6.2 A Historical progression:

Previous work can be partitioned into systems that combine information retrieval and DBMS together, as systems that extend relational DBMS to information retrieval functionality. We now describe each of these approaches in detail.

6.2.1 Combining seperate systems:

Several researchers proposed integrated solutions which consist of various a central layer of software to send requests to underlying DBMS and information retrieval systems [Schek and Pistor, 1982]. Queries are parsed and the structured portions are submitted as a query to the DBMS, while text portions of the query ~~are submitted as a query to the DBMS~~ are submitted to an information retrieval system. The results are combined and presented to the user. It does not take long to this software and since information retrieval systems and DBMS are readily available this is often seen as an attractive solution. The key advantage of this approach is that the DBMS and information retrieval system are commercial products that are continously improved upon by vendors. Additionally, software development costs are minimized. The disadvantage include poor data integrity, portability and run-time performance.

6.2.1.1 Data Integrity

Data integrity is sacrificed because the DBMS transaction log and the information retrieval transaction log are not coordinated if a failure occurs in the middle of an update transactions log are not coordinated if a failure occurs in the middle of an update transaction the DBMS will end in a state where the entire transaction is either completed or it is entirely undone. It is not possible to complete half of an update.

The information retrieval log would not know about the DBMS log. Hence, the umbrella application that coordinates work between the two systems must handle all recovery. Recovery

done with an application is typically easy prone and in many cases, applications simply ignore this coding. Hence if a failure should occur in the information retrieval system, the DBMS will not know about it. An update that must take place in both system can succeed in the DBMS, but fail in the information retrieval system. A partial update is clearly possible but is logically flawed.

6.2.1.2 portability.

portability is sacrificed because the query language is not standard. presently a standard information retrieval query language does not exist. However, some work is being done to standard information retrieval query languages. If one existed, it would require many years for widespread commercial acceptance to occur. The problem is that developers must be retrained each time a new DBMS and information retrieval system is brought in. Additionally system administration is far more difficult with multiple systems.

6.2.1.3 performance.

Run time performance suffers because of the lack of parallel processing and query optimization. Although most commercial DBMS have parallel implementations, most information retrieval system do not.

Query optimization exists in every relational DBMS. The optimizer's goal is to choose the appropriate access path to the data. A rule based optimizes is to uses pre-defined rules, while a cost-based optimizes estimates the cost of using different access paths and chooses the cheapest one. In either case, no rules exist for the unstructured portion of the query and no cost estimate could be obtained because the optimizer would be unaware of the acces

paths that may be chosen by the information retrieval system. Thus, any optimization that included both structured and unstructured data would have to be done by the umbrella application. This would be a complex process. The difficulties with such optimization were discussed by the authors who suggested this approach [Lynch and Stonebraker, 1988]. Hence, run-time performance could suffer due to a lack of parallel algorithms and limited global query optimization.

6.2.1.4 Extension to SQL.

Brapr, in an unpublished paper in 1974, proposed that SQL could be modified to support text. Subsequently, a series of papers between 1978 and 1981 were written that described several extensions to SQL. The SMART information retrieval prototype initially developed in the 1980's used the INGRES relational database system to store its data. These papers described extensions to support relevance ranking as well as Boolean searches. The authors focused on the problem of efficiently searching text in a RDBMS. They went on to indicate that the RDMS would store the inverted index in another table thereby making it possible to easily view the contents of the index. An information retrieval system typically hides the inverted index as simply an access structure that is used to obtain data. By storing the index as simply an access structured that is used to obtain data. By storing the index as a relation, the authors pointed out that users could easily view the contents of the index and make changes if necessary. The author mentioned extensions, such as RELEVANCE (*), that would compute the relevance of a document to a query using some pre-defined relevance function.

More recently a language called SQLX was used to access documents in a multimedia database. SQLX assumes that an

Initial cluster based search has been performed based on keyword SQLx extensions allow for a search of the result with special connector attributes that obviate the need to explicitly specify joins

6.2.2 User defined operations:

User defined operators that allow users to modify SQL by adding their own functions to the DBMS engine were described as early as. commercialization of this idea has given rise to several products including the Teradata multimedia object Manager, Oracle cartridges, IBM DB2 Text Extender, as well as features in Microsoft SQL server.

Ex:1 SELECTMAX

In the information retrieval domain, an operator such as proximity() could be defined to compute the result set for a proximity search. In this fashion the "spartan simplicity of SQL" is preserved, but users may add whatever functionality is needed.

