INFORMATION RETRIEVAL USING METASEARCH

A dissertation submitted to the Jawaharlal Nehru University in partial fulfillment of the requirements for the award of the degree of

MASTER OF TECHNOLOGY

IN

COMPUTER SCIENCE AND TECHNOLOGY

BY

HEMANT ARYA

SCHOOL OF COMPUTER AND SYSTEMS SCIENCES
JAWAHARLAL NEHRU UNIVERSITY
NEW DELHI – 110067

JULY 2009

Dedicated to

My loving Parents

SCHOOL OF COMPUTER AND SYSTEMS SCIENCES
JAWAHARLAL NEHRU UNIVERSITY
NEW DELHI – 110067

DECLARATION

This is to certify that the dissertation entitled "**Information Retrieval using Metasearch**" is being submitted to the School of Computer and Systems Sciences, Jawaharlal Nehru University, New Delhi, in partial fulfillment of the requirements for the award of the degree of **Master of Technology** in **Computer Science & Technology**, is a record of bonafide work carried out by me under the supervision of **Dr. T.V. Vijay Kumar**.

The matter embodied in the dissertation has not been submitted in part or full to any University or Institution for the award of any degree or diploma.

[illegible] 29.07.2009

Hemant Arya
(Student)

SCHOOL OF COMPUTER AND SYSTEMS SCIENCES
JAWAHARLAL NEHRU UNIVERSITY
NEW DELHI – 110067

CERTIFICATE

This is to certify that this dissertation entitled "**Information Retrieval using Metasearch**" submitted by **Mr. Hemant Arya**, to the School of Computer and Systems Sciences, Jawaharlal Nehru University, New Delhi, for the award of degree of **Master of Technology** in **Computer Science & Technology**, is a research work carried out by him under the supervision of **Dr. T. V. Vijay Kumar**.

T. V. Vijay Kumar
Dr. T. V. Vijay Kumar 29-07-12
(Supervisor)

[illegible]
Prof. S. Minz
(Dean)

Acknowledgement

As a student, we always dream doing something concrete and useful which can be applicable. I fulfill my quest of creative thinking while working on this dissertation, which involved expert guidance.

I express my sincere gratitude for **Dr. T.V. Vijay Kumar** for introducing me to the fascinating area of Metasearch. His constant support and encouragement have been available throughout the completion of the work. It has indeed been a privilege to carry out this work under his supervision.

I would also like to thank **Prof. S. Minz**, **Prof. S. Balasundaram, Dr. D. P. Vidyarthi** and **Mr. Zahid Raza** for their vast experience and array of qualification make me proud with honor of being the student of this illustrious institution.

I express my heartily thanks to **Vikram Singh** and **Ajay Kumar Verma** for helping me to design the solutions and in programming. And their consistent support to update my modus operandi. I owe my indebtedness to **Mohammad Haider** and **Santosh Kumar** for their wonderful suggestion and judicious guidance. I also want to extend my thanks to my friend **Manoj** for their encouragement to peruse the M.Tech.Course. I would like to mention the names of Callan and Garcia-Molina for their wonderful papers.

I would be failing in my duty if I did not acknowledge **Anil Kumar Giri**, **Rajesh Kumar** and **Kumar Dilip** for their constant inspiring and support provided at all stages of this work.

I owe my deepest gratitude to my family members for their love and care given to me during the course of my work.

[illegible] 29.07.2009

Hemant Arya

Table of Contents

Chapter 1

IR Using Metasearch

Information Retrieval is the process of finding out of information (usually unstructured data) by giving the unstructured keyword. Textual data is usually unstructured. User gives their query in the form of a keyword. This is sent to the underlying IR system [SKS06]. The relevant documents are retrieved based on the relevance measure between the user query keyword and the underlying documents.

In IR systems, textual data is unstructured. Information retrieval generally refers to the querying of unstructured textual data. Information retrieval has played a critical role in making a web document a productive and useful for researches. Keyword based information retrieval can be used not only for retrieving textual data, but also retrieved video and audio data[SKS06].

In IR system each query and document term have associated weight. The weight specifies the magnitude of role to select the document. Document term weight [SKS06] can be computed as $w_j = tf_j * idf_j$ that is product of term frequency and inverse document frequency

Term frequency [SKS06] of jth term $tf_j = \log(1+\frac{n(d,t)}{n(d)})$ where n(d) denotes the number of term in the document d and, n (d, t) denotes the number of occurrence of term jth in the document d.

Inverse Document Frequency [SKS06] can be calculated as:

$$idf_j = \log\frac{N}{n_i} \quad ,$$

Where N is total number of document and n is the number of document that contain the jth term

So the weight of jth term in ith document $w_{ij} = tf_{ij} \times \log\frac{N}{n_i}$

Query term weight is the number of occurrence of a particular term over all query term [RB04]. For query term weight, Salton and Buckley [RB04] suggest

$$w_{i,q} = (0.5 + \frac{0.5\,freq_{i,q}}{\max freq_{i,q}}) \times \log\frac{N}{n_i}$$

Where $freq_{i,q}$ the frequency of ith term in query q and $\max freq_{i,q}$ is the maximum frequency in query q

Several retrieval methods exist namely Boolean model, Vector Space model, Probabilistic model, Language model, Inference model, impact factor model, connectivity model, mutual citation model, page rank model, HITS, SALSA model, Associative interaction model and Bayesian model.

Boolean model [RB04] is based on the concept of set theory and Boolean algebra. User gives its query in the form of Boolean expression. Term weight in case of Boolean model is either {0, 1} means binary weight that's means term either present or absent. There is no concept of partial matching. The absence of partial matching is a major disadvantage of Boolean model.

In a vector space model [RB04] each query and document can be represented as a vector of query term weight and document term weight respectively. Similarity between the document and the query can be computed by a cosine similarity function. Each query term have some weight associated with it, which signifies the importance of a term. Weight is not usually binary as in the case of Boolean. So there can be partial matching. Let q= (q_1, q_2, q_3.....q_m) where q_1, q_2, q_3.....q_m are query term weight and d= (w_1, w_2, w_3...w_m) where w_1, w_2, w_3...w_m are document term weight and m is the number of query term. The similarity between the document and query can be calculated by [RB04]:

$$Sim(q,d) = \frac{\sum_{j=1}^{m} q_j \cdot w_j}{|q||d|}$$

Where value of sim(q,d) lies between 0 and 1. If sim(d,q)=0 ,then not matched else if sim(d,q)=1,then perfectly matched else if sim(d,q)=[0,1] ,then partial matched.

The probabilistic model is based on probability ranking principle [RB04], which states that optimal retrieval performance is achieved when documents are ranked according to their probabilities of being judged relevant to a query. For the probabilistic model, the index term weight variables are all binary .A query q is a subset of index terms. Let R be the set of relevant document. Let $\overline{R}$ be the complement of R (set of non relevant document). Let $P(\frac{R}{d_j})$ be the probability that the document d_j is relevant to the query q and $P(\frac{\overline{R}}{d_j})$ be the probability that d_j is non relevant to q [RB04]. The similarity sim(d_j,q) of the document d_j to the query q [RB04] is calculated as

$$Sim(d_j,q) = \frac{p(R/d_j)}{p(\overline{R}/d_j)}$$

Using Baye's rule [RB04],

$$Sim(d_j,q) = \frac{P(\vec{d_j}/R) * P(R)}{P(\vec{d_j}/\overline{R}) * P(\overline{R})}$$

$P(\frac{\vec{d_j}}{R})$ denotes the probability of randomly selecting the document d_j from relevant document R. $P(\frac{\vec{d_j}}{\overline{R}})$ denotes the probability of randomly selecting document d_j from the non relevant set $\overline{R}$. Since $p(R)$ and $P(\overline{R})$ are the same for all document in the collection [RB04], therefore

$$Sim(d_j,q) \sim \frac{P(\vec{d_j}/R)}{P(\vec{d_j}/\overline{R})}$$

There are some basic problems with an IR system such as all the document need not exist in a single database. This limits the search area. Further it becomes difficult for IR systems to handle documents dynamically. This scalability issues is addressed by Metasearch engines. A metasearch engine is a search engine over search engines. A metasearch engine has large number of databases, containing documents, attached to them. Metasearch engine is discussed next.

1.1 Metasearch Engine

Metasearch engine [MYL02] can be defined as the search engine over search engines. Various collection/databases can be attached with the unified system called the Meta search engine. User send query to the Meta search engine, which is responsible for identifying useful subset of databases that may contain relevant documents. The documents retrieved from these identified databases are combined to form a unified single ranked list ordered according to the relevance with most relevant document at the top. The metasearch engine architecture [MYL02] is shown in Figure 1.1.

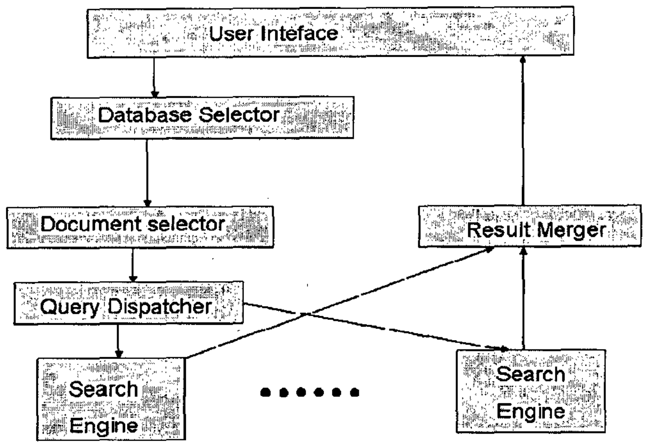

Figure 1.1: Two-level architecture of a Metasearch engine [MYL02]

Database selector: The main objective of the database selector is to identify the potentially useful database from all the databases [MYL02].

Document Selector: Document selector selects relevant document from the databases that are selected. Various similarity measure methods such as Boolean, vector space model, probabilistic model are exit for calculating the relevance document with respect to user query [MYL02].

Query Dispatcher: It is responsible for the passing the query with necessary changes to the original query and forward the modified query to the appropriate databases [MYL02].

Result merger: It merges the document retrieved from the selected databases and arranges them in single rank list, which is ordered by relevance with the most relevant document appearing at the top.

Metasearch engines increase the search coverage area, solves the scalability problem of searching the Web, facilitate the invocation and cooperation of multiple search engines and also improve the retrieval effectiveness.

1.1.1 Database Selection

Metasearch engine on receiving a user query forwards it to a subset of database which may contain relevant document. Appropriate selection of database is carried out by using good database selection algorithms. These algorithms differ on the database representative which is used to indicate the content of database. There are three basic data representative approaches:

Rough Representative Approach: In this approach content of each database is represent by using few keywords or paragraph [MYL02], which is mostly performed manually as in ALIWEB [KM94].

Statistical Representative Approach: In this approaches the content of each database is represented by detailed statistical information such as term frequency, document frequency [MYL02]. This approach allows more accurate estimation of database usefulness because of detailed statistical information. Main problem with this approach is scalability issue, like in D-WISE [YL96], CORI-Net [CLC95], gGLOSS[GGM95].

Learning Based Approach: In this approaches, the usefulness of the database for a new query is based on the past retrieval experiences with training query [MYL02]. There are three type of learning approach such as Static, Dynamic and Hybrid. In static learning approaches, the retrieval knowledge once learned can't be changed. This is not suitable to the changes in database and query patterns, like in MRDD Approach [VGL95b]. In Dynamic learning approach, problem due to static approaches can be overcome by using real user queries and updating the retrieval experience like in Savvy Search Approach [DH97]. Hybrid learning approaches overcomes the weakness of the other two learning approaches by obtaining knowledge from training queries. This knowledge be further updated using the real user queries like in PROFUSION [FG99].

Cost Based Approaches: In this approach, the usefulness of the database is based on various cost factors such as cost of accessing digital library, time, money and query evaluations cost [F97]. This approach allows only retrieval of document from databases that minimize the accessing cost. Like TRD-CS [ASB02] and ReDDE[SC03] etc.

Several database selection models exist like D-WISE (Distributed Web Index Search Engine), gGLOSS (generalized Glossary Of Servers' Server Approaches), CORI-Net (Collection Retrieval Inference Network Approaches), MRDD (Modeling Relevant Document Distribution Approaches), SavvySearch, PROFUSION, TRD-CS (Top Ranked Document for Collection Selection Approaches), DTF (Decision theoretic framework Approaches), LWP (Light Weight Probes Approaches), ReDDE (Resource Selection that normalize database size)

D-WISE (Distributed Web Index Search Engine) [MYL02]

This approach uses statistical information of each database i.e. df_{ij}, which is the document frequency of jth term in ith database i.e. the number of document in ith database that cointain jth term, and n_i is the total number of document in the ith database[MYL02]. The Cue Validity (CV), which specify the proportion of document that contains relevant jth term in ith database over all documents containing jth term in remaining database, is calculated as [MYL02]:

$$CV_{ij} = \frac{\frac{df_{ij}}{n_i}}{\frac{df_{ij}}{n_i} + \frac{\sum_{k \neq i}^{N} df_{kj}}{\sum_{k \neq i}^{N} n_k}}$$

Where N is the total number of component database in metasearch engine. Now calculate the distinguishing power of jth term called CVV_j, which is Variance of CVij of each term t_j over all the databases.

$$CVV_j = \sum_{i=1}^{N} \frac{(CV_{ij} - ACV_j)^2}{N}$$

Where ACV_j is the average of Cue Validity of the jth term in all databases [MLY02]. CVV_j specifies the importance of a term t_j over all other query terms. It helps in retrieving relevant document. If $CVV_r > CVV_s$, then $t_r > t_s$ where term t_r has more weightage than t_s. Now the ranking [MLY02] can calculated as.

$$r_i = \sum_{j=1}^{M} CVV_j \cdot df_{ij}$$

where M is number of query terms and r_i is the ranking score of ith database

GLOSS (Glossary Of Servers' Server approaches)[GGM95]

GLOSS can be categorised as vGLOSS, bGLOSS and hGLOSS.

vGLOSS (vector GLOSS[GGM95])

In case of vGLOSS the similarity between the document and the query can be calculated as

$$Sim(q,d) = \sum_{j=1}^{m} q_j \cdot w_j$$

Where q_j is weight of jth term in query and wj is weight of jth term in document.

