by the user is then found. To find this MatchMaker score an interface to the MatchMaker is used through which we can pass the query as input and in return the scores of the products are received as output.

e) The MatchMaker score calculated from the previous step is now added with the *tf-idf*, *widf* and weighted term frequency scores calculated earlier and the new *tf-idf* + MatchMaker, *widf* + MatchMaker and weighted term frequency + MatchMaker scores are found. This step is performed because it was advised by the project supervisor from RossTech that this project is essentially an extension to the existing MatchMaker implementation.

f) In the case of a multi-word query, if the words in the query occur next to each other in the product in the same way as in the query, then the score of that product is increased by a marginal value (0.5). The marginal value added, increases the score considerably, thus giving the product an edge over other products in which the query words does not co-occur and also care should be taken that the marginal value does not increase the score to a very high value. So a safe value of 0.5 was chosen.

This could be better understood when explained with an example. Consider the query “Queen Palace”. Consider product A to be “Queen Palace – A great place to visit” and product B to

be “King Palace – Now owned by Doctor Queen”. Both product A and B contain the same query words but still the context changes in B as the query words does not co-occur in it. So to handle this situation, product A is given a marginally higher score than product B.

g) The rank of the products is adjusted on the basis of the historical data describing which products were earlier chosen by a user. The click-through of the users is tracked to obtain the following pair of data, query and product actually chosen by the user from the result displayed for that query. These pairs are taken as training sets and the products which were chosen earlier by a user independent of the query are given a slightly higher score over the other products which are not yet chosen by any user.

This step is done basically to assign a higher score to the products which are favoured by users the most and also considering the fact that a new user is more likely to choose a product which was already chosen by him or some other user earlier.

h) Finally the products are now ranked based upon their final score and the top ranked products with highest scores are displayed to the user as the search result.

4. Implementation

4.1 Languages and tools used

4.1.1 C++

C++ was chosen as the programming language primarily because the MatchMaker was implemented using it. It is an extremely fast and is best suited for general purpose programming. It is due to these reasons that C++ is used as the back-end in many applications. It is also the best choice for time constrained applications like search engine where one desires to get their output within few moments after entering the query.

4.1.2 Microsoft Visual C++ 2010 Express

Visual C++ 2010 Express is a free and powerful Integrated Development Environment that Windows developers can use for developing various C++ applications. It was preferred over Eclipse because of its simplicity and easier configurability.

4.1.3 Exorbyte MatchMaker 4.2

The latest stable version of Exorbyte MatchMaker which is 4.2, is used to find the score assigned to the products for the query entered by the user. Finding this MatchMaker score is necessary in the project as this project is primarily an improvement over the original MatchMaker application.

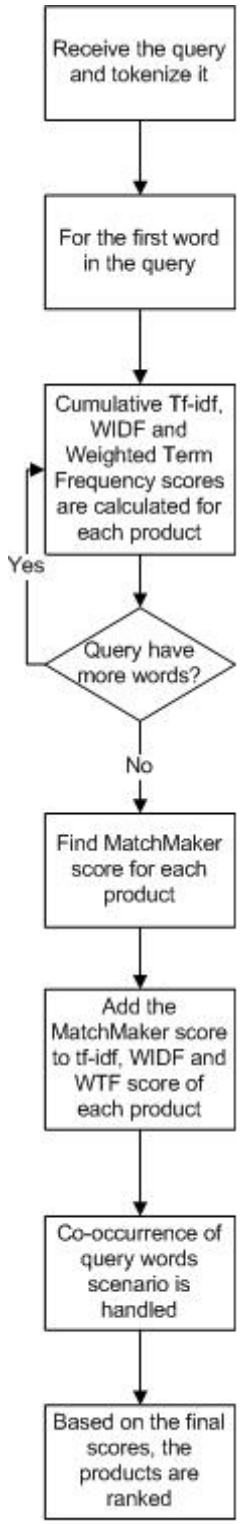


Figure 4: Flow chart representation of the design of the project where a rectangular box represents a process, rhombus represents a decision making process and arrows denote the flow of control.

MatchMaker Interface (MMI)

The C++ MatchMaker interface is a client stub library to access the MatchMaker server [27]. The interface uses a stateful MatchMakerSession object. The initial step for connecting to the MatchMaker server is to create a MatchMakerSessionInterface object. Then the MatchMakerSession object is created and initialized by passing the MatchMakerSessionInterface object to the init method of MatchMakerSession [27]. More than one MatchMakerSession object can be created using a single MatchMakerSessionInterface object. The MatchMakerSession object is then queried which returns result in the form of MatchMakerResult objects [27]. Finally the MatchMakerResult object can be inspected to obtain the results for the query made.

Input to the MatchMaker

The MatchMaker in order to answer a query requires two input files. The first input file containing the product name and description. The name and description of each product is written in a separate line, thus each line represents a different product. The second input file is an index file which contains all the distinct words from the first input file and also the list of product id in which those words appear. A C++ code is written to build this input file for MatchMaker.

4.1.4 Java SE Development Kit 6 Update 26

Java Development Kit (JDK) is a toolkit to develop, compile and execute Java programs. Some of the data processing and validation for the test data was done using Java code. The structure and processing of these test data are explained in detail in section 5.

4.2 Coding style and documentation

This project is intended to be integrated with MatchMaker, which is a commercial products search engine. So the coding structure and formatting was done to the standards of commercial software with adequate documentation. The documentation and design of the program structure is made as simple as possible so that even a person with moderate knowledge in programming can understand it easily.