Ex:2 SELECT DOC-Id
from DOC

This query can take advantage of an inverted index to rapidly identify the terms. To do this, the optimizer would need to be made aware of the new access method. Hence user defined functions also may require user-defined access methods.

6.2.2.1 Integrity.

User-defined operators allow application developers to add functionality to the DBMS rather than the application that uses the DBMS. This unfortunately opens the door for application developers to circumvent the integrity of the DBMS. For user defined operators to be efficient they must be linked into the same module as the entire DBMS, giving them access to the entire address space of the DBMS.

6.3 Information Retrieval as a Relational Application:

Wong with extensions to SQL started first in an unpublished paper and continued with several papers by Macleod and Crawford between 1978 and 1981

Initial extensions described by Macleod are based on the use of a QUERY relation that stores the terms in the query and an INDEX (Docid, term) relation that indicates which terms appear in which documents. The following query lists all the identifiers of documents that contain at least one term in

QUERY:

```
Ex: 5  SELECT DISTINCT
         FROM INDEX i, QUERY q
         WHERE i.term = q.term.
```

frequently used terms or stop terms are typically eliminated from the document collection. Therefore, a STOP-TERM relation may be used to store the frequently used terms. The STOP-TERM relation contains a single attribute. A query to identify documents that contain any of the terms in the query except those in the STOP-TERM relation is

```
Ex: 6  SELECT DISTINCT
         FROM INDEX i, QUERY q, STOP-TERM
         WHERE i.term = q.term AND
           i.term ≠ s.term.
```

Finally to implement a logical AND of the terms InputTerm1, InputTerm2, and inputTerm 3, Macleod and Crawford proposed the following

```
Ex 7: SELECT DOCID
        FROM INDEX
        WHERE term = inputTerm1
        INTERSECT
          SELECT DOCid
          FROM INDEX
        WHERE term = inputTerm2
        INTERSECT
        SELECT DOCid.
```

The query consists of three components. Each component results in a set of documents that contain a single term in the query. The INTERSECT keyword is used to find the intersection of the three sets. After processing an AND is implemented.

6.3.1 preprocessing

Input text is originally stored in source files either at remote sites or locally on CD-ROM for purposes of this discussion, it is assumed that the data files are in ASCII or can be easily converted to ASCII with SGML markers. SGML markers are a standard means by which different portions of the document are marked. The markers in the working example are found in the TIPSTER collection which was in previous years as the standard dataset for TREC. These markers begin with a < and end with a >

A preprocessor that reads the input file and output separate flat files is used. Each term is read and checked against a list of SGML markers.

6.3.2 A working Example

Throughout this chapter the following working example is used. Two documents are taken from the TIPSTER collection and modelled using relations. The documents contain both structured and unstructured data and are given below.

<DOC>
<DOCNO> WSJ870323-0180
<HL> Italy's commerical vehicle sales </HL>
<DD> 03/23/87 </DD>
<DATELINE> TURIN, Italy </DATELINE>
<TEXT>
</TEXT>
</DOC>

<DOC>
<DOCNO> WSJ870323-0161 </DOCNO>
<HL> Who's News: Du Pont Co. </HL>
<DD> 03/23/87 </DD>
<DATELINE> Du Pont Company, Wilmington.
<TEXT>
</TEXT>
</DOC>

The preprocessor accepts these two documents as input and creates the two files that are then loaded into the relational DBMS. The corresponding DOC and INDEX relation given in Tables

INDEX models a pre-created index by storing the occurrences of a term in a document. Without this relation it is not possible to obtain high performance text search within the relational model. Simply storing the entire document in a Binary Large Object removes the storage problem, but most searching operations on BLOB's are limited in that BLOB's typically cannot be indexed.

In a typical information retrieval system a lengthy preprocessing phase occurs in which parsing is done all stored terms are identified. A posting list that indicates for each term, which documents contain that term is identified. A pointer from the term to the posting list is implemented. In this fashion, a hashing function can be used to quickly jump to the term and the pointer can be followed to the posting list. This inverted file technique is so effective that it was used in some of the earliest structured systems in the mid 1960's such as TDBMS.

*Semi-Structured Search Using A Relational Schema :-

Numerous proprietary approaches exist for searching extensible Markup Language (XML) documents, but these lack the ability to integrate with other structured or unstructured data. Relational systems have been used to support XML by building a separate relational schema to map to a particular XML schema or DTD (Document Type Definition). An approach which uses a static relational schema was described in [Florescu and Kossman, 1999] and additional support for a full implementation of an XML query language XML-QL is also described.