The Goodness of a database is computed as

$$Goodness = (l, q, db) = \sum_{d \in db \wedge Sim(q,d) \geq l} Sim(q,d)$$

The number of document in each database, with similarity to the query greater than threshold i.e. l are used to find Goodness [GGM95].

bGLOSS (boolean GLOSS[GGM95])

Let $|db_i|$ be the total number of document in database db_i and f_{ij} is the number of document in db_i that contain t_j. For a given database db_i and t_1, t_2, ...t_n keyword pair and any document d $\in$ db_i. So the estimation number of document in db_i that will satisfy all query terms is given as

$$Estimate\left(t_1 \wedge t_2 \wedge \ldots \wedge t_n, db_i\right) = \frac{\prod_{j=1}^{n} f_{ij}}{|db_i|^{n-1}}$$

hGLOSS (hybrid GLOSS[GGM95])

Let us consider a number of GLOSS Server G_1, G_2..........................G_S. All servers are of same type i.e. either they are bGLOSS or vGLOSS.

hGLOSS summarize the content of their GLOSS servers. hGLOSS can treat the information about a database as tradition GLOSS server treat the information about a document in the underlying databases. The document for hGLOSS will be the database summaries at the GLOSS servers. GLOSS server stores h_{rj} i.e. the number of database in vGLOSS, Gr that contain term t_j, d_{rj} is the total number of document that contain term t_j in each database in vGLOSS G_r.

CORI-Net (Collection Retrieval Inference Network Approaches) [CLC95]

CORI-Net [CLC95] is inference network based probabilistic approach. It uses the document frequency and database frequency to select databases. If a term appears in k document in the database, the term is repeated k times in the super document. As a result, the document frequency of a term in the database becomes the term frequency in the super document [CLC95]. Let D denote the database of all super documents. Note that the database frequencies of term in the database become the term frequency in D. In each super document is represented as a vector of weight tf*idf (term frequency weight times inverse document frequency weight)[CLC95]. The cosine similarity function can be used between stored super document and given user query. This similarity function can be used to rank all component database (or super document).

Let N be the number of database in the metasearch engine, df_{ij} is document frequency of jth term in ith component database and db_j is database frequency of jth term[SJ04].

First the belief that D_i contain useful document due to the jth query term is computed by [SJ04]:

$$P\left(\frac{t_j}{D_i}\right) = C_1 + (1 - C_1) \cdot T_{ij} \cdot I_j$$

Where

$$T_{ij} = C_2 + (1 - C_2)\frac{df_{ij}}{df_{ij} + K},$$

$$I_j = \frac{\log\left(\frac{N + 0.5}{dbf_j}\right)}{\log(N + 1.0)}$$

$$K = C_3 \cdot (1 - C_4) + C_4 \cdot \frac{dw_i}{adw_i}$$

where dw_i is number of words in D_i and adw_i is average number of words in a database. The value of C_1, C_2, C_3, C_4 estimated empirically by performing experiment on the actual text collection [SJ04]. $P\left(\frac{t_j}{D_i}\right)$ is essentially the tf*idf weight of term t_j in the super document corresponding to database D_i and can be estimated, for query term weight t_j in q. Now ranking score of each database with respect to given query can be calculated by as [SJ04]:

$$\sigma_i = P\left(\frac{q}{D_i}\right) = \sum_{j=1}^{K} P\left(\frac{q}{t_i}\right) \cdot P\left(\frac{t_j}{D_i}\right)$$

where k is total number of query terms.

Learning based Approach:

Learning based approaches can be classified as Static based (e.g. MRDD), Dynamic based (e.g. SavvySearch) and Hybrid based (e.g. PROFUSION)

MRDD (Modeling Relevant Document Distribution Approaches) [VGL95b]

In this approach, the knowledge once learned can't be changed. In learning phase, some set of queries also called training query is used. These set of queries is forwarded to each component database. From the retrieved documents, relevant documents are identified. Distribution vector of all relevant documents is obtained [MYL02]. The user query is matched with the k most similar training query. For each database D, the average relevant document over the k vector for each k queries and D is obtained [MYL02]. These average distribution vectors are then used to rank the database. The documents from higher rank database are retrieved. In MRDD approaches, the representative of a component database is the set of distribution vectors for all training queries. Main drawback of MRDD is that the entire learning part is performed manually and it is very difficult to identify appropriate training queries. The approach does not perform well when content of database keeps changing dynamically.

SavvySearch Approach [DH95]

In SavvySearch Approach [DH95], weight vector (w_1, w_2... w_m) is maintained where each w_i's corresponds to i^{th} term in the database. The weight wi is adjusted when a query containing term t_i is used to retrieve documents from the component database D. If no document is retrieved, the weight is reduced by 1/k. If at least one document is read/clicked by the user, the weight is increase by 1/k. Hence large positive value w_i indicates that the database D responds well to term t_i and vice versa. Given a query q containing term t_1, t_2,....,t_k, the ranking score for each database D is calculated [MYL02] as:

$$r(q,D)=\frac{\sum_{i=1}^{k} w_{t_i}\cdot\log\left(N/f_i\right)}{\sqrt{\sum_{i=1}^{k}|w_i|}}-(p_h-p_r)$$

where P_h is the penalty for a search engine, which depends upon the retrieve results, P_r is the penalty for a search engine, which depends upon the response time, $\log(N/f_i)$ is inverse database frequency weight of term t_i, N is number of databases and f_i is number of database having a positive weight value for term t_i. The Savvy Search have limitations like terms in the previous queries are only considered, does not perform well for new query term and also for query terms that are rarely used.

PROFUSION Approach [FG99]

The PROFUSION approach [FG99] use both static and dynamic approaches and is also called hybrid Approach[FG99]. Problem due to only static features and only dynamic features can be overcomes by using both the approaches together. The hybrid learning approach utilized in 13 categories problems such as Science and engineering, Computer science, Travel, Medical and Biotechnology, Business and Finance, Social and Religion, Society, law and government, Animals and environment, History, Recreation and entertainment, Arts, Music and Food. Some set of term is associated with each of the 13 categories to specify the type of document each category contains. For each of the 13 categories, a set of training queries are identified and for a given database D and a given categories C each associated training query is submitted to database D. Score reflecting the performance of D with respect to the query and the categories C is computed [MYL02] as

$$C \cdot \frac{\sum_{i=1}^{10} N_i}{10} \times \frac{R}{10}$$

where C is the constant and R is the number of relevant document in 10 top documents. Value of N_i is calculated as

$$N_i = 1/i, \text{ if } i^{th} \text{ ranked document is relevant}$$
$$= 0, \text{ otherwise}$$

Score of all training queries associated with the category C is averaged for database D and this average is the confidence factors of the database with respect to category. At the end of the training, there is a confidence factor for each database with respect to each of the above 13 categories [MYL02].

In the testing phase, the user query, containing a set of terms, is mapped to set of categories based on query terms. The database is then ranked based on the sum of the confidence factors of each database with respect to mapped categories. This sum of confidence factor with respect to query is called the ranking score of the database for q. The score of the database with respect to a given query is also updated dynamically based on retrieved result. If the user considers document d to be relevant and d is not the topmost document then the score of the database having document d is increased and score of all other databases whose documents are ranked higher then d is decreased[MYL02].

Cost Based Approach

Several cost based approaches exist in literature like TRD-CS, DTF, ReDDE, LWP etc.

TRD-CS (Top Ranked Document for Collection Selection Approach) [ASB02]

In TRD-CS [ASB02], the user query is broadcasted to all available databases. Each of these databases retrieves n ranked documents. The retrieved documents are sorted based on their scores. The databases containing the n-first documents are selected. The document score can be calculated [ASB02] as

$$Score(d,q) = (C_1 \cdot nb_d) + (C_2 \cdot dis\_ind(d,q)) + \frac{nb\_occ}{C_3}$$

where nb_d is number of search keyword included in the document d, nb_occ is number of occurrence of query terms in d and dis_ind(d,q) is the distance between two query terms[ASB02] and is computed as

$$dis\_ind(d,q) = \sum_i dis(K,l),$$

where K& l are search keyword position within the document d delimited by ith block and dis (k, l) is

$$dis(K,l) = \begin{cases} \frac{1}{|(K,l)_i|} & \text{if} \quad |(K,l)_i| > 1 \\ 1 & \quad |(K,l)_i| \le 1 \end{cases}$$

Document should be retrieved from the database having higher score [ASB02].

DTF (Decision theoretic framework Approach) [F97]

This is also called Cost based resource selection approach [F97] in which there is a retrieval cost $C_i(S_i, q)$ associated with each digital library DL_i where S_i documents are retrieve for query q. There are following factor that include cost are Cost, Time and Quality. Cost may include money[F97]. Time may include computation time at digital library site, computation time for delivering the resultant document over the network, response time for several queries [F97]. Quality may include cost to retrieve high quality document [F97]. For a user query, and total n document to be retrieved, the task is to find an optimum solution. Let

$$\vec{S} = (S_1, S_2, \ldots, S_m)^T,$$

where m is the total number of digital libraries

$$|\vec{S}| = \sum_{i=1}^{m} S_i = n$$

The overall cost is minimized

$$M(n,q) = \min_{|\vec{S}|=n} \sum_{i=1}^{m} C_i(S_i,q)$$

If $r_i(S_i,q)$ denotes the number of relevant document in the result set when S_i document are retrieved from digital library DLi for query q, we obtain the cost function [F97].

$$relC_i(S_i,q) := [\, r_i(S_i,q).C^{+}] + [S_i - r_i(S_i,q)].C^{-}$$

Where C^+ is the cost of retrieving relevant document and C^- cost of retrieving irrelevant document

ReDDE (Resource Selection that normalize database size) [SC03]

ReDDE[SC03] estimate the distribution of relevant document across the set of available database. For making its estimation ReDDE considers both database size and content similarity. ReDDE is the resource selection algorithm that normalizes the database size. For a query q posed on database C_i, the number of document relevant can be estimated as [SC03]:

$$\mathrm{Rel}\_Q(i) = \sum_{d_j \in C_{i\_Samp}} P(rel \mid d_j) \cdot \frac{1}{N_{C_{i\_Samp}}} \cdot \hat{N}_{C_i}$$

Where

$N_{c_i\_samp}$ is the number of sampled document from the ith database, N_{c_i} is the estimated size of the ith database and P (rel/dj) is the probability of relevance for a specific document where P (rel/dj) is given as [19]

$$p(rel/d_j) = \begin{cases} C_Q & \text{if} \quad Rank\_central\,(d_j) < ratio \cdot \sum_i N_{c_i} \\ 0 & otherwise \end{cases}$$

Rank for the centralized sample database is the union of all sampled document from different database [SC03].

$$rank\_central\,(d_j) = \sum_{d_k} \frac{N_{c(d_k)}}{N_{c(d_k)\_samp}}$$

$$rank\_sample\,(d_k) < rank\_sample\,(d_j)$$

The estimation is normalized to remove the query dependent constant, which provides the distribution of relevant document among the database.

$$Dist\_\mathrm{Re}l\_Q(i) = \frac{\mathrm{Re}l\_Q(i)}{\sum_j \mathrm{Re}l\_Q(j)}$$

LWP (Light Weight Probes Approach) [HT99]

The LWP approach, for large term queries that require more space and time to execute at the server side, forwards probe, instead of the entire query term, to the server. The probes are useful as they have low cost and are smaller in size thereby use low bandwidth and has low latency [HT99].

Let D_i be the total number of document on the server, f_{prox} be the number of documents containing the specified number of terms within a specified proximity with each other, $f_{cooccur}$ is the number of documents in which a specified number of the term co-occurs and f_i is the number of documents containing each individual term t_i then the score can be calculated as [HT99]

$$S_i: =C_1 f_1' + C_2 f_2' + C_3 f'_{cooccur} + C_4 f'_{prox}$$

where

$$C_i = \begin{cases} \sum_{j=1}^{|S|} f_{ij} \Big/ f_k & \text{if} \sum_{j=1}^{|S|} f_{ij} \le f_k \\ f_i \Big/ \sum_{j=1}^{|S|} f_{ij} & \text{otherwise} \end{cases}$$

where f_{ij} is the raw frequency of probe term i on server j. Term are weighted according to closeness of their total frequency to target total frequency f_t.

1.1.2 Result merging

For a given query, the selected search engines return the relevant documents. The merging of these retrieved relevant documents in a single ranked list is referred to as result merging. The result merging problem is difficult because document scores returned by the different databases cannot be compared directly. The goal of a result merging algorithms to produce a single rank list that is comparable to what would have been produced if the document from all available databases were combined into a single database.

Several result merging methods exist in literature like CORI merging [NF03], Semi supervised Learning Approach to merge results (SSL) [SCO03a], Hybrid Result merging [PSS07], LMS [RAS03], SORTED RANK SCORE [LXG03] etc.

CORI result merging algorithm[NF03]

The CORI algorithm[NF03] is based on a Bayesian Inference Network Model of information retrieval in which each resource is ranked by the belief $P(Q/C_i)$ that Query Q is satisfied given that resource C_i is observed(searched)

In CORI, the score of databases is normalized on the scale of [0, 1], where C_{min} and C_{max} are the maximum and minimum score of the database [NF03].

$$C' = \frac{C_i - C_{\min}}{C_{\max} - C_{\min}} \qquad \text{(Normalized document of i}^{\text{th}}\text{ database)}$$

This is followed by normalizing the score of document on the scale of [0,1],where D_{min} and D_{max} are the maximum and minimum score of the document[NF03].

$$D' = \frac{D - D_{\min}}{D_{\max} - D_{\min}} \qquad \text{(Normalized database)}$$

The score of the database are updated as follows[NF03]

$$D'' := \frac{1.0 * D' + 0.4 * C' * D'}{1.4} \quad \text{(Global normalized score of each document)}$$

Finally, documents are ranked in decrease order of score D".

Semi –Supervised Learning (SSL) [SCO03]

Semi-supervised learning (SSL) [SCO03] is a result merging algorithm that is based on linear regression. It takes advantage of a centralized sample index, which comprises of the entire sampled document from remote databases. It relies heavily on the discovery of a significant number of matching document relevance score returned from remote collections in order to function effectively.