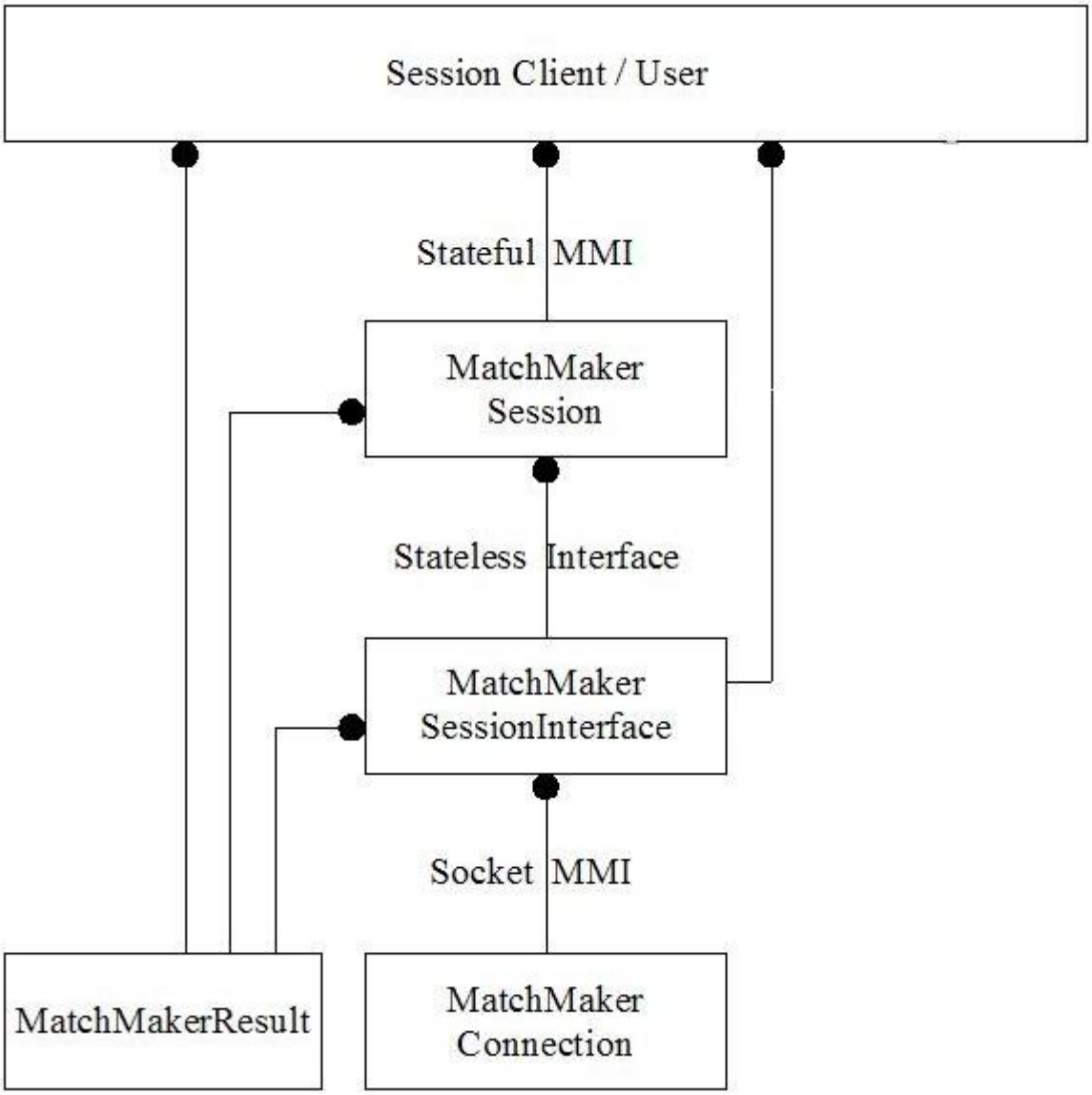


Figure 5: MatchMaker Interface. Each box represents an object or instance and each line represents the interaction between them where the box at the shaded circle end of the line uses the box at the other end of the line. (taken from Ref. [27]).

5. Test data

Testing is a very important part of development; it helps to find the accuracy of the project developed and also to find any faults or errors. So selecting an appropriate testing method is very crucial for evaluating any product.

5.1 Test data structure

To check the accuracy of the scoring and ranking techniques used in this project the evaluation parameter used was *the rank assigned by that technique to the product which was*

actually chosen by the user previously from the list of results displayed by the original MatchMaker for the query entered.

The structure of the test data used is very simple. It contains the query entered by the user and the corresponding product which was chosen by the user. Some pairs of example query and product chosen are shown in *table 9*. These data were taken from REAL product data of Konstanz, a German based company. 200 such pairs were used for testing and comparing the techniques used in the project. Efforts were made to collect more data but the log files from which these pairs could be extracted were erased by an accident in RossTech. In order to compensate, many different analysis were performed from the results obtained by the test data in hand.