Background:

XML has become the standard for platform-independent data exchange. There were a variety of methods proposed for storing XML data and accessing them efficiently. One approach is customized tree file structure, but this lacks portability and does not leverage existing database technology. Other approaches include building a database system specifically tailored to storing semi-structured data from the ground up

or using a full-inverted index.

Static Relational Schema to Support XML-QL:

It was first proposed in [Florescu and Kossmann, 1999] to provide support for XML query processing. Later, in the IIT Information Retrieval Laboratory, it was shown that a full XML-QL query language could be built using this basic structure. This is done by translating semi-structured XML-QL to SQL. The use of a static schema accomodates data of any XML schema without the need for document-type definitions or XSchemas.

The static relational storage schema stores each unique XML path and its value from each document as a separate row in a relation. This is similar to the edge-table described, named for the fact that each row corresponds to an edge in the XML graph representation. This static relational schema is capable of storing an arbitrary XML document.

The hierarchy of XML documents is kept in tact such that any document indexed into the database can be reconstructed using only the information

in the tables. The relations used are:

TAG_Name (TagId, tag)

TAG_PATH (TagId, path)

ATTRIBUTE (AttributeId, attribute)

DOCUMENT (DocId, fileName).

INDEX (Id, parent, path, type, tagId, athId, pos, value)

for the remainder of this section, consider once again our sample text.

<DOC>
<DOCNO> WSJ870323-0180 </DOCNO>
<HL> Italy's Commercial Vehicle Sales </HL>
<DD> 03/23/87 </DD>
<DATELINE> TURIN, Italy </DATELINE>
<TEXT>.
Commercial-vehicle sales in Italy rose 11.4% in February from a year earlier, to 8,848 units, according to provisional figures from the Italian Association of Auto Makers.
</TEXT>
</DOC>

Storing XML Metadata:-

These tables store the metadata (data about the data) of the XML files. TAG_NAME (see Table 6.14) and TAG_PATH (see Table 6.15) together stores the name

of each unique tag in the XML collections TAG_PATH stores the unique paths found in the XML documents. The ATTRIBUTE (See Table 6-16) relation stores the names of all the attributes. In our example we have added an attribute called LANGUAGE which is an attribute of the tag TEXT. In the TAG_NAME and TAG_PATH relations, the tagId is a unique key assigned by the pre-processing stage. Similarly, attribute Id is uniquely assigned as well. As with our examples earlier in the chapter, these tables are populated each time a new XML file is indexed. This process consists of passing the XML file and extracting all of this information and storing it into these tables.

Tracking XML Documents:

Since XML-QL allows users to specify what file they wish to query, many times we do not want to look at each record in the database but only at a subset of records that correspond to that file. Each time a new file is indexed, it receives a unique identifier that is

known as the pin value. This value corresponds to a single XML file. The DOCUMENT relation contains a tuple for each XML document. For our example, we only store the actual file name that contains this document. An example of this relation is shown in Table 6.17

tagId	tag
10	DOC
11	DOCNO
12	HL
13	DD
14	DATELINE
15	TEXT

Table 6.14 TAG_NAME

tagId	path
10	[DOC]
11	[DOC, DOCNO]
12	[DOC, HL]
13	[DOC, DD]
14	[DOC, DATELINE]
15	[DOC, Text]

Table 6.15. TAG_PATH

AttributeId	attribute
7	LANGUAGE

table 6.16. ATTRIBUTE

docID	fileName
2	doc-0.xml
3	doc-1.xml

Table 6.17. DOCUMENT.

Index :-

The INDEX table (see table 6.18) models an XML index. It contains the mapping of each tag, attribute or value to each document that contains this value. Also, since the order of attributes and tags is important in XML, the position order of the tags is also stored. The tagId and AttrId is a foreign key to the TagId in the TagName and Attribute tables.

id	parent	path	type	tagId	AttrId	docId	pos	value
41	0	10	E	10	1	6	0	NULL
42	41	11	E	11	1	6	0	1055187032 3-0180
43	41	12	E	12	1	6	0	Italy's Commercial
44	41	13	E	13	1	6	0	03/23/87
45	41	14	E	14	1	6	0	TURIN, Italy