Hybrid Result Merging [PSS07]

It combines both regression and downloadable methologies [PSS07]. It selectively downloads a limited number of documents that are used as training data for the regression method. The ranking of a document in the remote collection is based on a particular query. The centralized sample index is utilized as a reference statistics database that provides estimates of global collection statistics (global idf) [PSS07]. The decision of downloading document is incorporated into the algorithm thereby minimizing the produced overheads and maximizing the gained accuracy.

Sorted Rank (SR) Score Merging [LXG03]

In SR score merging [LXG03], all document for a particular databases are considered in a sorted form. The merging of the document is based on the document of the selected databases. Any documents that have higher score even from the less scored database must be ranked in final rank list. Database rank is only used to select subset of databases and documents from these selected databases are used to arrive at final rank merge list. Score of the document of selected database are updated by the product of original score of document and score of corresponding database.

LMS Merging [RAS03]

This merging approach uses result length to calculate merging score. Since this approach uses the document score and result length, this approach is simple to implement. In this approach, first the result length of each database is computed i.e. the number of relevant documents in the database are computed. The result length is then used to calculate database score

$$S_i = \log\left(1 + \frac{l_i \cdot K}{\sum_{j=1}^{|C|} l_j}\right)$$

Based on this database score, the weight for the ith databases is computed as [RAS03]:

$$w_i = 1 + \left[\frac{S_i - \bar{S}}{\bar{S}}\right]$$

The basic idea behind the LMS merging approach [RAS03] is to increase the weight of those databases that have score less than the average score and decrease the score of those databases that have score higher than the average score.

1.2 Aim

This dissertation aims to carry out the following:

- Compare performance of database selection methods CORI, vGLOSS and CVV
- Compare performance of Result merging methods LMS, CORI and SR Merge
- Compare performance by combining the various database selection and result merging method.

1.3 Organization

The brief introduction of IR System and Metasearch Engine is given in chapter 1. The performance of database selection method CORI, vGLOSS and CVV are compared in chapter 2. This is followed by comparing performance of result merging method LMS, CORI and SR Merge in chapter 3. Chapter 4 compares performance by combining the various database selection and result merging method. Chapter 5 is the conclusion.

Chapter 2

Database Selection

The main objective of the databases selection is to select only those databases which contain useful document for a given query. User query is sent only to the selected databases. In this chapter, the database selection algorithms CORI, vGLOSS and CVV model are compared on performance parameters. The algorithms of each of these methods along with an example are discussed. It is followed by experimental based comparisons of these methods.

2.1 CORI Collection Selection Model [MYL02]

As discussed earlier CORI selection model is based on Bayesian CORI [MYL02] algorithm takes database information, like number of database m, number of documents n in each database and number of terms l in each documents, and query information, number of terms l in the query, as input. In step 1, document frequency of each term in each database is calculated. The database frequency of each term is evaluated in step 2. Step 3 calculates the inverse database frequency of each term. This is followed by calculating the weight of each term in database in step 4. In step 5, score of a database with respect to query is calculated. These databases are sorted with respect to their scores in descending order and finally the top ranked databases are produced as output. Inference network [[SJ04]. The algorithm based on CORI [SJ04] is shown in Figure 2.1.

```
Input: D [m, n, l],where m is number of database, n is number of  document, l is number of terms,
q[l]
Output: Ranked list of topmost databases
Method:
        Step1: For each kth term of the query
                   For (each ith database)
                       Calculate $F_{c,t}[i] [k]$; Number of documents containing kth term in ith database
        Step2: For each kth term of the query
                   Calculate $f_t[k]$; Number of databases contains the term.
        Step3: For each kth term of the query
                   For (each ith database)
                   Calculate $I_{c,t}[i][k]=\frac{\log((N+0.5)/f_t)}{\log(N+1.0)}$ ); which is inverse document frequency
        Step 4: For each kth term of the query
                   For (each ith database)
                   Calculate $T_{c,t}[i][k]= 0.4+0.6*(\frac{f_{c,t}}{f_{c,t}+50+150*k})$ ; where k is 0.00039 and
                   Tct is weight of the term in the collection
        Step5: CORI (q, c) = $\frac{\sum_{t\in q\&c}(0.4+0.6*T_{c,t}[i][k]*I_{c,t}[i][k])}{|q|}$
        Step6: Sort the database in decreasing score of each database
```

Figure2.1: Algorithm based on CORI [SJ04]

An example illustrating the above algorithm is given next.

2.1.1 An Example

Table 2.1 gives database information, which contains information about five databases. Each of the five databases contains five documents, and there are four types of keyword in each document. The cell in the table gives information about the frequency of each keyword in a document of each database.

Database	Doc_0	Doc_1	Doc_2	Doc_3	Doc_4
DB_0	6 ,0,2,0	6,7,5,5	8,6,4,8	1,9,2,0	2,1,3,7
DB_1	9,1,0,5	4,9,2,1	9,6,1,1,	5,3,4,0	9,5,1,2
DB_2	9,8,6,4	2,3,6,4	9,1,7,0	5,2,8,6	7,2,4,9
DB_3	9,3,6,1	0,8,5,9	0,3,3,9	5,2,3,7	6,8,0,8
DB_4	6,0,2,2	7,5,6,8	6,9,8,9	1,7,5,1	5,0,0,1

Table 2.1: Database Information chart

The frequency and weight of terms in the user query are given in table 2.2.

	T_0	T_1	T_3	T_4
Frequency	7	5	3	7
weight	0.497325	0.551531	0.618072	0.497325

Table 2.2: Query information chart

Term weight of the keyword in query can be computed using the following formula [RB04]:

$$w_{i,q} = (0.5 + \frac{0.5\, freq_{i,q}}{\max\ freq_{i,q}}) \times \log \frac{N}{n_i}$$

In the example, value of maxfreq and N are 7 and 22 respectively. The database frequency, i.e. the number of databases containing each term is given in table 2.3.

Term	T_0	T_1	T_2	T_3
f_t	5	5	5	5

Table 2.3: Database Frequency Chart

The document frequency of each term in each database is given in table 2.4

$f_{c,t}$	DB_0	DB_1	DB_2	DB_3	DB_4
T_0	5	5	5	3	5
T_1	4	5	5	5	3
T_2	5	4	5	4	4
T_3	3	4	4	5	5

Table 2.4: Document Frequency Chart

Next inverse collection frequency is computed. That can be computed using the formula [SJ04]

$$I_{c,t}[i][k] = \frac{\log((m + 0.5)/f_t[k])}{\log(m + 1.0)}$$

where m is 5(number of databases). The $I_{c,t}$ values are given in table 2.5

$I_{c,t}$	DB_0	DB_1	DB_2	Db_3	DB_4
T_0	0.053194	0.053194	0.053194	0.053194	0.053194
T_1	0.053194	0.053194	0.053194	0.053194	0.053194
T_2	0.053194	0.053194	0.053194	0.053194	0.053194
T_3	0.053194	0.053194	0.053194	0.053194	0.053194

Table 2.5: $I_{c,t}$ Values

The weight of the term in collection $T_{c,t}$ is calculated next. The weight $T_{c,t}$ is computed as [SJ04]

$$T_{c,t}[i][k] = 0.4 + 0.6 * \left(\frac{f_{c,t}[i][k]}{f_{c,t}[i][k] + 50 + 150 * k} \right)$$

where $f_{c,t}$ have there usual meaning. The $T_{c,t}$ values are given in table 2.6

$T_{c,t}$	DB_0	DB_1	DB_2	DB_3	DB_4
T_0	0.454487	0.454487	0.454487	0.433925	0.454487
T_1	0.444396	0.454487	0.454487	0.454487	0.433925
T_2	0.454487	0.444396	0.454487	0.444396	0.444396
T_3	0.433925	0.444396	0.444396	0.454487	0.454487

Table 2.6: $T_{c,t}$ chart

Next, the similarity of the user query with respect to the given database is computed using [SJ04]

$$CORI(q,c) = \frac{\sum_{t \in q \& c} (0.4 + 0.6 * T_{c,t}[i][k] * I_{c,t}[i][k])}{|q|}$$

where q is the number of query terms The database similarity is given in table 2.7

Databases	Database Similarity
DB_2	0.414425
DB_1	0.414344
DB_0	0.414261
DB_3	0.414261
DB_4	0.414261

Table 2.7: Database Similarity by CORI Collection Selection model

The databases in the table are in descending order of their scores. The database at the top are selected for answering the user query.

2.2 vGLOSS Collection Selection Model [GGM95]

This model is based on the vector space model. The algorithm based on vGLOSS[GGM95] is shown in Figure 2.2

```
Input: D [m, n, l], Q[l] where m is number of database, n is number of document and l is number of terms
Output: Ranked list of topmost databases
Method:
        Step 1: Repeat Step (1) to (6)
        Step 2: Calculate term frequency (tf) of kth term in jth document
                $tf_i[j][k] = \log(1 + \frac{n(d,k)}{n(d)})$
        Step 3: Calculate Inverse document frequency of kth term in jth document
                $idf_i[j][k] = \log(\frac{N}{n_k})$ ; where n is the number of documents containing kth term
        Step 4: Calculate weight of kth term in jth document
                $w_i[j][k] = tf_i[j][k] * idf_i[j][k]$
        Step 5: for each databases
                Calculate document frequency of document
                $Sim_i(q, d_j) = \frac{\sum_{k=1}^{l} q_k * w_{jk}}{|q||d_j|}$ ; where $q_k$ is the weight of kth term in query and $w_{jk}$ is the weight
                of kth term in jth document
        Step 6: Now, calculate goodness of each ith database
                For (each jth document)
                $Goodness_i = (l, q, db_i) = \sum_{d_j \in db_i \wedge Sim_i(q,d_j) \ge T} Sim_i(q, d_j)$
                Add the similarity of all documents whose score is greater than T (Threshold)
        Step 7: Sort the databases based on the goodness
                Generate list of top-S database as output
```

Figure 2.2: Algorithm based on vGLOSS[GGM95]

vGLOSS takes database information and user query q as input. The term frequency of each term in each document of each database is computed in step 1. In step 2, the document frequency of each term is computed. The weight of the term, based on the term frequency and inverse document frequency, is computed in step4. Step 5 evaluates the similarity between the document and query. In step 6, goodness of each database is computed using the documents having similarity greater than the threshold. The databases are sorted in descending order of their goodness scores in step 6.

2.2.1 An Example

The cosine similarity of all documents with respect to query is given in table 2.8.

	Doc$_0$	**Doc$_1$**	**Doc$_2$**	**Doc$_3$**	**Doc$_4$**
DB$_0$	0.0000000	0.652078	0.596110	0.507540	0.494404
DB$_1$	0.457657	0.715162	0.725796	0.568774	0.667025
DB$_2$	0.457657	0.457657	0.000000	0.457657	0.457657
DB$_3$	0.605235	0.568774	0.568774	0.592228	0.457657
DB$_4$	0.568774	0.711777	0.680501	0.659872	0.000000

Table 2.8: Display similarities of all document w.r.t query

The goodness of database is calculated using the following formula [GGM95]:

$$\text{Goodness} = (l, q, db) = \sum_{d \in db \wedge Sim(q,d) \geq l} Sim(q, d)$$

The Goodness of the databases with respect to query are given in table 2.9

Databases	**Databae Goodness**
DB$_1$	3.134414
DB$_3$	2.792668
DB$_4$	2.620924
DB$_0$	2.250131
DB$_2$	1.830628

Table 2.9: Database Goodness w.r.t query

The databases having high Goodness value are selected for answering user query.

2.3 CVV Collection Selection Model [YL96]

This model is based on the distinguishing power of the term in the databases. The representation of the databases consists of the document frequency of each term and the cardinality of each database. The algorithm based on CVV [YL96] is shown in Figure 2.3

```
Input: D [m, n, l] where m is number of database, n is the number of document, l is the number terms, q[l]
Output: Ranked list of topmost databases
Method:
        Step 1: For each term j^th of a query
                For (each database i^th database)
                $df[i][k] = \log(1 + \frac{n_k}{N})$ ; where df[i][k] is the document frequency of k^th term in i^th database ,N
                is the total number of documents and $n_k$ is the number of document containing k^th term
        Step 2: Calculate Cardinality of databases
                For (each i^th database)
                Card[i] = number of document in i^th database
        Step 3: Calculate Cue validity (CV)
                For (each k^th term)
                    For (each i^th databases)
                $CV_{ik} = \frac{\frac{df_{ik}}{n_i}}{\frac{df_{ik}}{n_i} + \frac{\sum_{p \neq i}^{N} df_{pk}}{\sum_{p \neq i}^{N} n_p}}$
        Step 4: Calculate Average Cue validity (CV)
                For (each i^th database)
                $ACV_k = \frac{\sum_{i=1}^{M} CV_{ik}}{M}$ ; where M is the number of database
        Step 5: Calculate Variance of Cue validity ($CV_{ik}$)
                For (each i^th database)
                $CVVk = \sum_{i=1}^{M} \frac{(CV_{ik} - ACV_k)^2}{M}$
        Step 6: Calculate rank i=1.....m ing can be d by
                For (each i^th databases)
                $r_i = \sum_{k=1}^{1} CVV_k * df_{ik}$
        Step 7: Generate top-S databases
```

Figure 2.3: Algorithm based on CVV [MYL96]

CVV algorithm takes the database information and query information as input. In step 1, the document frequency of each query term in each database is computed. This is followed calculating the cardinality of each database in step 2. Step3 computes the Cue validity of each term. In step 4, the average cue validity (ACV) is computed. The variance of cue validity i.e. CVV is calculated in step 5. Based on the document frequency and CVV of each term, the databases ranking is done in step 6. This sorted rank of database is produced as output in step 7.

2.3.1 An Example

Using CVV algorithm, first the document frequency of each term is computed and is given in table 2.10

Databases	T_0	T_1	T_2	T_3
DB_0	5	4	5	3
DB_1	5	5	4	4
DB_2	5	5	5	4
DB_3	3	5	4	5
DB_4	5	3	4	5

Table 2.10: Document frequency

The cardinality of the databases i.e. the number of documents contained by each database is computed. The cardinality of databases is shown in table 2.11.