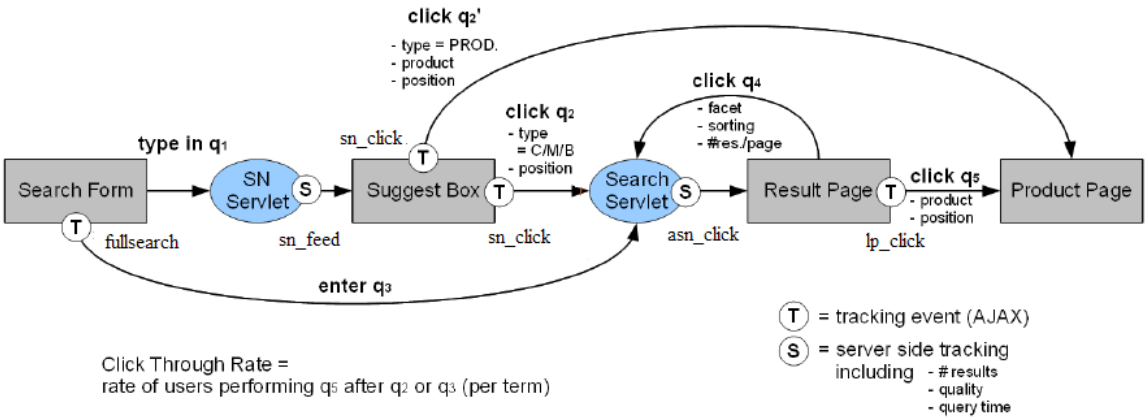
QUERY	**PRODUCT CHOSEN**
Popolini	Popolini PopoWrap
Stilleinlagen	Philips AVENT Stilleinlage waschbar
Avent	Philips AVENT Flasche PP
Wickelbody	Petit Bateau Wickelbody Kurzarm
Lodger	Lodger Wrapper Baumwolle
Trainers	ImseVimse Trainers

Table 9: A few sample pairs of query and the corresponding product chosen by the user.

5.2 Collection of test data

The collection of test data described in the previous section requires tracking the click-throughs of the user. In this process the mouse click events made by the user on the MatchMaker Graphical User Interface are tracked by a JavaScript code. This collection of test data is performed by Exorbyte, but the process used for this purpose throws some light into the working of the MatchMaker software and is explained here.

In the user activity diagram in *figure 6*, a rectangular box represents the entity in which a user can perform an action, oval shaped box represents a servlet and arrows represent the user actions which are tracked, with the name of the user action mentioned above the arrow. User actions such as suggest feeds, fullsearches and ASN click) [28] causes an internal search and they are tracked with the help of server-side tracking servlets. Suggest feeds are the suggestions displayed to the user as and when the user types any character into the search box.ASN clicks are the after search navigation clicks which could be the click on a brand name, sorting the entries on a page, etc. User actions such as suggest clicks or click on the result page which does not cause an internal search are tracked using the browser-side servlets written in AJAX code [28].



Terms used

- q1 - suggest query/feed
- q2/q2' - suggest click
- q3 - fullsearch/direct search
- q4 - ASN (after search navigation) click
- q5 - result page click
- sn_feed (a user enters something and the suggest pops up)
- sn_click (click in the suggest window)
- fullsearch (submitting a query via the search button)
- lp_click (click on a product on a result page)
- asn_click (after search navigation: click on a facet, sort, paging, number of entries per page)

Figure 6: User Activity Diagram. (taken from Ref. [28]).

Tracking is performed in the following format for all actions of the user. The various attributes in the tracking format are mentioned below [28].

- Time stamp (mandatory)
- User ID (time stamp hash, stored in a cookie)
- IP address of the user (to be able locate the user)
- Action type (mandatory)
- Entered term (full search or suggest click (content of the search field))
- Clicked term (suggest click or facet click)
- Clicked term type:
    - for suggest clicks: article, category, manufacturer, brand
    - for ASN clicks: *cf* (category facet), *bf* (brand facet), *pf* (price facet), *p* (paging), *n* (number of entries per page), *s* (sort), empty for first display after a fullsearch or if a facet is deleted

• Click position (suggest click or result page click)
• Product ID (suggest click on a product or result page click)
• Number of results
• Quality of the best result
• Query time

Possible values for the action type are:

• sn_feed (a user enters something and the suggest pops up)
• sn_click (click in the suggest window)
• fullsearch (submitting a query via the search button)
• lp_click (click on a product on a result page)
• asn_click (after search navigation: click on a facet, sort, paging, number of entries per page)

This log file created is then processed to find the entries with action type value as lp_click as this represents that a user has clicked on a product from the result page. From such entries the entered term and clicked term are separated. Now the clicked term is the query entered by the user and the clicked term is the product chosen by the user for that query from the result page. Thus the query and the product chosen by the user for that query pairs are obtained as the test data.

5.3 Testing procedure

From the test data pairs created, the text from the 'Query' column is retrieved and given as the query to all the three techniques used in the project, tf-idf, widf and weighted term frequency and also to the original MatchMaker. The result for this query is the ranked products, which is displayed for each of the technique. Now the rank held by the product under the 'Product Chosen' column of that query is found for all the techniques and noted down. In the case of the weighted term frequency which requires training data, i.e., the products chosen by any user earlier, all the products under the 'Product Chosen' column of the test data are provided as training data except the product which was chosen for the query which is being tested. This process is repeated for all pairs of data in the test data and finally the average rank assigned by each of the technique to the product which was actually chosen by the user is calculated. Based on this average value calculated various techniques used in the project are evaluated and compared.

6. Results and discussion

The results observed from the project and the discussion on those results is summarized here.

6.1 Term Frequency – Inverse Document Frequency

The effectiveness of the *tf-idf* technique was compared with the original MatchMaker which only uses the Levenshtein algorithm. The comparison was based on the rank assigned by the technique to the product which was actually chosen by the user for that query. The average value of this rank assigned by *tf-idf* technique and the original MatchMaker software for the collected test data are depicted and compared in the graph in the *figure 7*.