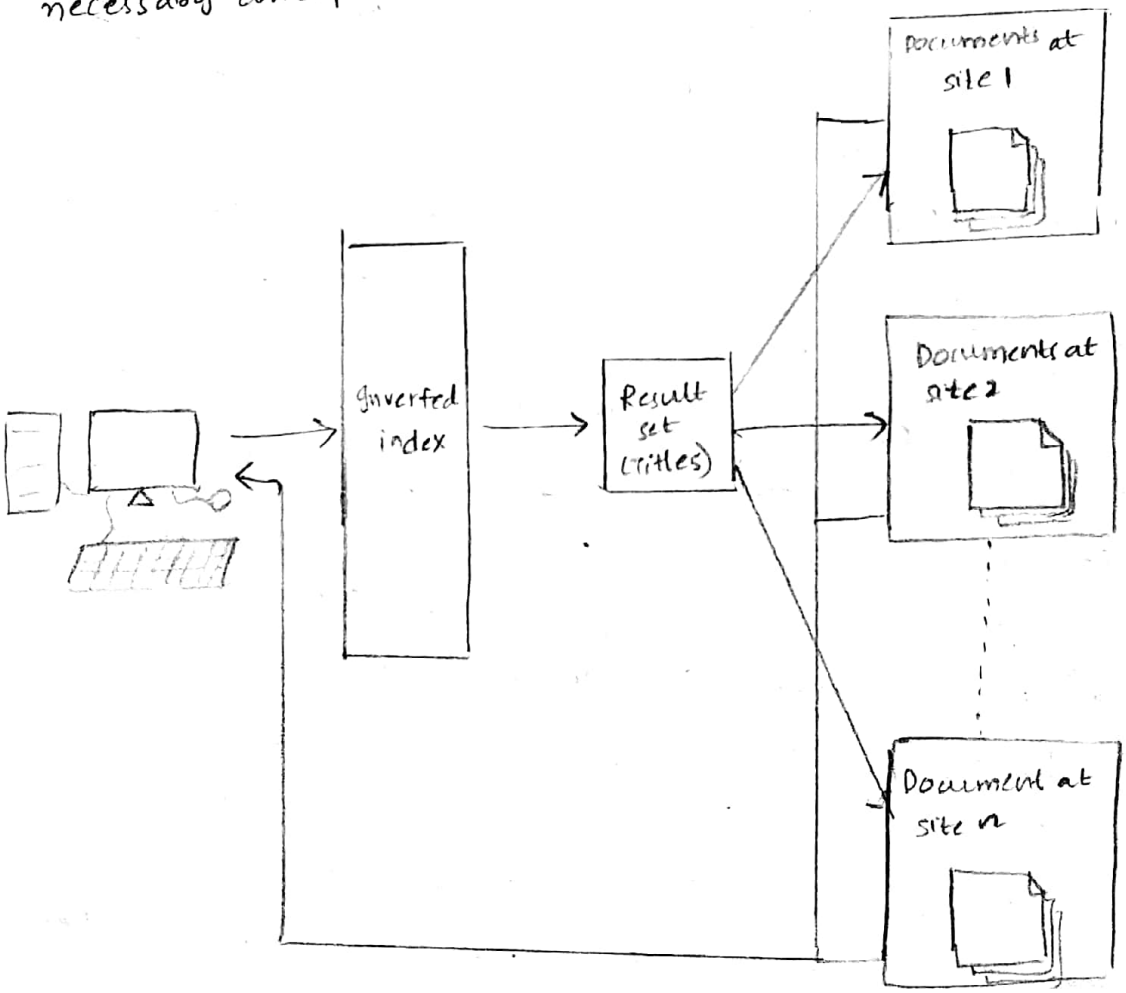
Table 6.18. INDEX

Unit -5

A Theoretical Model of Distributed Retrieval

Distributed Information Retrieval.

The document collection is spread acoross various geographical locations. The potential to access and process these collection of data is more necessary, instead of efficiently handlias them many constraints are enforced at the time of centralizing the data - such as data security, sheer volume which inhibits the physical transfer, rate of change, legal and political constraints. In a way similar to this, there exist systems called Distributed Information Retrieval systems (DIRS) which allows usess to obtain access to data located at different geographical areas on various system. Hence, making it a necessary concept in today's corporate world.



Distributed Document Retrieval

The theoretical model of distributed retrieval includes the followings.

1. Centralized information retrieval system model
2. Distributed information retrieval system model.

Centralized Information Retrieval System Model

The information retrieval system can be specified as triple.

$$I = (D, R, \delta_j)$$

where D = Document Collection

R = set of Queries

$\delta_j = R_j \rightarrow 2^{D_j}$. It represents the mapping which allots jth query to the group of documents.