Databases	Cardinality
DB_0	5
DB_1	5
DB_2	5
DB_3	5
DB_4	5

Table 2.11: Databases and their cardinalities

The document frequency and the cardinality of each databases are used to compute the the cue validity of each jth term in the ith databases using the formula [MYL96]

$$CV_{ij} = \frac{\frac{df_{ij}}{n_i}}{\frac{df_{ij}}{n_i} + \frac{\sum_{k \neq i}^{N} df_{kj}}{\sum_{k \neq i}^{N} n_k}}$$

The cue validity of terms in databases are given in table 2.12

	T_0	T_1	T_2	T_3
DB_0	0.523616	0.470588	0.530973	0.403361
DB_1	0.526316	0.533333	0.475248	0.477612
DB_2	0.526316	0.533333	0.535714	0.481203
DB_3	0.375000	0.519481	0.466019	0.529801
DB_4	0.526316	0.393443	0.466019	0.529801

Table 2.12: Cue validity

Then average value of cue validity for each term over all databases can be computed and is given in table 2.13

Terms	**Average CV**
T_0	0.496053
T_1	0.490036
T_2	0.494795
T_3	0.484356

Table 2.13: Average CV of terms

The variance of the cue validity is computed using the formula [MLY96]

$$CVV_j = \sum_{i=1}^{N} \frac{\left(CV_{ij} - ACV_j\right)^2}{N}$$

The variance of cue validity of each term i.e. CVV is given in table 2.14

Terms	**CVV**
T_0	0.003663
T_1	0.002865
T_2	0.001004
T_3	0.002149

Table 2.14: CVV values of each term

The rank of a database is computed using the formula [MYL96]

$$r_i = \sum_{j=1}^{M} CVV_j * df_{ij}$$

where M is the number of term. The databases and their ranks are given in table 2.15

Databases	Ranks
DB_0	0.041246
DB_1	0.045256
DB_2	0.046260
DB_3	0.040078
DB_4	0.041675

Table 2.15: Databases and their Ranks

The top databases are selected for answering the user query.

2.4 Experimental Results

The three database selection algorithms CORI, vGLOSS and CVV were implemented in C Language on a data set consisting of 40 databases containing documents and terms. First graphs were plotted one each for algorithm CORI, vGLOSS and CVV for selecting top-10, top-20, top-30 and top-40 databases. These graphs are shown in Figure 2.4.

These algorithms were compared on database score. The database scores of the three algorithms were compared for top-10, top-20, top-30 and top-40 databases. The graphs are shown in Figure 2.4. It can be inferred from the graph that the database score increase is close to be linear for selecting top-10, top-20, top-30 and top-40 databases for all the three algorithms.

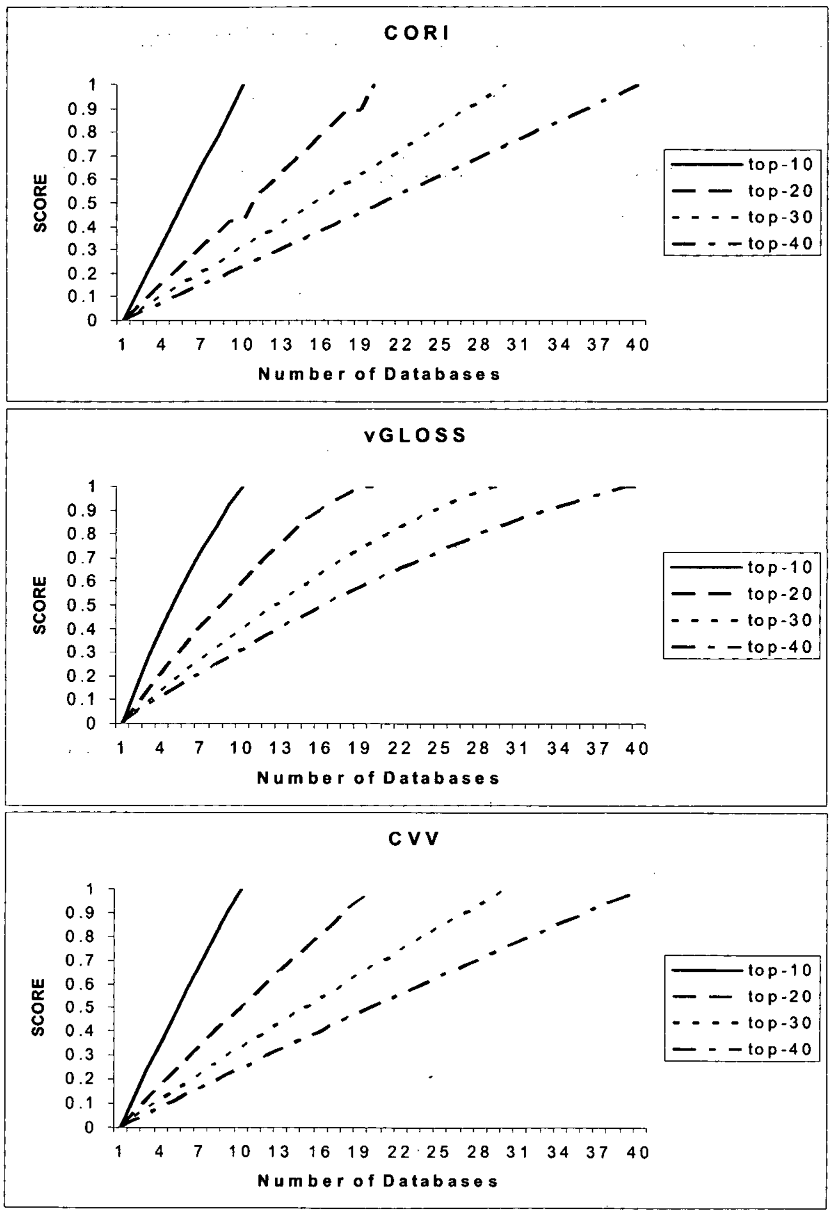

Figure 2.4: Selection of top-(10, 20, 30, 40) databases using CORI, vGLOSS and CVV

To compare the three database selection algorithms on the database score for top-10, top-20, top-30 and top-40 databases, graphs were plotted and are shown in Figure 2.5. It can clearly seen in these graphs that vGLOSS selects databases with higher scores as compared to CVV and CORI. CORI performs the worst when compared with the other two algorithms.

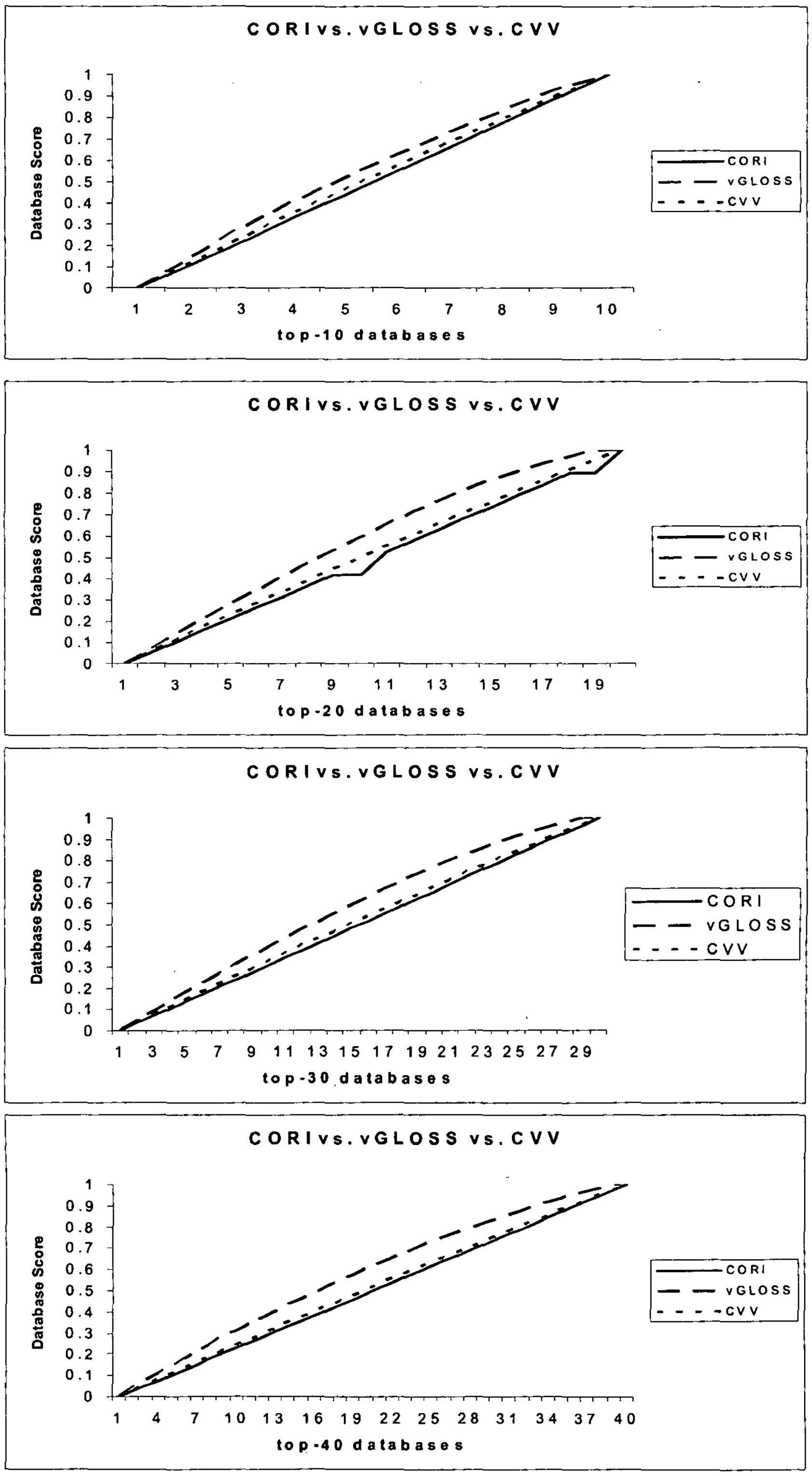

Figure 2.5: Comparison of CORI, vGLOSS and CVV for selecting of top-(10, 20, 30, 40) databases

The three database selection algorithms were compared on the number of databases in common selected by them while selecting top-5, top-10, top-15, top-20, top-25 and top-30 databases. The graphs for these comparisons were plotted and are shown in Figure 2.6.

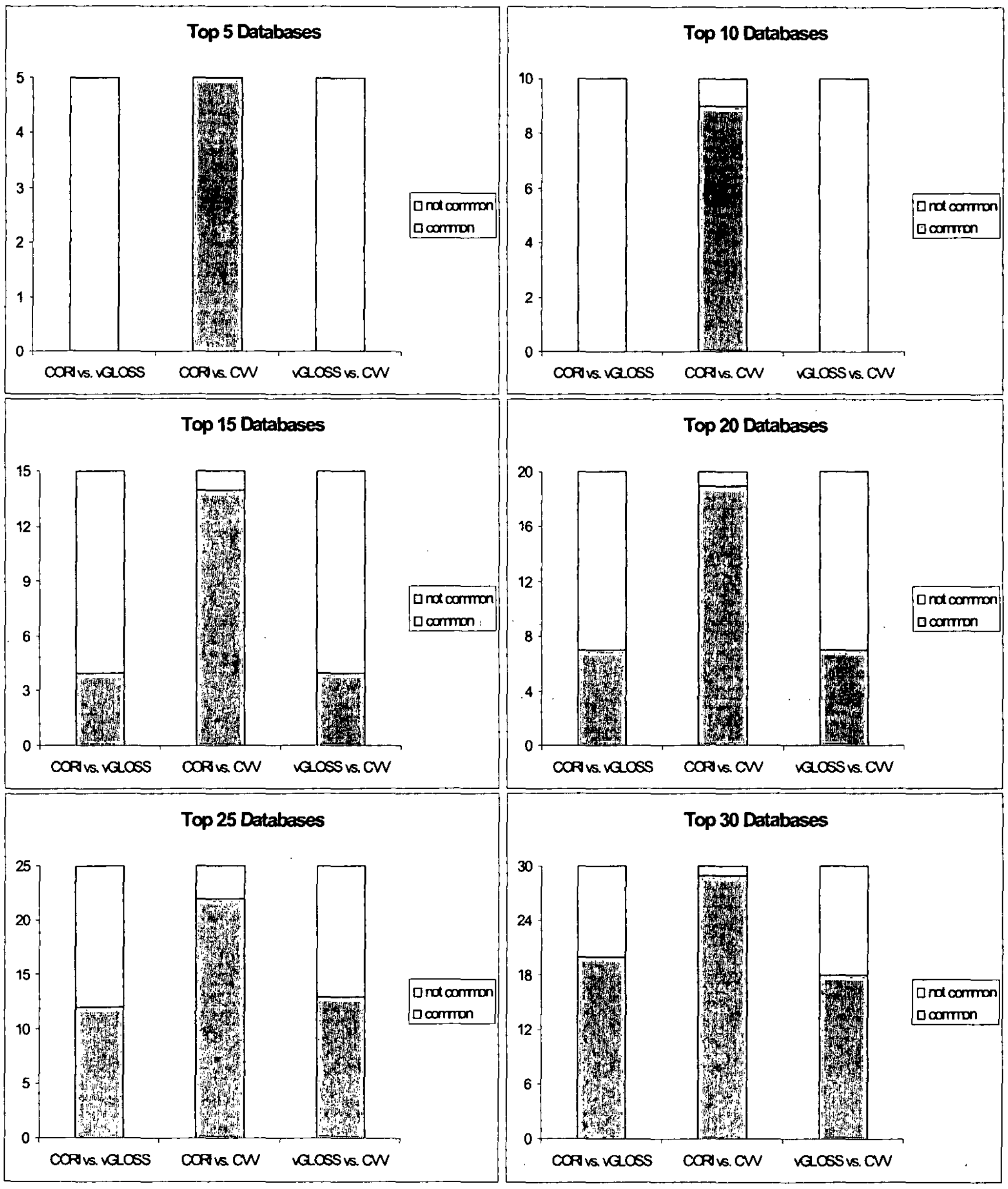

Figure 2.6: Selection of databases in common by CORI, vGLOSS and CVV for top-(5, 10, 15, 20, 25, 30) databases

The graphs show that the algorithm CORI and CVV have high number of databases in common selected by them whereas vGLOSS differs with CORI and CVV in terms of the database selected by it. Therefore, it can be inferred that CORI and CVV selects almost the same databases whereas this difference is high with databases selected by vGLOSS. Further, to view this behavior graph were plotted showing the number of databases in common selected by the three algorithms. The graph is shown in Figure 2.7. It can be clearly seen in the graph that CORI and CVV are similar with respect to the databases selected by them.