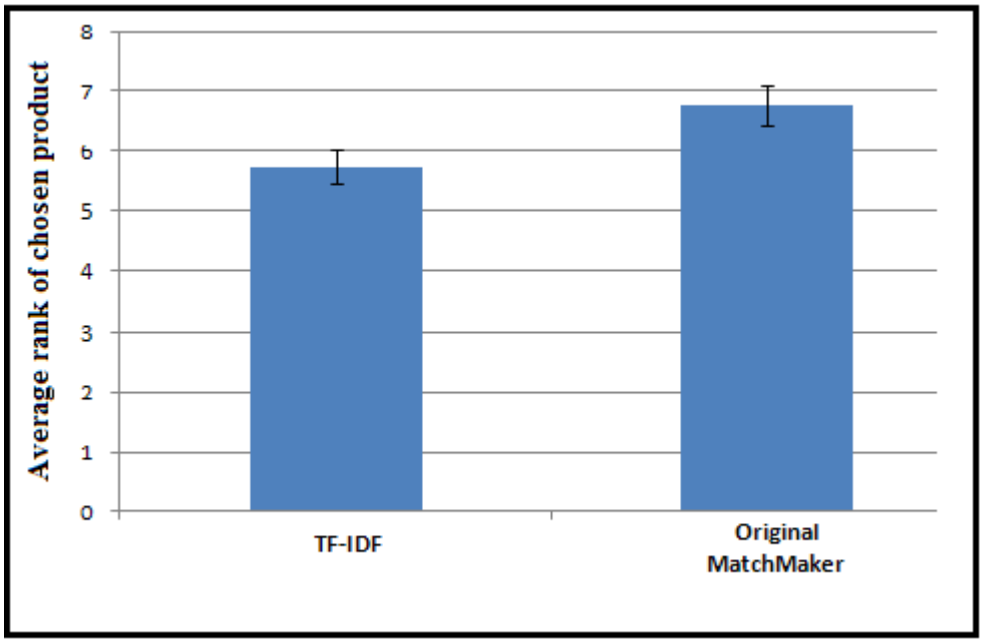


Figure 7: Comparison of tf-idf and original MatchMaker software, with error bars.

From the graph in *figure 7*, it can be noted that the effectiveness of the MatchMaker software is found to increase when *tf-idf* technique is used.

Statistical analysis

Various statistical analyses were performed on the results obtained to find the level of accuracy of the results. The first analysis made was using the error bars. The error bars in a graph represent the degree of uncertainty in the measurement or calculation. The error bars in *figure 7* were drawn using the standard error. It can be noted from the above figure that the error bars of the average rank assigned by *tf-idf* and the original MatchMaker do not overlap indicating that the difference between them is statistically significant. In other words, the improvement made over the original MatchMaker using the *tf-idf* technique is statistically significant.

The second statistical analysis performed was the two-tailed paired t-test. Two-tailed paired t-test was preferred over one-tail because the mean might vary in both positive and negative direction. The result of the t-test between *tf-idf* technique and the original MatchMaker was calculated to be 0.00504 which is much lesser than the critical value of 0.05 and so it can be concluded that the previous test which was performed to compare *tf-idf* method with the original MatchMaker technique is significant.

Discussion on the observed result

The significantly improved performance of the tf-idf technique over the original MatchMaker software can be attributed to the inverse document frequency term in it. The original MatchMaker assigns equal weight to all the words in the query and rank the products based on only the term frequency of the query words in them. But for efficient ranking, appropriate weights must be assigned to each query word based on the commonality of the word. The inverse document frequency part of the tf-idf technique assigns this term weight for each word in the query and gives an edge for the tf-idf technique over the original MatchMaker which only considers the term frequency.

6.2 Weighted Inverse Document Frequency

The effectiveness of the Weighted Inverse Document Frequency technique with that of the original MatchMaker was compared on the test data and the resultant graph is in *figure 8*.

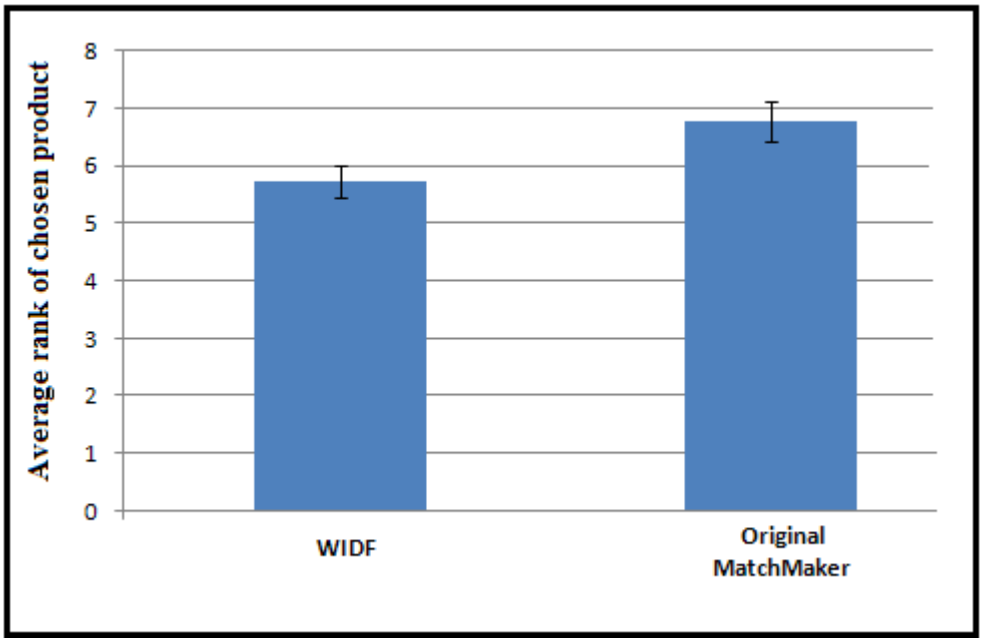


Figure 8: *Comparison of WIDF and original MatchMaker software, with error bars.*

It can be clearly noted that the widf technique performs better when compared to the original MatchMaker software. Various analyses were made on this result and are discussed below.