Most of the IRS are thesaurus dependent, where in the user can expand the query inorder to add synonyms corresponding to the keyword. This is done inorder to find the matching synonyms with in the document.

The proposed expression of the IRS, i.e., the triple can be extended to a quadruple inorder to accomodate thesaurus

$$I = (T, D, R, \delta)$$

where T is set of distinct terms.

Consequently, the relation $p \subseteq T \times T$ such that $p(t_1, t_2)$. Here t_1 is considered as synonym of t_2. The synonym relation is applied to the documents as a set of descriptors and ascriptors. For instance D_i is considered and descriptors 'd' which holds all term belonging D_i. It should state that

1. The descriptor should not be synonym of another descriptor.
2. Each and every descriptor should be unique.

On the other hand, ascriptor can be specified as a term which behaves a synonym of a descriptor. Only one ascriptor can be synonymous to only one descriptor. Therefore, descriptors produce least description of document. Moreover generalization relation above the sets of descriptors is stated as, $\gamma \subseteq d \times d$.

where, $r(t_1, t_2)$ = It is a more general term than t_2.

Example

r[animal, cat) is an example of generalization.

with generalization relation, a hierarchical knowledge base belonging to the set of descriptors can be created, which can be done manually as well as automatically. However, the terms could not be defined easily, such as the relationship w.r.t spatial and temporal substances.

The terms which doesnot form any synonyms with other terms are benifited in thesaural as, it simplifies the document description. Even though new synonyms are included, it doesnot increase the semantic value of the document. The query processing is carried out by the generalization relating stating "list all animals", the document which is resulted carry the information with to cat, mouse, dog, etc. The term dog and mouse are not included in the document are also returned.

The partial ordering of documents can be specified using generalization, $\leq$ representing the partial ordering and $t(d_i)$ represents list of descriptors corresponding to the document d. The partial ordering $\leq$ is stated as,

$$t(d_1) \leq t(d_2) \iff (\forall t \in t(d_1))((\exists t \in t(d_2))(r(t, t'')).$$

The above ordering $d_1 \leq d_2$ corresponds to the document d_1 where descriptors are specified as the generalization of descriptor present in d_2.

Distributed Information Retrieval system Model

The centralized information retrieval system can be divided into 'n' local information retrieval system $S_1, S_2, S_3, \ldots S_n$. Here every single system takes the form such that

$$S_j = (T_j, D_j, R_j, \delta_j).$$

where in,

T_j = Thesaurus.

D_j = Document collection

R_j = Set of Queries

$\delta_j = R_j \rightarrow 2^{D_j}$, it performs the mapping of queries to the documents.

The distributed information retrieval systems can be specified upon performing the union of local sites.

$$S = (T, D, E, S)$$

where in, $T = \bigcup_{j=1}^{n} T_j$

$$S_j = S \cap (T_j \times T_j),$$

$$R_j = R \cap (d_j \times d_j)$$

This above equation signifies that the reconstruction of the global thesaurus is possible from the local thesauri subsequently, the queries exists at site 'j' adds only the descriptors present at site 'j' this is necessary because it avoids the retrieval of any documents for the terms corresponding to the query which are not descriptors.

$$D = \bigcup_{j=1}^{n} D_j$$

Integration of document collection on each site will retrieve the document collection D

$$R \supset \bigcup_{j=1}^{n} R_j$$

$$\leq \bigcap_{j=1}^{n} (R_j \times R_j)$$

In a way similar to this, queries retrieval can be done by integrating the queries existing at each local site. Also, the partial order specified at site 'j' corresponds to the queries present at site 'j'

$$(\forall r \in R)(\delta(r) = d : d \in D \wedge r \leq t(d))$$

The document collection for each query within the system have documents stored in the collection. Here, the documents are as specific as query.

The hierarchical representation given by 'r' is segregated among various sites. Consequently, the query sent to the originating site will also be received by local site as well where in the local query is computed. The obtained responses are sent back to the originating sites where they are integrated to obtain final result set, therefore the local sites holds good, the criteria of subsystem of IRS

$S_1 = (T_1, D_1, R_1, \delta_1)$ is a system of $S_2 = (T_2, D_2, R_2, \delta_2)$.

if

$(T_1 \supset T_2) \wedge (R_1 = R_2) \cap (d_1 \times d_2) \wedge (\delta_1 = \delta_2) \cap (T_1 \times T_2)$

It is important to note that thesaurus of T_1 is a superset of T_2.

$$D_1 \supset D_2$$

Document collection present at site S_1 holds collections D_2

$$R_1 \in R_2 \wedge \leq_1 = \leq_2 \cap (R_1 \times R_2)$$

Also the queries present at site S_1 Holds the queries present at S_2

$$\delta_1(r) = \delta_2(r) = \cap D_1 \text{ for } r \in R$$

It shows that document collection retrieved by queries present in S_1, will add all the documents returned by queries present in S_2.