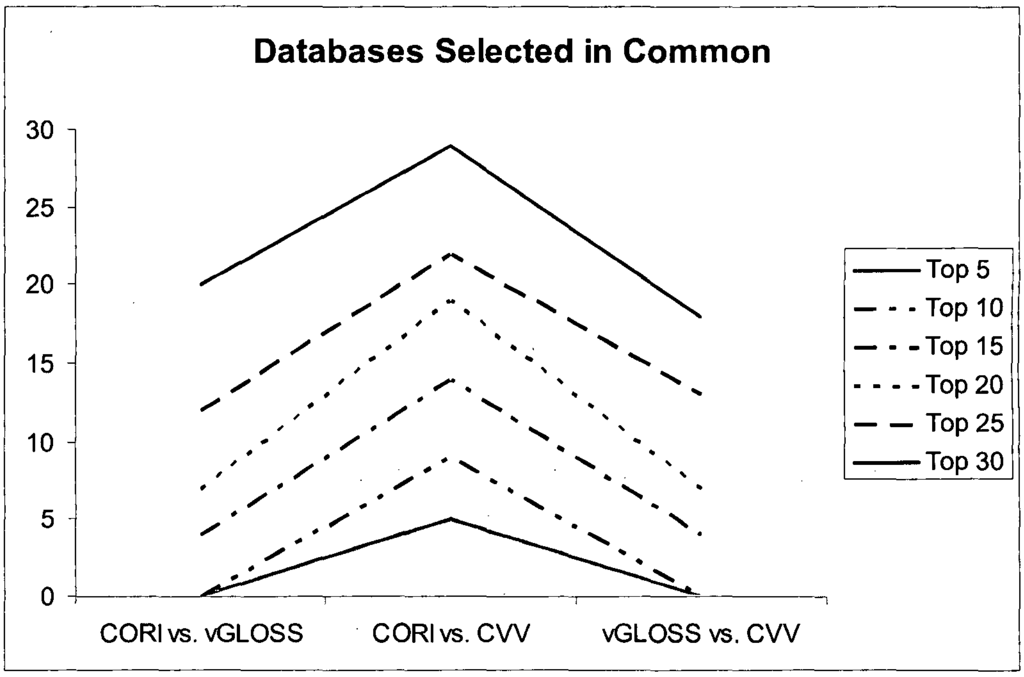

Figure 2.7: Comparing CORI, vGLOSS and CVV for selecting databases in common

Chapter 3

Result Merging

The main objective of the Result Merging is to merge the document from the selected databases in such a way so that the most relevant document appears at the top. The user query is posed on the selected databases. Based on the underlying similarity methods, each database produces relevant documents as output. Result merger component of metasearch engine is responsible to sort the list according to global similarity measure. In this chapter, the result merging algorithms LMS, CORI and SR Result merging are compared on performance parameters. The algorithms of each of these methods along with an example are discussed. This is followed by experimental based comparisons of these methods.

3.1 LMS Results Merging Approach [RAS03]

The LMS algorithm takes selected databases as input and produces sorted list of documents as output. The algorithm based on LMS Result Merging [RAS03] is shown in Figure 3.1.

```
Input: Sorted list of databases D[S, n, l], Query with l terms q[l]
Output: Sorted list of t documents d[t]
Method:
        Step 1: Repeat step (1) to (5) for each selected i^th database,
        Step 2: Calculate the term frequency (tf) of each term in selected databases
            For (each selected j^th databases)
             For (each k^th term in selected databases)
              $tf_{jk} = \log(1+\frac{n(d,t)}{n(d)})$ ; n (d, t) is number of occurrence of term t in document d and n(d) is total
              number of terms in document d
        Step 3: Calculate the inverse document frequency (idf)
            For (each selected j^th documents)
              For (each k^th term in documents)
              $idf_{ij} = \log(N/n_i)$; N is the number of documents in i^th database and $n_i$ is the number of
              documents containing j^th term
        Step 4: For (each selected j^th documents)
             For (each k^th term in documents)
              $w_{jk} = df_{jk} * idf_{jk}$ weight of each term in each document
        Step 5: Calculate cosine similarities of a document with respect to query
              $Sim_i(q, d_j) = \frac{\sum_{k=1}^{l} q_k * w_{jk}}{|q||d|}$
        Step 6: Calculate the number of relevant documents in i^th database l[i]
             For (each i^th database)
              For (each j^th document)
              if ($Sim_i(q,d_i) > T_{Threshold}$) ;
              l[i]=l[i]+1;
        Step 7: Calculate $S_i$ for each database i
              $S_i = \log[1 + \frac{l_i * k}{\sum_{i=1}^{|C|} l_i}]$
              K is a constant (set to 600 in evaluations) and C is number of selected database
        Step 8: Calculate weight ($w_i$) for each database i
              $w_i = 1 + \left[\frac{S_i - \bar{S}}{\bar{S}}\right]$ ; where $\bar{S}$ is the mean collection score
        Step 9: Final document score can be updated as
             New score[i][j]=$Sim_i(q,d_j) * w[i]$ ;
        Step 10: Sort the document based on new score values
             Generate top-t documents list as output
```

Figure 3.1: Algorithm based on LMS Merging [RAS03]

In LMS algorithm [RAS03], list of selected databases are given as input to the algorithm. In step2, the term frequency of each term of all document in selected databases are computed. The inverse document frequency of all term in each document is computed in step 3. Step 4 computes the weight of each term in each document. This term weight is computed by multiplying tf and idf of each term in a document. In step5, similarity of a document with respect to the query is evaluated and numbers of relevant document are identified in step 6. Step7 assign score to each database based on number of relevant documents in a database. The weight for each database is calculated in step 8. The score of the document is multiplied with new weight to evaluate the new score for a document in step9. The documents are sorted in descending order of their scores in step 10. The top documents are then displayed to the user.

3.1.1 An Example

Consider the list of databases selected by CORI selection approach (table 2.7 in Chapter 2). The list of databases is given in table 3.1.

Databases	Similarity of DB
DB_2	0.414425
DB_1	0.414344
DB_0	0.414261
DB_3	0.414261
DB_4	0.414261

Table 3.1: list of databases

Consider the top three databases such as DB_2, DB_1, and DB_0. Next, the document score of these selected databases is computed using the cosine similarity [RB04] between the document and query. The document score of these selected databases is given in table 3.2

	Doc_0	Doc_1	Doc_2	Doc_3	Doc_4
DB_2	0.457657	0.457657	0.000000	0.457657	0.457657
DB_1	0.457657	0.715162	0.725796	0.568774	0.667025
DB_0	0.000000	0.652078	0.596110	0.507540	0.494404

Table 3.2: Similarities of all document w.r.t query

These document score will be updated by multiplying by a factor [RAS03]

$$W_i = 1 + \left[\frac{S_i - \bar{S}}{\bar{S}}\right]$$

where S_i is the score of i^th database and is computed as

$$S_i = \log\left(1 + \frac{l_i * K}{\sum_{j=1}^{|c|} l_j}\right)$$

where l_i is the number of relevant document in ith database. l[2]=4, l[1]= 5 and l[0]=4. The top-five documents along with their scores are given in table 3.3

Document	Scores
DB1:doc2	0.746078
DB1:doc1	0.735147
DB1:doc4	0.685665
DB0:doc1	0.642967
DB0:doc2	0.587781

Table 3.3: Top 5 documents

These 5 documents, which are considered to be relevant, are displayed to the user.

3.2 CORI Results Merging Model [NF03]

The algorithm based on CORI Result Merging [NF03] is shown in Figure 3.2.

```
Input: D [m, n, l] where m is number of database, n is number of documents and l is number of terms,q[l]
Output: Ranked list of topmost databases
Method:
    Step1: Normalize, the database score on the scale of [0, 1]
        For (each ith database)
            $C'_i := \frac{C_i - C_{min}}{C_{max} - C_{min}}$; where $C_{min}$ is minimum collection score, $C_{max}$ is maximum
            collection score and $C'_i$ normalized score of ith database
    Step 2: Normalize, the documents score on the scale of [0, 1]
        For (each jth document)
            $D'_{ij} := \frac{D_{ij} - D_{min}}{D_{max} - D_{min}}$; where $D_{min}$ is minimum document score, $D_{max}$ is
            maximum document score and $D'_{ij}$ is normalized score of jth document of ith
            database
    Step 3: Calculate, the new value of the document score
            $D''_{ij} := \frac{1.0D'_{ij} + 0.4C'_i . D'_{ij}}{1.4}$; where $D''_{ij}$ is the new score of jth document in ith
            database
    Step 4: Sort documents on the basis of new document score (D")
        Generate top-t documents
```

Figure 3.2: Algorithm based on CORI Result Merging [NF03]

CORI Results Merging Model takes top rank databases as input [NF03]. The score of the databases is normalized on the scale of [0, 1] in step 1 and document similarity score (based on the cosine similarity measure between document vector and query vector) is normalized on the scale [0, 1] in step2. In step3, the new document score can be calculated by [NF03]

$$D'' := \frac{1.0D' + 0.4C'.D'}{1.4}.$$

Finally, the documents are sorted based on the new score for the document in step4. The algorithm produces the top-t documents as output.

3.2.1 An Example

The top five selected databases list, given in table 3.1, are normalized on the scale [0, 1]. The database and their normalized score are given in table 3.5.

Databases	Normalized Score of DB
DB_2	0.000000
DB_1	0.509249
DB_0	1.000000
DB_3	0.000000
DB_4	0.000000

Table 3.5: Normalized Score on Scale [0, 1]

Consider databases DB_2, DB_1 and DB_0. The cosine similarity of a document of each these selected databases with respect to query is given in table 3.6.

	Doc_0	Doc_1	Doc_2	Doc_3	Doc_4
DB_2	0.457657	0.457657	0.000000	0.457657	0.457657
DB_1	0.457657	0.715162	0.725796	0.568774	0.667025
DB_0	0.000000	0.652078	0.596110	0.507540	0.494404

Table 3.6: similarities of document w.r.t query

The documents maximum similarity score and minimum similarity score are

D_{max}: =0.725796

D_{min}: =0.000000

The documents scores are normalized using the formula [NF03]

$$D' := \frac{D - D_{min}}{D_{max} - D_{min}}$$

The normalized document score on [0, 1] is given in table 3.7.

	Doc_0	**Doc_1**	**Doc_2**	**Doc_3**	**Doc_4**
DB_2	0.630559	0.630559	0.000000	0.630559	0.630559
DB_1	0.542145	0.847188	0.859785	0.673775	0.790165
DB_0	0.000000	0.641736	0.586656	0.499491	0.486563

Table 3.7: Normalize document score on [0, 1]

The document scores are updated using the formula [NF03]

$$D'' := \frac{1.0D' + 0.4C'.D'}{1.4}$$

The updated document scores are given in table 3.8

	Doc_0	**Doc_1**	**Doc_2**	**Doc_3**	**Doc_4**
DB_2	0.630559	0.630559	0.000000	0.630559	0.630559
DB_1	0.542145	0.847188	0.859785	0.673775	0.790165
DB_0	0.000000	0.641736	0.586656	0.499491	0.486563

Table 3.8: Updated document score

The top five documents selected are given in table 3.9.

Documents	**Score**
database[1]:doc[2]	0.859785
database[1]:doc[1]	0.847188
database[1]:doc[4]	0.790165
database[1]:doc[3]	0.673775
database[1]:doc[1]	0.641736

Table 3.9: Top five Documents

These top documents are then displayed to the user.

3.3 SR SCORE Merging [LXG03]

The algorithm based on SR Score Merging [LXG03] is shown in Figure 3.3.

```
Input: Sorted list of databases
Output: Sorted list of documents
Method:
    Step1: databases should be rank based on the score only top rank document is to be selected for selecting
    their document
    Step2: document of the selected databases should be rank based on the document similarity measure of the
    document and query.
    Step3: update the document score based on the product of database score and document score with the
    database.
    Step4: sort the document list.and output the top t document list.
```

Figure 3.3: Algorithm based on SR Result Merging [LXG03]

In SR SCORE Merging [LXG03], the selected database information and query information forms input to the SR merging algorithm. In step 1, databases are ranked based on their scores and top databases are considered for selecting documents. In step2, the similarity of all documents in selected databases with respect to the query is computed. In step3, a score is assigned to document. This score is computed as the product of document score and databases score. The documents are sorted and the top-t documents are produced as output in step 4.

3.3.1 An Example

Consider the top-three databases DB_2, DB_1 and DB_0 from the databases selected, as given in table 3.1, using the CORI Selection approach.

The similarity score of the document with respect to query is given in table in table 3.10

	Doc_0	Doc_1	Doc_2	Doc_3	Doc_4
DB_2	0.457657	0.457657	0.000000	0.457657	0.457657
DB_1	0.457657	0.715162	0.725796	0.568774	0.667025
DB_0	0.000000	0.652078	0.596110	0.507540	0.494404

Table 3.10: similarities of all document w.r.t query

Next, document score is computed as product of document score and databases score. The new document score is given in table 3.11

	Doc0	Doc1	Doc2	Doc3	Doc4
DB_2	0.189664	0.189664	0.000000	0.189664	0.189664
DB_1	0.189627	0.296323	0.300072	0.235668	0.276377
DB_0	0.000000	0.270130	0.246945	0.210254	0.204812

Table 3.11: New score of all document w.r.t query

The top five document produced by SR merging is given in table 3.12

Documents	Score
database[1]:doc[2]	0.300072
database[1]:doc[1]	0.189627
database[1]:doc[4]	0.276377
database[0]:doc[1]	0.270130
database[0]:doc[2]	0.246945

Table 3.12: Top five Documents

These top five documents are displayed to the user.

3.4 Experimental Results

The three result merging algorithms LMS, CORI, and SR were implemented in C Language on a data set consisting of 40 databases containing documents and terms. First graphs were plotted one each for algorithm LMS, CORI and SR for selecting top-10 documents using the top-10, top-20, top-30 and top-40 databases selected using CORI database selection algorithm. These graphs are shown in Figure 3.4. The graphs show that there is no significant change in the documents score of top-10 documents selected from top-(10, 20, 30, 40) database selected by CORI database selection.