Statistical analysis

The error bars in the graph from *figure 8* were drawn based on the standard error measurement for both *WIDF* method and the original MatchMaker software. The two error bars are at a safe distance apart and it can be deduced that the difference in performance of widf technique and the original MatchMaker software is statistically significant and that the widf method clearly outperforms the original MatchMaker software.

The next statistical analysis performed on the result was the two-tailed paired t-test. The result of the t-test between *WIDF* technique and the original MatchMaker was calculated to be 0.004474 which proves that the previous testing experiment is significant and so the result obtained from it is authentic.

Discussion on the observed result

The denominator part of the *WIDF* score calculation makes it perform better than the original MatchMaker software. The denominator part is the sum of the term frequency of the query word in all the products of the corpus collection. *WIDF* could be defined as the ratio of the term frequency of the word in the document considered to the term frequency of the same word in the entire corpus collection. Thus the score assigned by *WIDF* technique to a product also depends on the other products in the corpus collection which contain the same query word and the degree to which the score is affected depends on the term frequency of the word in those documents. Whereas the original MatchMaker only considers the product for which the score is being calculated and assigns the score to that product independent of other products. This explains the reason behind the significant improvement made by the *WIDF* technique over the original MatchMaker software.

6.3 Weighted Term Frequency

The weighted term frequency technique and the original MatchMaker technique were compared based on the average rank assigned by the technique to the product which was actually chosen by the user for the query and the graph depicting the values is in *figure 9*. From the graph it can be clearly seen that the effectiveness of the weighted term frequency technique was better than the original MatchMaker.

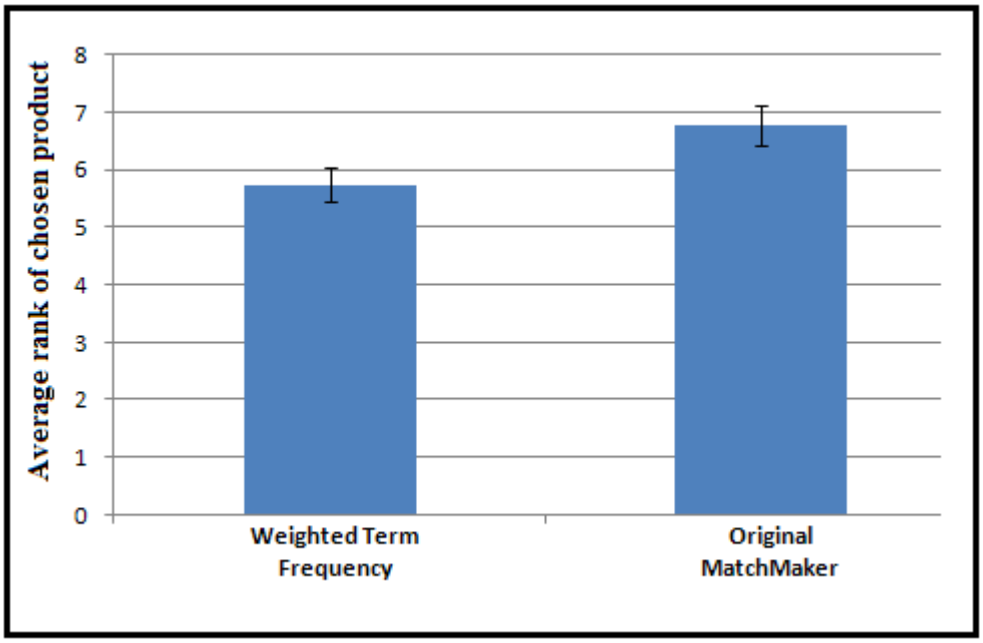


Figure 9: *Comparison of Weighted Term Frequency technique and original MatchMaker software, with error bars.*

Statistical analysis

Though the result obtained show that the weighted term frequency technique performed better than the original MatchMaker based on the evaluation criteria, statistical analysis has to be made to prove that the improvement made is statistically significant. The first statistical analysis made was drawing the error bars. The error bars in *figure 9* were drawn using the standard error measurement. The error bars in the below graph show that the difference between the two techniques is statistically significant which implies that the improvement made by the weighted term frequency technique over the original MatchMaker software is also statistically significant.

The second important analysis made on the result was the two-tailed paired t-test. The probability value obtained from the t-test was 0.004126 which proves that the test results were certain and significant.

Discussion on the observed result

The reasons for the weighted term frequency technique to outperform the original MatchMaker software are discussed here. The main reason is due to the presence of the *term weight* part in the weighted term frequency score calculation. *Term weight* is calculated for each word in the query and it depends on the number of times the word appear in the relevant

and non-relevant product descriptions from the entire corpus collection. Higher the number of times a query word appears in relevant product descriptions, higher is the term weight assigned to the word and vice-versa. Thus an appropriate relative score is assigned to a product and this helps in improving the final ranking of products according to the relevancy of the product to the query entered. This improvised ranking in the case of weighted term frequency allows it to gain advantage over the original MatchMaker and to outperform it.

