RETRIEVAL STRATEGIES

Vector Space Model:

Both the query and each document are represented as vector in the term space. A measure of the similarity between the two vectors is computed.

~~The Vector Spac~~

If one can represent the words in the document by a vector, it is possible to compare documents with queries to determine how similarity their content is. If a query is considered to be like a document, a similarity coefficient (SC) that measures the similarity between a document and a query can be computed.

This more formal definition, and slightly larger example, illustrates the use of weights based on the collection frequency. Weight is computed using the Inverse Document Frequency (IDF) corresponding to a given term.

To construct a vector that corresponds to each document, consider the following definitions

t = number of distinct terms in the document collection

tf_{ij} = number of occurrences of term t_j in document D_i.
This is referred to as the term frequency (TF)

df_j = number of documents which contain t_j.
This is the document frequency (df)

$idf_j = \log\left(\frac{d}{df_j}\right)$ where d is the total number of documents
This is the inverse document frequency (idf)

The weighting factor for a term in a document is defined as a combination of term frequency, and inverse document frequency. That is, to compute the value of the i^{th} entry in the vector corresponding to document i, the following equation is used.

$$d_{ij} = tf_{ij} \times idf_j$$

Consider a document collection that contain a document D_1, with occurrences of the term green and a document D_2, with only five occurrences of term green.

When a document retrieval system is used to query a collection of documents with t distinct collection-wide terms, the system computes a vector $D(d_{i1}, d_{i2}, \ldots d_{it})$ of size t for each document. The vector are filled with term weights as described.
Similarly a vector $Q(w_{q1}, w_{q2}, \ldots w_{qt})$ is constructed for the terms found in the query.

A simple similarity coefficient (SC) between query Q and a document D_i is defined by the dot product of two vectors.

$$SC(Q, D_i) = \sum_{j=1}^{t} w_{qj} \times d_{ij}$$

Example of Vector Space model:-

Consider a case insensitive query and document collection with a query Q and a document collection consisting of the following three documents.

Q: "gold silver truck"

D_1: "Shipment of gold damaged in a fire"

D_2: "Delivery of silver arrived in a silver truck"

D_3: "Shipment of gold arrived in a truck"

In this collection, there are three documents, so $d=3$. If a term appears in only one of three documents, its idf is $\log \frac{d}{df_i} = \log \frac{3}{1} = 0.477$

The idf for the terms in the three documents is given below:

$idf_a = \log \frac{3}{3} = \log 1 = 0$

$idf_{arrived} = \log \frac{3}{2} = 0.176$

$idf_{damaged} = \log \frac{3}{1} = 0.477$

$idf_{delivery} = \log \frac{3}{1} = 0.477$

$idf_{fire} = \log \frac{3}{1} = 0.477$

$idf_{in} = \log \frac{3}{3} = \log 1 = 0$

$idf_{of} = \log \frac{3}{3} = 0$

$idf_{silver} = \log \frac{3}{1} = 0.477$

$idf_{shipment} = \log \frac{3}{2} = 0.176$

$idf_{truck} = \log \frac{3}{2} = 0.176$

$idf_{gold} = \log \frac{3}{2} = 0.176.$

Document vectors can now be constructed. Since eleven terms appear in the document collection, an eleven-dimensional document vector is constructed. The alphabetical ordering given above is used to construct the document vector

So that t_1 corresponds to term number one which is "a" and t_2 is arrived, etc.

The weight for term i in vector j is computed as the $idf_i \times tf_{ij}$.

Table: Term frequency

	a	arrived	damaged	delivery	fire	gold	in	of	shipment	silver	truck
D_1	1	0	1	0	1	1	1	1	1	0	0
D_2	1	1	0	1	0	0	1	1	0	2	1
D_3	1	1	0	0	0	1	1	1	1	0	1

Table: Document-Vector ($idf_i \times tf_{ij}$)

docid	a	arrived	damaged	delivery	fire	gold	in	of	shipment	silver	truck
D_1	0	0	.477	0	.477	.176	0	0	.176	0	0
D_2	0	.176	0	.477	0	0	0	0	0	.954	.176
D_3	0	.176	0	0	0	.176	0	0	.176	0	.176
Q	0	0	0	0	0	.176	0	0	0	.477	.176

$$SC(Q, D_1) = D_1 \cdot Q = (0)(0) + (0)(0) + (.477)(0) + (0)(0) + (.477)(0) + (.176)(.176) + (0)(0) + (0)(0) + (.176)(0) + (0)(.477) + (0)(.176) = 0.031$$

Similarly

$$SC(Q, D_2) = (0.954)(0.477) + (0.176)^2 = 0.486$$

$$SC(Q, D_3) = (0.176)^2 + (0.176)^2 \doteq 0.062$$

The ranking would be D_2, D_3, D_1.