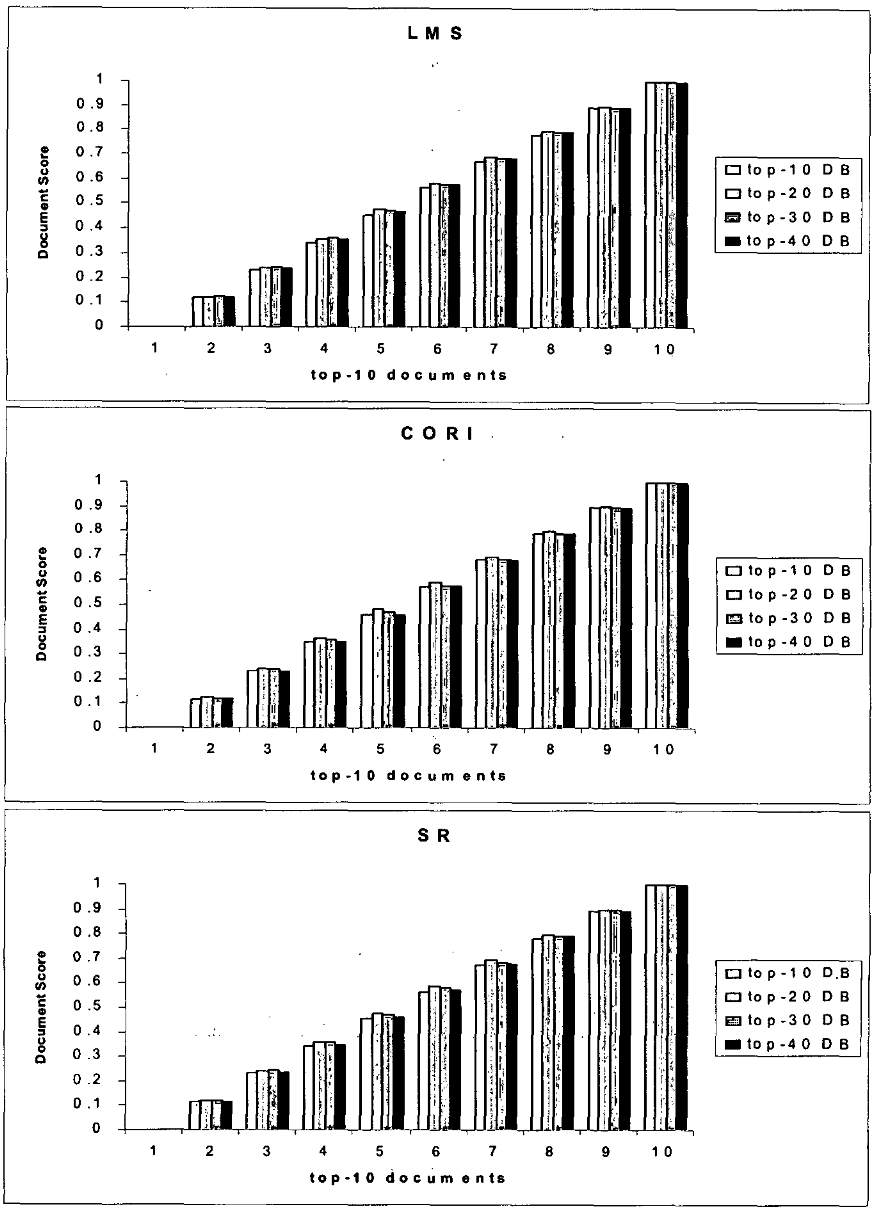

Figure 3.4: Selection of top-10 documents using LMS, CORI and SR

To compare the three results merging algorithms on the document score for selecting top-10 documents from top-(10, 20, 30, 40) databases selected by algorithm CORI. The graphs were plotted and are shown in Figure 3.5. The graphs show that CORI result merging algorithm performs slightly better than the LMS and SR merging algorithms.

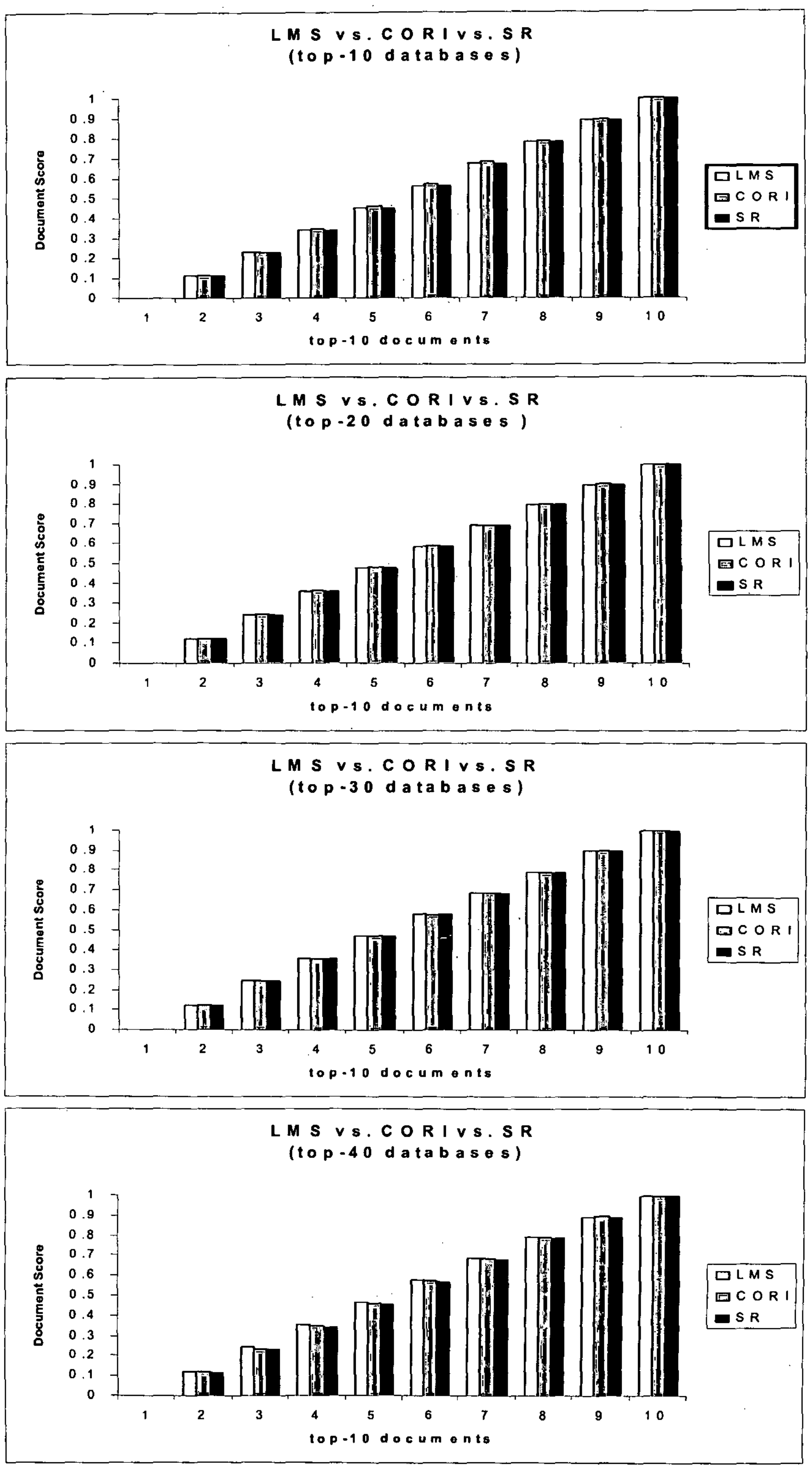

Figure 3.5: Comparison of LMS, CORI and SR for selecting of top-10 documents

The three result merging algorithms were compared on the number of documents in common selected by them while selecting top-5, top-10, top-15, top-20, top-25 and top-30 documents from top-40 databases selected by CORI. The graphs for these comparisons were plotted and are shown in Figure 3.6.

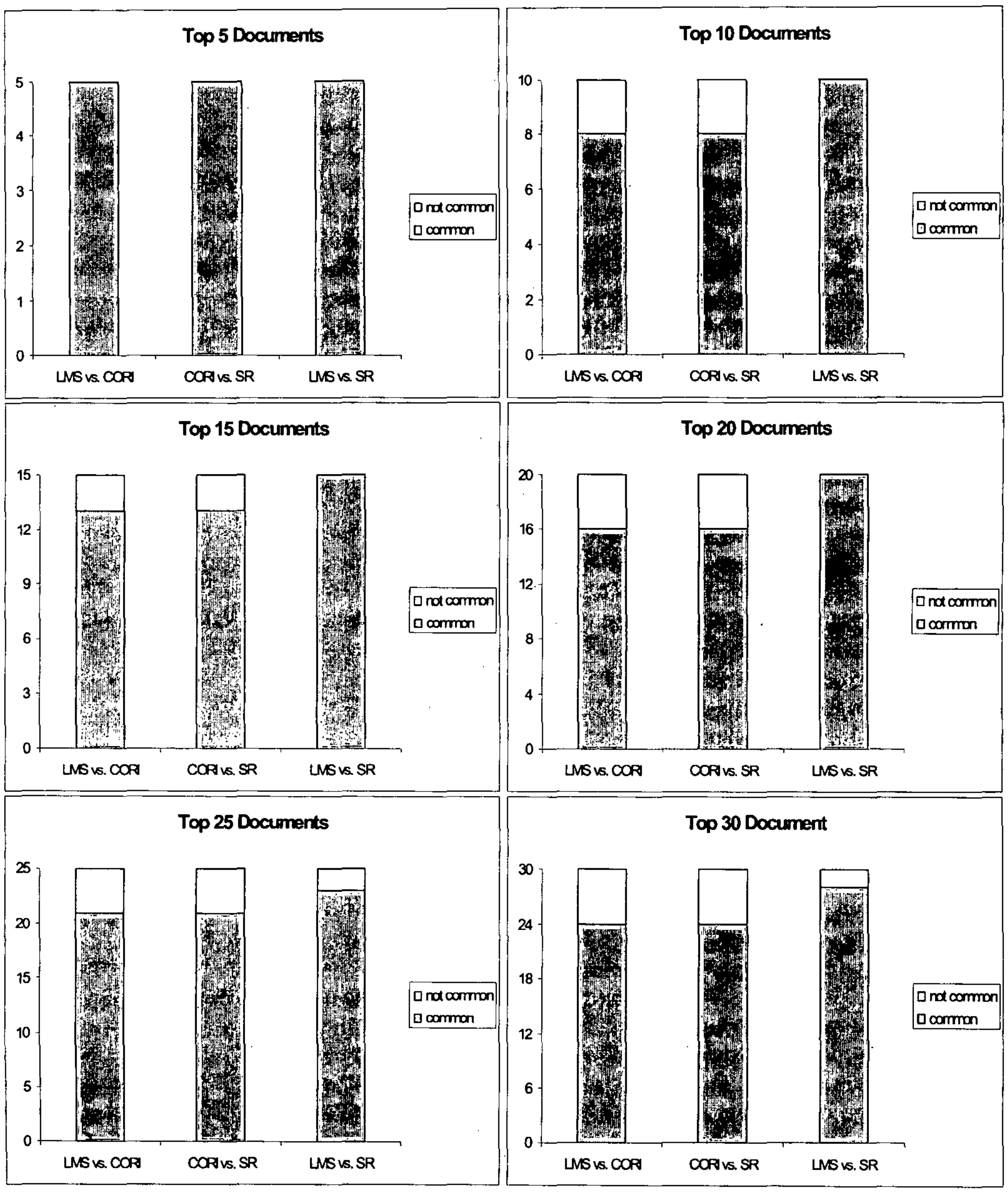

Figure 3.6: Selection of documents in common by LMS, CORI and SR for top-(5, 10, 15, 20, 25, 30) documents

The graphs show that the algorithm LMS and SR have high number of documents in common selected by them whereas CORI differs with LMS and SR in terms of the documents selected by it. Therefore, it can be inferred that LMS and SR selects almost the same documents whereas this difference is high with documents selected by CORI. Further, to view this behavior graph were plotted showing the number of documents in common selected by the three algorithms. The graph is shown in Figure 3.7. It can be clearly seen in the graph that LMS and SR are similar with respect to the documents selected by them.

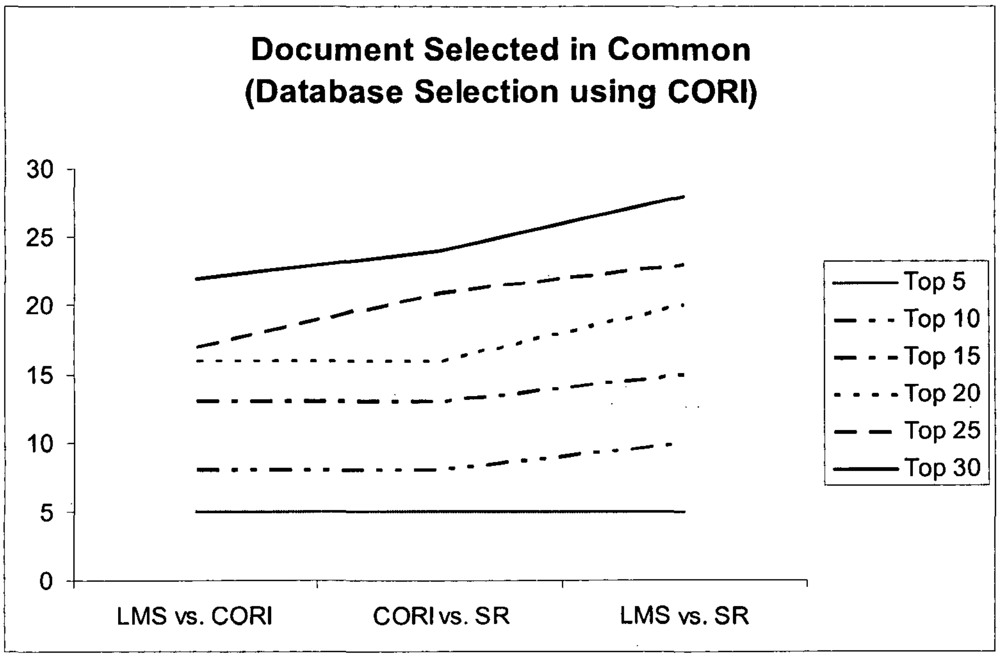

Figure 3.7: Comparing LMS, CORI and SR for selecting documents in common

Chapter 4

Metasearch Retrieval

Using the selection and the merging techniques given in chapter 2 and chapter 3 respectively, the following metasearch retrieval of documents are possible.