6.4 Comparison between *tf-idf, WIDF* and Weighted Term Frequency

The results obtained and discussed above prove that all the three techniques namely, Term Frequency – Inverse Document Frequency, Weighted Inverse Document Frequency and Weighted Term frequency perform better than the original MatchMaker software. But further analysis is required to find which one among the three works best. The graph in *figure 10* compares all the three techniques based on the same criteria used in previous comparisons, the average rank assigned by the technique to the product which was chosen by the user for a query.

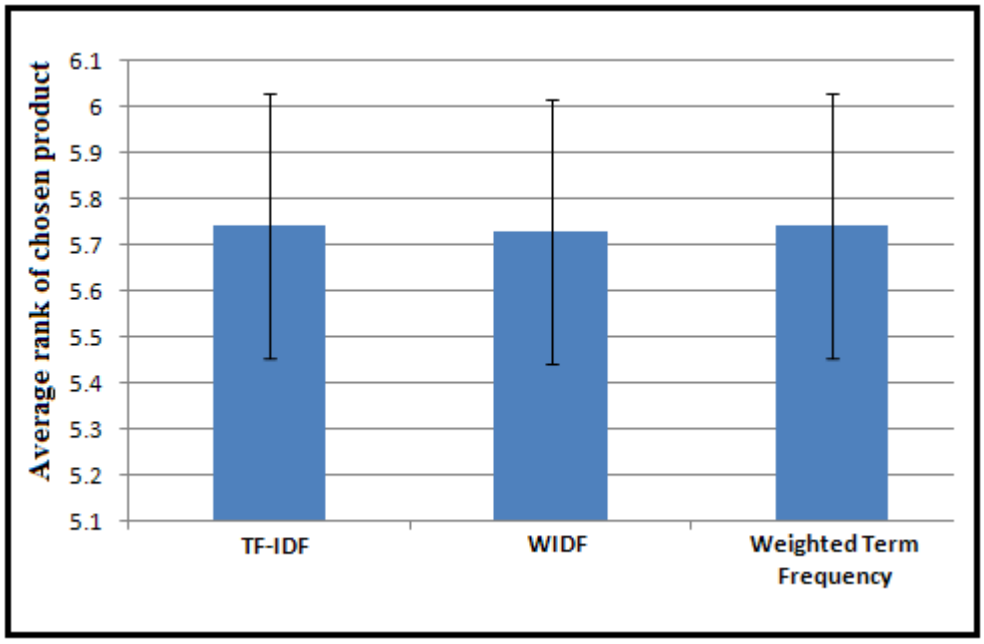


Figure 10: *Comparison of tf-idf, Weighted Inverse Document Frequency and Weighted Term Frequency technique, with error bars.*

Though from the graph in *figure 10* it can be deduced that weighted inverse document frequency performs better than both *tf-idf* and weighted term frequency methods, the difference in performance is very small.

Statistical analysis

The performance of all the three techniques namely, *tf-idf*, Weighted Inverse Document Frequency and Weighted Term Frequency were almost similar with Weighted Inverse Document Frequency leading the race. This scenario makes the statistical analysis over this data interesting.

The error bars drawn in the graph in *figure 10* show that the differences in performance of the three techniques are within the standard error measurement and so the differences are not statistically significant. This implies that the differences seen in the graph could be due to biased test data, specific biased test conditions or due to error in testing procedure and that it is not certain.

Two-tailed paired t-test was also performed to compare the performance of the three techniques. Paired t-test was performed on three pairs, *tf-idf* and Weighted Inverse Document Frequency, Weighted Inverse Document Frequency and Weighted Term Frequency and Weighted Term Frequency and *tf-idf*. But in all the three cases the t-test proves that the tests were not significant. The t-test values were calculated to be 0.3204, 0.8201 and 1.0 respectively which are much higher than the tolerable value of 0.05, so the test experiment and the results are not significant.

Discussion on observed result

The term which is present in the calculation of score for all the three techniques, *tf-idf*, Weighted Inverse Document Frequency and Weighted Term Frequency is the *term frequency*. The part of the calculation which differentiates the three techniques is the *term weight*. All the three techniques use different methods to calculate the *term weight* (also known as *term relevance*) of a word.

Tf-idf and Weighted Inverse Document Frequency use almost similar methods to calculate the *term weight* of a query word. *Tf-idf* considers only the fraction of total number of products which contain the particular query word in its description. On the other hand, Weighted Inverse Document Frequency goes a step further and also considers the number of times the word appears in each product description in the entire corpus collection. This term frequency of the word in other product descriptions in the corpus collection is not considered by the *tf-idf* technique and so it is natural that Weighted Inverse Document Frequency is expected to perform better than the *tf-idf* method. In the comparison made as a part of the project it is proved that Weighted Inverse Document Frequency performs better than *tf-idf* for the test data used.

Tf-idf and Weighted Term Frequency also use almost similar technique to calculate the score of a product and are compared here. As described earlier *tf-idf* assigns term weight based on the number of products from the entire corpus collection which contain the word in its description. Weighted Term Frequency extends this concept a bit further and takes into account the number of relevant and non-relevant products from the corpus collection which contains the word in its description. It then assigns a higher weight to a word which is present in large number of relevant product descriptions and lower weight to the words which appear in large number of non-relevant product descriptions. So Weighted Term Frequency is generally expected to outperform the *tf-idf* method but this is true only when the training data used for the calculation of *term weight* in Weighted Term Frequency complements it. Weighted Term Frequency requires extensive training data to operate effectively, like the relevance information of every product description in the corpus collection [26]. But it is very hard to obtain this extensive data. This small chance of improvement comes with the cost of extra time required for training the system in order to find whether a product is relevant or not. The comparison between Weighted Term Frequency and *tf-idf* in this project show that the performance of both techniques are identical for the training and test data used. But when a large number of training data are used to train and the number of test data is also increased, Weighted Term Frequency would most likely perform better than the *tf-idf* method.