Probabilistic Retrieval Strategies:

The probabilistic model computes the similarity coefficient (SC) between a query and a document as the probability that the document will be relevant to the query. Probability theory can be used to compute a measure of relevance between a query and a document. Two fundamentally different approaches were proposed. The first relies on usage pattern to predict relevance. the second uses each term in the query as clues as to whether or not a document is relevant.

Simple Term Weights:-

The use of term weights is based on the Probability Ranking Principle (PRP) which assume that optimal effectiveness occurs when document are ranked based on an estimate of the probability of their relevance to a query. The key is to assign probabilities to components of the query and then use each of these as evidence in computing the final probability that a document is relevant to the query.

The terms in the query are assigned weights which correspond to the probability that particular term, in a match with a given query. will retrieve a relevant document. The weights for each term in the query are combined to obtain a final measure of relevance.

For an information retrieval query, the terms in the query can be viewed as indicators that a given document is relevant. The presence or absence of query term A

can be used to predict whether or not a document is relevant. Hence, after a period of observation, it is found that when term "A" is in both the query and the document, there is an "x" percent chance the document is relevant. We then assign a probability to term 'A'. Assuming independence of terms, this can be done for each of the terms in the query. Ultimately, the product of all the weights can be used compute the probability of relevance.

Most work in the probability model assume independence of terms because handling dependencies involves substantial computation. It is unclear whether or not effectiveness is improved when dependencies are considered. We note that relatively little work has been done implementing these approaches. It is necessary to obtain sufficient training data about term co-occurrance in both relevant and non relevant documents.

Consider a document, d_i, consisting of t terms $(w_1, w_2, \ldots w_t)$, where w_i is the estimate that term i will result in this document being relevant. The weights or 'odds' that document d_i is relevant is based on the probability of relevance for each term in the document. For a given term in a document, its contribution to the estimate of relevance for the entire document is computed as:

$$\frac{P(w_i|rel)}{P(w_i|nonrel)}$$

Now that we have described how the individual term estimate can be combined into a total estimate of relevance for the document, it is necessary to describe a means of estimating the individual term weights. In 1976 paper, Robertson and Sparck Jones considered several methods They began by presenting two mutually exclusive Independence assumptions:

I1: The distribution of terms in relevant document is independent and their distribution in all documents is independent.

I2: The distribution of terms in relevant documents is independent and their distribution in non-relevant document is independent.

They also presented two methods, referred to as Ordering Principles, for presenting the result set:

O1: Probable relevance is based ~~on~~ only on the presence of Search terms in the documents.

O2: Probable relevance is based on both the presence of Search terms in the documents and their absence from documents.

I1: indicates that terms occur randomly within a document - that is the presence of one term in a document in no way impacts the presence of another term in the same document.

I2: indicates that terms in relevant documents are independent i.e, they satisfy I1 and terms in non-relevant documents also satisfy I1.

O1 indicates that documents should be highly ranked only if they contain matching terms in the query (i.e., the only evidence used is which query terms are actually presented in the document).

O2 takes O1 a little further and says that we should consider both the presence and absence of search terms in the query. Hence, for a query that asks for term t_1 and term t_2 - a document with just one of these terms should be ranked lower than a document with both terms.

four weights are then derived based on different combination of these Ordering Principles and Independence assumptions. Given a term t, consider the following.

N — Number of documents in the collection.