1. CORI Collection Selection [MYL02] with LMS Result Merge[RAS03]
2. CORI Collection Selection [MYL02] with CORI Result Merge [NF03]
3. CORI Collection Selection [MYL02] with SR Result Merge [LXG03]
4. vGLOSS Collection Selection [GGM95]with LMS Result Merge [RAS03]
5. vGLOSS Collection Selection [GGM95] with CORI Result Merge [NF03]
6. vGLOSS Collection Selection [GGM95] with SR Result Merge [LXG03]
7. CVV Collection Selection [MYL02] with LMS Result Merge [RAS03]
8. CVV Collection Selection [MYL02] with CORI Result Merge [NF03]
9. CVV Collection Selection [MYL02]with SR Result Merge [LXG03]

The metasearch retrieval options 1, 2 and 3 have already been dealt with in chapter 3. The remaining combinations are discussed next.

4.1 vGLOSS Collection Selection with LMS Result Merge

First the vGLOSS Collection Selection [GGM95] is used to find out the top three databases. The LMS Result Merging [RAS03] is then used to retrieve the top five documents.

For selecting the top three databases, first the cosine similarity of all documents with respect to query is computed and is shown in table 4.1

	Doc_0	**Doc_1**	**Doc_2**	**Doc_3**	**Doc_4**
DB_0	0.0000000	0.652078	0.596110	0.507540	0.494404
DB_1	0.457657	0.715162	0.725796	0.568774	0.667025
DB2	0.457657	0.457657	0.000000	0.457657	0.457657
DB_3	0.605235	0.568774	0.568774	0.592228	0.457657
DB_4	0.568774	0.711777	0.680501	0.659872	0.000000

Table 4.1: Similarities of all Document w.r.t Query

Next, the goodness of the databases is computed and is given in table 4.2

Databases	**Goodness**
DB_1	3.134414
DB_3	2.792668
DB_4	2.620924
DB_0	2.250131
DB_2	1.830628

Table 4.2: Goodness score of database

Top three selected databases are DB_1, DB_3, and DB_4. Next, apply LMS document merge to merge the document of these top three databases. The cosine similarity between the documents of these selected three databases and the query is computed. These are given in table 4.3.

	Doc_0	**Doc_1**	**Doc_2**	**Doc_3**	**Doc_4**
DB_1	0.457657	0.715162	0.725796	0.568774	0.667025
DB_3	0.605235	0.568774	0.568774	0.592228	0.457657
DB_4	0.568774	0.711777	0.680501	0.659872	0.000000

Table 4.3: Display similarities of documents w.r.t query

Documents that satisfy some threshold are considered relevant. Number of relevant document selected from each databases is computed and for each databases a database score is computed. Then database score is used to compute the weight of each database. The document score is then updated as the product of the weight of the databases and document score. The top five documents are given in table 4.4

Documents	Score
database[1]:doc[2]	0.735933
database[1]:doc[1]	0.725150
database[4]:doc[1]	0.691895
database[1]:doc[4]	0.676341
database[4]:doc[2]	0.661492

Table 4.4: Top five Documents

4.2 vGLOSS Collection Selection with CORI Result Merge

First the vGLOSS Collection Selection [GGM97] is used to find out the top three databases. The CORI Result Merging [NF03] is then used to retrieve the top five documents. As mentioned above, the top three databases selected by vGLOSS are DB_1, DB_3 and DB_4. Next, apply the CORI Result Merging. First, the database is normalized on the scale of [0, 1] as given in table 4.5.

DB	Normalized Score of DB
DB_0	0.321757
DB_1	1.000000
DB_2	0.000000
DB_3	0.737881
DB_4	0.606154

Table 4.5: Normalized database on scale [0, 1]

The document score is normalized on the scale of [0,1] and is given in table 4.6.

	Doc_0	Doc_1	Doc_2	Doc_3	Doc_4
DB_1	0.630558	0.985348	1.000000	0.783655	0.919025
DB_3	0.833891	0.783655	0.783655	0.815970	0.630558
DB4	0.783655	0.980684	0.937592	0.909170	0.000000

Table 4.6: Document score on scale [0, 1]

The updated document score is given in table 4.7.

D_{max}:=0.725796, D_{min}:=0.000000

	Doc_0	Doc_1	Doc_2	Doc_3	Doc_4
DB_1	0.630558	0.985348	1.000000	0.783655	0.919025
DB_3	0.771439	0.724966	0.724966	0.754861	0.583334
DB_4	0.694957	0.870330	0.832087	0.806863	0.000000

Table 4.7: Updated document score

Sort the document list and output the top five documents as given in table 4.8

Documents	Score
database[1]:doc[2]	1.000000
database[1]:doc[1]	0.985348
database[1]:doc[4]	0.919025
database[4]:doc[1]	0.870330
database[4]:doc[2]	0.832087

Table 4.8: Top five Documents

4.3 vGLOSS Collection Selection with SR Result Merge

First the vGLOSS Collection Selection [GGM95] is used to find out the top three databases. The SR Result Merging [LXG03] is then used to retrieve the top five documents.

As mentioned above, the top three databases selected by vGLOSS are DB_1, DB_3 and DB_4. Next, the SR Result Merging is applied. First, the cosine similarity of the document with respect to query is computed as given in table 4.9.

	Doc_0	Doc_1	Doc_2	Doc_3	Doc_4
DB_1	0.457657	0.715162	0.725796	0.568774	0.667025
DB_3	0.605235	0.568774	0.568774	0.592228	0.457657
DB_4	0.568774	0.711777	0.680501	0.659872	0.000000

Table 4.9: Similarities of documents in databases

The new score value is computed as a product of document score and the corresponding database score and is given in table 4.10.

	Doc_0	Doc_1	Doc_2	Doc_3	Doc_4
DB_1	1.434486	2.241613	2.274945	1.782773	2.090732
DB_3	1.690220	1.588396	1.588396	1.653896	1.278084
DB_4	1.490713	1.865513	1.7835414	1.729474	0.000000

Table 4.10: New score of documents in selected databases

The documents are sorted based on the new scores. Finally, the top five documents are produced as output. The top five documents are given in table 4.11.

Documents	Score
database[1]:doc[2]	2.274945
database[1]:doc[1]	2.241613
database[1]:doc[4]	2.090732
database[4]:doc[1]	1.865513
database[4]:doc[2]	1.783541

Table 4.11: Top five Documents

4.4 CVV Collection Selection with LMS Result Merge

First the CVV Collection Selection [MYL02] is used to find out the top three databases. The LMS Result Merging [RAS03] is then used to retrieve the top five documents. The CVV collection selection algorithms first compute the document frequency of each term as given in table 4.12.

	T_0	T_1	T_2	T_3
DB_0	5	4	5	3
DB_1	5	5	4	4
DB_2	5	5	5	4
DB_3	3	5	4	5
DB_4	5	3	4	5

Table 4.12: Document Frequency

The cardinality of the databases is then computed as the number of documents contained in each database as given in table 4.13.

Database	Cardinality
DB_0	5
DB_1	5
DB_2	5
DB3	5
DB_4	5

Table 4.13: Cardinality of each database

Using the document frequency and the cardinality of each database, the cue validity of each term is computed and is given in Table 4.14

	T_0	T_1	T_2	T_3
DB_0	0.523616	0.470588	0.530973	0.403361
DB_1	0.526316	0.533333	0.475248	0.477612
DB_2	0.526316	0.533333	0.535714	0.481203
DB_3	0.375000	0.519481	0.466019	0.529801
DB_4	0.526316	0.393443	0.466019	0.529801

Table 4.14: cue validity of each term

Next, average of these cue validity (ACV) is computed for each query term over all the databases. These are given table 4.15

Term	ACV
T_0	0.496053
T_1	0.490036
T_2	0.494795
T_3	0.484356

Table 4.15: ACV of terms

Next variance of cue validity CVV is computed and is given table 2 The CVV value are given in table 4.16

Term	ACV
T_0	0.003663
T_1	0.002865
T_2	0.001004
T_3	0.002149

Table 4.16: CVV of terms

Using CVV and document frequency, the rank of the database is computed. The sorted list of databases based on their rank is given in table 4.17

Databases	Rank
DB_2	0.046260
DB_1	0.045256
DB_4	0.041675
DB_0	0.041246
DB_3	0.040078

Table 4.17: Top five database

The top three databases selected are DB_2, DB_1, and DB_4. Next, the LMS document merge to merge the document of these top three databases is applied. The cosine similarity between the documents of these selected databases (DB_2, DB_1 and DB_4) and the query is computed. These are given in table 4.18.

	Doc_0	Doc_1	Doc_2	Doc_3	Doc_4
DB_2	0.457657	0.457657	0.000000	0.457657	0.457657
DB_1	0.457657	0.715162	0.725796	0.568774	0.667025
DB_4	0.568774	0.711777	0.680501	0.659872	0.000000

Table 4.18: Display Similarities of all document w.r.t Query

Documents that satisfy some threshold are considered relevant. Number of relevant document selected from each databases is computed and for each databases a database score is computed. Then database score is used to compute the weight of each database. The document score is then updated as the product of the weight of the databases and document score. The top five documents are given in table 4.19.

Documents	Score
database[1]:doc[2]	0.746078
database[1]:doc[1]	0.735147
database[4]:doc[1]	0.701832
database[1]:doc[4]	0.685665
database[4]:doc[2]	0.670992

Table 4.19: Top five document

4.5 CVV Collection Selection with CORI Result Merge:

First the CVV Collection Selection [MYL02] is used to select top three databases. The CORI Result Merging [NF03] is then used to retrieve the top five documents.

As mentioned above, the CVV collection approach selects the top three databases as DB_2, DB_1 and DB_4. Next, CORI Result Merging is applied. First, the databases are normalized on the scale of [0, 1] as given in table 4.20.

DB	Normalized Score of DB
DB2	1.000000
DB1	0.837593
DB4	0.258330
DB0	0.188935
DB3	0.000000

Table 4.20: Normalized database on scale [0, 1]

The document score is then normalized on the scale [0, 1] and is given in table 4.21.

	Doc_0	Doc_1	Doc_2	Doc3	Doc_4
DB_2	0.630559	0.632559	0.000000	0.630559	0.630559
DB_1	0.630559	0.985348	1.000000	0.783655	0.919025
DB_4	0.783655	0.980684	0.937592	0.909170	0.000000

Table 4.21: Normalize document score on scale [0, 1]

The document score is updated. The updated document score is given in table 4.22.

	Doc_0	Doc_1	Doc_2	Doc_3	Doc_4
DB_2	0.630559	0.630559	0.000000	0.630559	0.630559
DB_1	0.601294	0.939625	0.953598	0.747291	0.876380
DB_4	0.617594	0.772871	0.738910	0.716511	0.000000

Table 4.22: updated score of documents on scale [0, 1]

The documents are sorted base on updated document score and are given in table 4.23.

Documents	Score
database[1]:doc[2]	0.953598
database[1]:doc[1]	0.939625
database[1]:doc[4]	0.876380
database[4]:doc[1]	0.772871
database[1]:doc[3]	0.747291

Table 4.23: Top five documents

4.6 CVV Collection Selection with SR Result Merge

First the CVV Collection Selection [MYL02] is used to select top three databases. The SR Result Merging [LXG03] is then used to retrieve the top five documents.

As mentioned above, CVV collection selection approach selects the top three databases as DB_2, DB_1 and DB_4. Next, SR Result Merging is applied. The cosine similarity of the document w.r.t query is computed and is given in table 4.24.

	Doc_0	**Doc_1**	**Doc_2**	**Doc_3**	**Doc_4**
DB_2	0.457657	0.457657	0.000000	0.457657	0.457657
DB_1	0.457657	0.715162	0.725796	0.568774	0.667025
DB_4	0.568774	0.711777	0.680501	0.659872	0.000000

Table 4.24: Similarities of documents w.r.t Query

The new score for each database is computed as a product of the document score and the corresponding database score and is given in table 4.25.

	Doc_0	**Doc_1**	**Doc_2**	**Doc_3**	**Doc_4**
DB_2	0.046260	0.046260	0.000000	0.046260	0.046260
DB_1	0.020711	0.032365	0.032846	0.025740	0.030186
DB_4	0.023703	0.029663	0.028359	0.027500	0.000000

Table 4.25: New score of documents in selected databases

The documents are sorted on new score. The top five documents are produced as output. The top five documents are given in table 4.26.

Documents	**Score**
database[2]:doc[0]	0.046260
database[2]:doc[1]	0.046260
database[2]:doc[3]	0.046260
database[1]:doc[4]	0.046260
database[1]:doc[2]	0.032846

Table 4.26: Top five documents

4.7 Experimental Results

Graphs were plotted for top-k (k=5, 10, 15, 20, 25, 30) documents selection from databases selected by vGLOSS and CVV algorithms. First, graphs were plotted for selecting top-k documents using LMS, CORI and SR merging algorithms from databases selected by vGLOSS. The graphs are shown in Figure 4.1.