Weighted Term Frequency and Weighted Inverse Document Frequency are both advanced forms of *tf-idf*. The comparison between the two suggests that Weighted Inverse Document Frequency performs slightly better than Weighted Term Frequency. The reason behind this observed result is that the *WIDF* method considers the term frequency of the word in all the product descriptions in the entire collection whereas Weighted Term Frequency only considers the number of relevant and non-relevant which contain the word. The score calculation in *WIDF* takes into account the exact distribution of the word in the entire corpus collection but Weighted Term Frequency only considers a rough distribution of the word while calculating the score. This gives *WIDF* a slight edge over Weighted Term Frequency and allows to it perform better. One more parameter that differentiates the two techniques is the range of the score calculated. In the case of *WIDF* the range of the score is [0, 1] but for Weighted Term Frequency the range of the score varies from positive to negative with no exact boundary on either side.

General discussion on the effect of the new techniques on MatchMaker engine

One important point to note is that usage of the techniques such as *tf-idf*, Weighted Inverse Document Frequency and the Weighted Term Frequency does not improve the *precision* or *recall* nature of the search engine, but it only improves the effectiveness by refining the ranking of the products which are retrieved for a query. An efficient search engine must not only retrieve all the matching results to the query but also rank the results appropriately according to the relevancy to the query. So it can be safely declared that the techniques used in the project improve the effectiveness of the MatchMaker search engine.

Another important thing is that all the three techniques used in this project could improve the effectiveness of the MatchMaker search engine only in the case of a multi-word query and for a single word query their performance will be similar to the original MatchMaker software. This is because in a single word query, term weighting is not required and both the score and the ranking of the products only depend on the term frequency of the query word in them.

7. Future works

Though the intended target was attained in the project as a significant improvement was made to the effectiveness of the MatchMaker software, there are still room for improvements. Some important techniques which can be integrated with this project to further improve the effectiveness of the MatchMaker search engine are described here.

Query expansion technique [14] which broadens the search by finding the synonym terms of the query words from the corpus collection and adds them to the search query, is one among the techniques that would increase the performance and the retrieval feature of the search engine. These synonyms are the words which are found to frequently co-occur alongside the query word in the product descriptions. This technique was explained in more detail in *section 2.4*. Adding this technique to the search engine increases its response time by a considerable amount, so it is not good to use this technique in all cases. Query expansion technique could be used to find more search results in the case where the search results returned by the search engine is very less or when it does not contain the result that the user is looking for. So in the case where less number of search results is retrieved for the query, the retrieved results are displayed to the user and also the question whether the user wants to find more related products. If yes then the query expansion technique is used and more results are retrieved and displayed to the user. Implementing this way, the search engine could maintain its low response time and time efficiency.

The Weighted Inverse Document Frequency and the Weighted Term Frequency techniques are both evolved forms of the basic *tf-idf* method. Weighted Inverse Document Frequency considers an extra parameter than *tf-idf*, the term frequency of the query word in all the product description in the corpus collection. Weighted Term Frequency tries to improve over the *tf-idf* technique by taking into account the number of relevant and non-relevant product descriptions in the entire corpus collection containing the query word. But a better method can be devised by taking into account the extra parameters considered by these two techniques, Weighted Inverse Document Frequency and Weighted Term Frequency. The new parameters must be combined in such a way that it improves the effectiveness of the system and must be fine tuned by testing with various kinds of test data.

More comparison and analysis can be made with the three techniques, tf-idf, Weighted Inverse Document Frequency and the Weighted Term Frequency, if more numbers of test

data could be obtained. Averaging over a large test dataset the exact ranks held by the three techniques in terms of effectiveness can be deduced.

8. Conclusion

With the constantly increasing storage of data, Search engines have attained an undeniably significant status and will maintain its status. High volume of data tends to degrade the performance of the Search engine as a large amount of data has to be processed and searched to find the search result. This increases the response time of the Search engine. To encounter this issue more effective Search engines are required. The techniques used in this project namely, *tf-idf*, Weighted Inverse Document Frequency and the Weighted Term Frequency, were all found to significantly improve the effectiveness of the Exorbyte MatchMaker Search engine. Improvement in effectiveness was achieved by assigning appropriate weights to the query words and calculating relevancy score efficiently for each product according to the query entered by the user. The extent to which they improve the effectiveness and the reason behind this improvement were also discussed.

Comparison was made between *tf-idf*, Weighted Inverse Document Frequency and the Weighted Term Frequency, to find which technique suit the MatchMaker search engine best. Results from the project reveal that Weighted Inverse Document Frequency improves the effectiveness of the MatchMaker engine by assigning better scores to the products than the other two techniques. But the comparison between the three techniques show that the difference in performance between them is not significant which implies that any technique can outperform the other two when the scenario and the training and test data used shows some bias towards it.

Weighted Inverse Document Frequency has a few advantages which makes it a better choice over *tf-idf* and Weighted Term Frequency techniques. It is simple and straight-forward when compared to Weighted Term Frequency method and the score calculated always remain in the range 0 and 1, in contrast to the *tf-idf* and Weighted Term Frequency method where the score has no limited boundary.