Web Search

Evaluation of web search

The evaluation of individual system in traditional retrieval environment is performed using standard queries and data. But the evaluation of these conditions are not supported by web environment. Additionally due to the large size and dynamic nature of web, manual evaluation also doesnot holds good. However, these problems can be addressed using automatic evaluation of the web search engine. It introduces a method called online taxonomies which serves as a part of Open Directory Project (ODP) ~~un~~respectively. The online directories serves as a relevant items for a query. And the occasence of the match is made only when query matches the title of stored item or directory file name holding known items.

High precision search

Web search engines can be evaluated using dissimilar measures in contrast to traditional measures ideally, precision & recall measures are both considered as important evaluation metrics, the estimation of recall is considered as laborious task where as calculating the precision is not much important because many user doesnot access those links which are beyond the first screen. Hence, web search engine developers ensure that first results screen contains highly accurate matches and is generated quickly.

The textual extraction is efficiently performed through template web documents. But the remainder of the frame or frames is discarded inorder to stop the recognition of the document. Hence, high precision measures shift and discard the irrelevant items.

Page Rank:

It is most effective and popular algorithm to improve web search. The algorithm was first proposed in Brin and page 200 which improvises the concept of hubs and authorities in webgraph. Ideally, the popular search engine Google operates on page Rank algorithm. The working of algorithm initiates by employing the incoming and outgoing links inorder to assimilate the webpage score. It is computed depending on popularity of webpage not with the users query. In comparison to page Rank algorithm, the traditional retrieval strategy produces two documents of same rank, but the page rank enhances the similarity measure w.r.t the popular document, where in document is said to be popular if it has many webpage links to the document.

The computation of the pageRank corresponding to pageA, covering all pages that connects to it $P_1, P_2, \ldots P_n$ is specified as,

$$PageRank(A) = (1-d) + d \sum_{P_1, P_2, \ldots P_n} \frac{PageRank(D_i)}{C(D_i)}$$

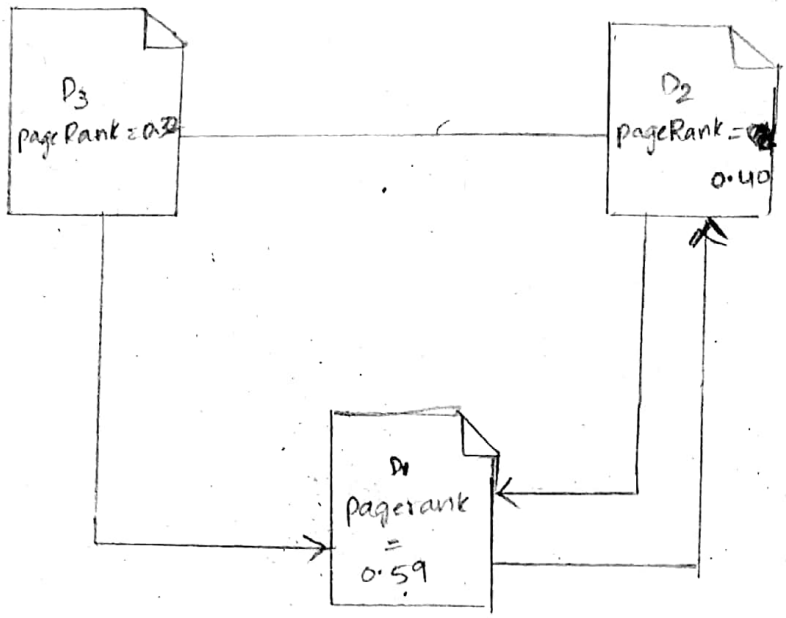
where in,

$C(D_i)$ = Number of pages which connects from page D_j

d = Dampening factor ranging from 0 1 the role of the dampening factor is to assign non-zero Page Ranking to the page which doesnot have links to them.

Example

With the common dampening factor of 0.85 and setting the beginning of each page Rank to 1.0, the number of iterations performed before converging the scores were 8.



PageRank Calculation.