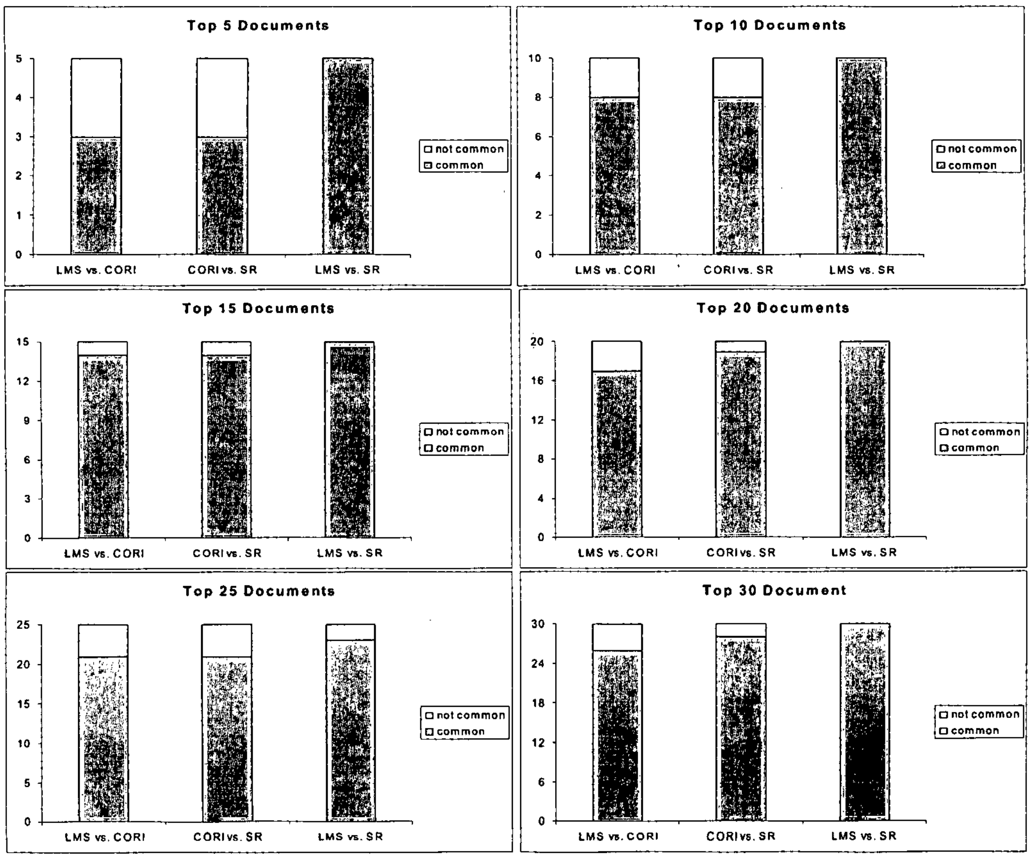

Figure 4.1: Selection of documents in common by LMS, CORI and SR for top-(5, 10, 15, 20, 25, 30) documents

It can be observed from the graphs that LMS and SR algorithms retrieve almost same documents whereas CORI selects lesser documents in common with LMS and SR. This difference can also be seen clearly in the graph, shown in Figure 4.2.

Graphs were plotted for selecting top-k documents using LMS, CORI and SR merging algorithms from databases selected by CVV. The graphs are shown in Figure 4.3.

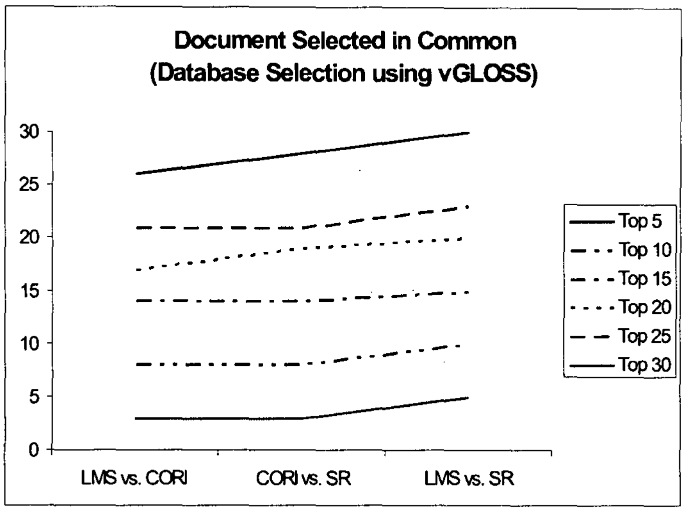

Figure 4.2: Comparing LMS, CORI and SR for selecting documents in common

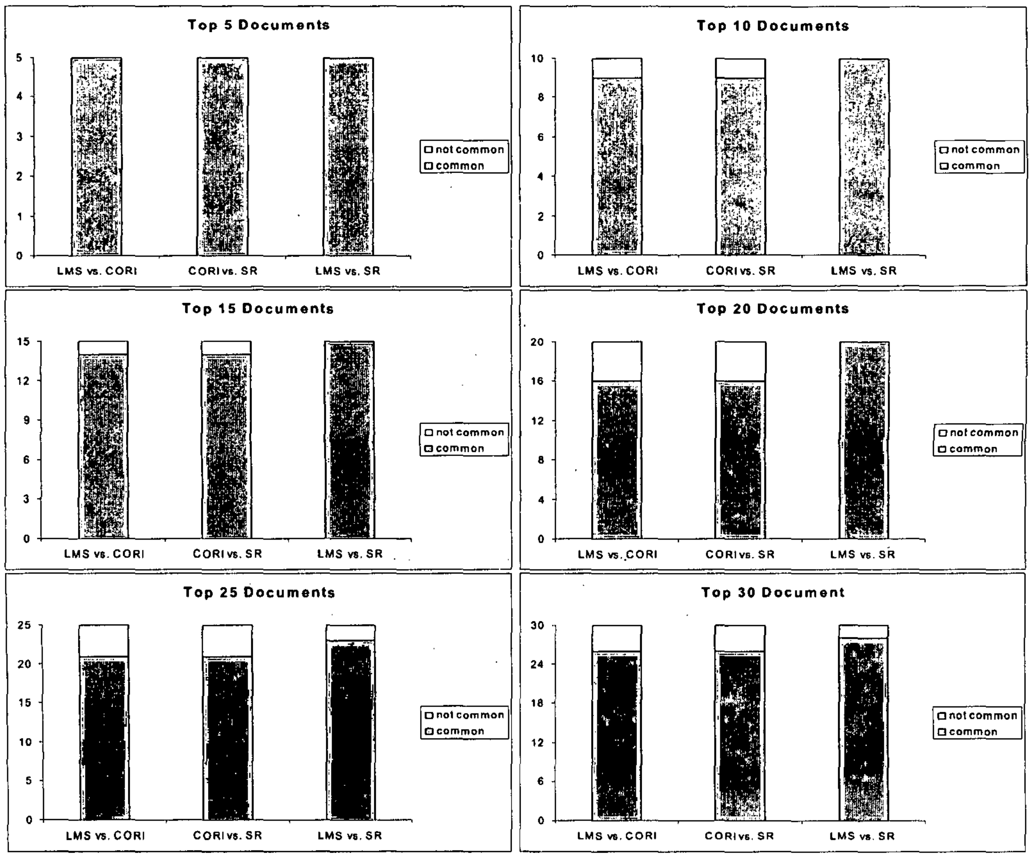

Figure 4.3: Selection of documents in common by LMS, CORI and SR for top-(5, 10, 15, 20, 25, 30) documents

It can be observed from the graphs that LMS and SR algorithms retrieve almost same documents whereas CORI selects lesser documents in common with LMS and SR. This difference can also be seen clearly in the graph, shown in Figure 4.4.

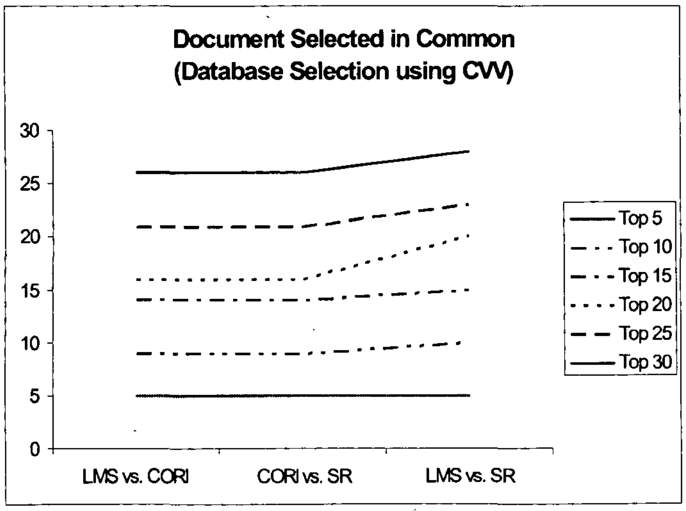

Figure 4.4: Comparing LMS, CORI and SR for selecting documents in common

Chapter 5

Conclusion

Most of the information on the Web is unstructured and dynamic in nature. Information retrieval systems are used to query such unstructured data. One of the main problems with information retrieval systems is scalability. This problem can be overcome using metasearch engine. Metasearch engine is a search engine over search engines. This dissertation focuses on retrieving information using metasearch. Metasearch engine comprises of two components namely database selection and result merging. In database selection, database relevant to a query are selected to limit the search space. The result merging merges relevant documents based on global similarity measure. In this dissertation, database selection and result merging techniques are studied with emphasis on database selection techniques like CORI, vGLOSS and CVV and result merging techniques like LMS, CORI and SR.

First, the database selection techniques CORI, vGLOSS and CVV were experimentally compared on parameters like database score and number of database in common selected by these techniques. The comparison shows that vGLOSS select databases with higher score as compared to CVV and CORI. Further, it was observed that CORI and CVV selected almost the same databases.

Next, the result merging techniques LMS, CORI and SR were experimentally compared on document score. It was observed that result merging technique CORI performed slightly better than LMS and SR result merging techniques.

Further, metasearch retrieval of information using all possible combination of the above-mentioned database selection and result merging techniques were experimentally compared on number of documents in common retrieved by these combinations. It was observed that LMS and SR retrieved almost same documents irrespective of the techniques used for database selection.

References

[ASB02] Abbaci, F., Savoy, J, and Beigbeder, M., A Methodology for Collection Selection in Heterogeneous Contexts.

[CLC95] Callan, J. Lu, Z. and Cropt, W.1995.Searching distributed collection with inference networks. In Proceeding of the ACM SIGIR Conference (Seattle, WA July1995), 21-28

[DH97] Drelinger, D. and Howe, A. 1997. Experience with Selecting Search engine Using Metasearch.ACM Trans. Inform. Syst. 15, 3(July), 195-222.

[FG99] Fan, Y. and Gauch, S. 1999. Adaptive agent for information gathering from multiple, distributed information source. In Proceeding of the 1999 AAAI Symposium on Intelligent Agent in Cyberspace, March 1999, 40-46.

[F97] Fuhr, N. A Decision –Theoretic Approach to Database Selection in Networked IR.

[GGM95] Gravano, L., and Garcia-Molina, H.1995. Generalizing gloss to vector space databases and broker hierarchies' .In proceeding of the International Conferences on Very Large Data Bases, August 1997, 196-205.

[GGM97] Gravano, L., Garcia-Molina AND Tomasic, A. GLOSS: Text Source Discovery over the Internet.

[HT99] Hawking, D. AND Thistlewaite, P. Methods for Information Server Selection, ACM Transaction on Information Systems, Vol.17, Jan 1999, Pages 40-76.

[LXG03] Lu, C., Xu, Y.AND Geva, S.Collection Profiling for Collection Fusion in Distributed Information Retrieval System

[MYL02] Meng, W., Yu C and Liu K. Building effective and efficient metasearch engine ACM Computing Surveys, Vol.34, No.1, March 2002.

[NF03] Nottelmann, H. AND Fuhr, N. Combining CORI and decision theoretic approach for advanced resource selection

[PSS07] Paltoglou, G., Salampasis, M.AND Satratzemi, M. Hybrid Result Merging.CIKM'07, Nov 2007

[RAS03] Rasolofo, Y., Abbaci, F. AND Savoy, J. Approaches to Collection Selection and Results Merging for Distributed Information Retrieval.

[RB04] R.Baeza-Yates and B.Ribeiro-Neto, Modern Information Retrieval, Pearson Education.

[SJ04] Souza, D.Zobel, J.Thom, A, J.Is CORI effective for Collection Selection? An Exploration of Parameter, Queries, and Data, In Proceeding of the 9th Australasian Document Computing Symposium, Dec 2004.

[SC03a] Si, L. and Callan, J. A Semi supervised Learning Method to Merge Search engine Results. ACM Transactions on information System, Vol.21, No.4, Oct 2003, pages 457-491.

[SC03b] Si, L. and Callan, J. The Effect of Database Size Distributed on Resource Selection Algorithms. SIGIR 2003 Ws Distributed IR LNCS 2924, pp.31-42, 2003.

[SCO02a] Si, L., Jin, R, Callan, J., and Ogilvie, P. A Language Modeling Framework for Resource Selection and Results Merging, CLKM'02, Nov 4-9, 2002.

[SC03] Si, L. and Callan J.Relevant Document Distribution Estimation Method for Resource Selection, SIGIR'03 July 28-Aug 1, 2003.

[SKS06] Silberschatz, A., Korth, H.F and Sudershan S.Database Management System Concept, Fifth Edition, McGraw- Hill International Edition.

[VGL95b] Voorhees, E., Gupta,N.K, and Laird, B.L. 1995b. Learning Collection fusion Strategies in Proceeding of the ACM SIGIR Conference (Seattle, WA July1995), 172-179.

[WMY01] Wu, Z., Meng, W. Yu,C. and Li,Z., Towards a highly scalable and effective metasearch engine. In proceeding of the tenth World Wide Web Conference, May 2001, 386-395.

[YL96] Yuwono, B. and Lee, D.1996. Search and ranking algorithm for locating resource on the World Wide Web. In Proceeding of the 5th International Conference on Data Engineering, Feb. 1996, 164-177.

[VGL95a] Voorhees, E.M., Gupta, N.K. and Laird, B.J. The Collection Fusion Problem. In Proceeding of the Third Text Retrieval Conference, March 1995.

[WC02] Wu, S. and Crestani, F. Data Fusion with Estimated Weights, CIKM'02, Nov 2002

[NF03] Nottelmann, H. and Fuhr, N.Evaluating Different Method of Estimating Retrieval Quality for Resource Selection, SIGIR'03, July 2003.

[TVG+95] Towell, G., Voorhees, E.M., Gupta, N.K.and Laird B.J. Learning Collection Fusion Strategies for Information Retrieval. Proceeding 1995 ACM SIGIR Conference on Research and Development in Information Retrieval.

[SC02] Si, Luo and Callan, J. Using Sampled Data and Regression to Merge Search Engine Results, SIGIR'02, Aug 2002.

[GGM94] Gravano, L., Garcia-Molina, H. and Tomasic, A., the Effectiveness of GLOSS for the Text Database Discovery Problem. In Proceeding of the 1994 ACM International conference on management of data (SIGMOD'94), May 04.

[KM94] Koster, M.1994.Aliweb: Archie-like indexing in the web. computer network and ISDN system.27,2 172-182.