The importance and the role of various parameters like term frequency, inverse document frequency and term weights in the relevance score calculation were thoroughly studied. The advantages and disadvantages of these parameters were discussed. Also the area for improvement in the Weighted Inverse Document Frequency and the Weighted Term Frequency techniques were highlighted.

In conclusion, the main aim of the project, to improve the effectiveness of the Exorbyte MatchMaker products search engine, was successfully met. The techniques used in the project, *tf-idf*, Weighted Inverse Document Frequency and the Weighted Term Frequency, provided significant improvement in the effectiveness of the MatchMaker software by ranking products in a better way based on the relevancy score calculated by the technique.

9. References

[1] Christopher D. Manning, Prabhakar Raghavan, Hinrich Schutze, *An Introduction to Information Retrieval*, Cambridge University Press, pp. 1, 2009.

[2] Christopher D. Manning, Prabhakar Raghavan, Hinrich Schutze, *An Introduction to Information Retrieval*, Cambridge University Press, pp. 155, 2009.

[3] Tokunaga, Takenobu and Iwayama, Makato, *Text categorization based on weighted inverse document frequency*, Special Interest Groups and Information Process Society of Japan, SIG-IPSJ, 1994.

[4] J. Aslam and M. Montague. *Models for metasearch*, ACM SIGIR 2001.

[5] J. H. Lee., *Analyses of multiple evidence combination*, ACM SIGIR '97, pages 267–275, Philadelphia, Pennsylvania, USA, July 1997. ACM Press, New York.

[6] E. A. Fox and J. A. Shaw, *Combination of multiple searches*, TREC 2 [1], pages 243–249.

[7] C. Clarke, J. Kamps, and M. Lalmas, *INEX 2006 retrieval task and result submission specification*, in N. Fuhr, M. Lalmas, and A. Trotman, editors, INEX 2006 Workshop Pre-Proceedings, pages 381–388, 2006.

[8] H. Turtle and W.B. Croft, *Evaluation of an interference network-based retrieval model*, ACM Transactions on Information Systems, Vol. 9, No. 3, pp. 187-222, 1991.

[9] T. Saracevic and P. Kantor, *A study of information seeking and retrieving. III. Searchers, searches, overlap*, Journal of the American Society for Information Science, Vol. 39, No.3, pp. 197-216, 1988.

[10] J. H. Lee, *Analyses of multiple evidence combination*. In Proceedings of the 20th Annual International ACM SIGIR Conference, pages 267{275, Philadelphia, Penn-sylvania, USA, July 1997.

[11] Harter, S.P., *A probabilistic approach to automatic keyword indexing. Parts 1 and 2*, Journal of the American Society for Information Science, 26, 197-206 and 280-289, 1975.

[12] Robertson, S.E.,van Rijsbergen, C.J. and Porter, M.F., *Probabilistic models of indexing and searching*, *Information retrieval research* (Ed. W.R. Oddy et al.). London: Butterworths, 35-65, 1981.

[13] Voorhees, E. (1994), *Query expansion using lexical-semantic relations*, proceedings of ACM SIGIR International Conference on Research and Development in Information Retrieval, pp. 61 – 69.

[14] Sparck Jones, K. (1971), *Automatic Keyword Classification for Information Retrieval*, Butterworth, London.

[15] Sttar, R., and Fraenkel, A. S. (1977), *Local Feedback in Full-Text Retrieval Systems*, Journal of the Association for Computing Machinery, 24(3), 397 – 417.

[16] Croft, W. B., and Harper, D. J. (1979), *Using probabilistic models of document retrieval without relevance information*, Journal of Documentation, 35, 285 – 295.

[17] Jinxi Xu and Bruce Croft, W., (1996), *Query Expansion Using Local and Global Document Analysis*, Proceedings of the 19th Annual International ACM-SIGIR Conference on Research and Development in Information retrieval New York, Association of Computing Machinery, 4-11.

[18] Callan, J., Croft, W. B., and Broglio, J. (11995), *TREC and TIPSTER experiments with INQUERY*, Information Processing and Management, pp. 327-343.
[19] Jing, Y., and Croft, W. B. (1994), *An association thesaurus for information retrieval*, Proceedings of RIAO 94, pp. 146-160.
[20] Juan Ramos, *Using TF-IDF to Determine Word Relevance in Document Queries*, First International Conference on Machine Learning, New Brunswick:NJ, USA, 2003.
[21] Jaap Kamps, Mounia Lalmas and Jovan Pehcevski, *Evaluating Relevant in Context: Document Retrieval with a Twist*, SIGIR'07, July 23–27, 2007, Amsterdam, The Netherlands.
[22] Mark Montague and Javed A. Aslam, *Relevance Score Normalization for Metasearch*, Proceedings of the 10th international conference on Information and knowledge management, 2001.
[23] K. Sparck-Jones, S. Walker, and S. E. Robertson, *A probabilistic model of information retrieval: Development and comparative experiments*, Information Processing and Management, pp. 779 - 840, 2000.
[24] Reference manual of Exorbyte MatchMaker-4.2, *Levenshtein Algorithm*, chapter – 2.2.1.
[25] Christopher D. Manning, Prabhakar Raghavan, Hinrich Schutze, *An Introduction to Information Retrieval*, Cambridge University Press, pp. 153-154, 2009.
[26] Harry Wu and Gerard Salton, *A comparison of search term weighting: Term relevance Vs Inverse Document Frequency*, TR 81 – 457, April 1981.
[27] MatchMakerInterface API reference manual of *Exorbyte MatchMaker 4.2 software.*
[28] Technical Specification, Project ECS Product.