R — number of relevant documents for a given query "q"

n — number of documents that contain term "t"

r — number of relevant documents that contain term "t"

choosing I1 and O1 yields the following weight $w_1 = \log\left[\frac{r/R}{n/N}\right]$

choosing I2 and O1 yields the following weight $w_2 = \log\left[\frac{r/R}{n-r/N-R}\right]$

choosing I_1 and O2 yields the following weight: $w_3 = \log\left[\frac{r/R-r}{n/N-n}\right]$

choosing I_2 and O2 yields the following weight: $w_4 = \log\left[\frac{r/R-r}{n-r/(N-n)-(R-r)}\right]$

Robertson and Spark Jones is argue w_4 is most likely to yield the best result, That then present results that indicate that w_4 and w_3 performed better than w_1 and w_2.

When Incomplete relevance information is available 0.5 is added to the weights. Robertson and Sparck Jones suggest that, "This procedure may seem somewhat arbitary, but it does in fact have some statistical justification". The modified weighting function appears as:

$$w = \log\left[\frac{\frac{r+0.5}{(R-r)+0.5}}{\frac{(n-r)+0.5}{(N-n)-(R-r)+0.5}}\right]$$

Example:

The documents and the query are:

Q : "gold silver truck."

D1 : "Shipment of gold damaged in a fire"

D2 : "Delivery of silver arrived in a silver truck."

D3 : "Shipment of gold arrived in a truck".

Since training data are needed for the probalitistic model, we assume that these three documents are the training data we deem documents D2 and D3 as relevant to the query.

To Compute the Similarity Co-efficient, we assign term weights to each term in the query.

.	Gold	Silver	Truck
N	3	3	3
n	2	1	2
R	2	2	2
r	1	1	2

Note that with our collection, the weight for silver is infinite, since $(n-r)=0$. This is because 'silver' only appears in relevant documents. Since we are using this procedure in predictive manner, Robertson and Sparck Toney recommended adding constant to each quantity
The new weights are:

$$w_1 = \log\left[\frac{\frac{r+0.5}{R+1}}{\frac{n+1}{N+2}}\right], \quad w_2 = \log\left[\frac{\frac{(r+0.5)}{(R+1)}}{\frac{(n-r+0.5)}{(N-R+1)}}\right]$$

$$w_3 = \log\left[\frac{\frac{(r+0.5)}{(R-r+0.5)}}{\frac{(n+1)}{(N-n+1)}}\right], \quad w_4 = \log\left[\frac{\frac{(r+0.5)}{(R-r+0.5)}}{\frac{(n-r+0.5)}{(N-n-(R-r)+0.5)}}\right]$$

Using these equations, we derive the following weights.

gold $$w_1 = \log\left[\frac{\frac{(1+0.5)}{(2+1)}}{\frac{(2+1)}{(2+2)}}\right] = \log\frac{0.5}{0.6} = -0.079$$

silver $$w_1 = \log\left[\frac{\frac{(1+0.5)}{(2+1)}}{\frac{(1+1)}{(3+2)}}\right] = \log\left[\frac{0.5}{0.4}\right] = 0.097$$

truck $$w_1 = \log\left[\frac{\frac{(2+0.5)}{(2+1)}}{\frac{(2+1)}{(3+2)}}\right] = \log\frac{0.833}{0.6} = 0.143$$

gold $$w_2 = \log\left[\frac{\frac{(1+0.5)}{(2+1)}}{\frac{(2-1+0.5)}{(3-2+1)}}\right] = \log\frac{0.5}{0.75} = -0.176$$

silver $$w_2 = \log\left[\frac{\frac{(1+0.5)}{(2+1)}}{\frac{(1-1+0.5)}{(3-2+1)}}\right] = \log\frac{0.5}{0.25} = 0.301$$

truck $W_2 = \log\left[\frac{\frac{(2+0.5)}{(2+1)}}{\frac{(2-2+0.5)}{(3-2+1)}}\right] = \log\frac{0.833}{0.25} = 0.523$

gold $W_3 = \log\left[\frac{\frac{(1+0.5)}{(2-1+0.5)}}{\frac{(2+1)}{(3-2+1)}}\right] = \log\frac{1.0}{1.5} = -0.176$

Silver $W_3 = \log\left[\frac{\frac{(1+0.5)}{(2-1+0.5)}}{\frac{(1+1)}{(3-1+1)}}\right] = \log\frac{1.0}{0.667} = 0.176$

truck $W_3 = \log\left[\frac{\frac{(2+0.5)}{(2-2+0.5)}}{\frac{(2+1)}{(3-2+1)}}\right] = \log\frac{5}{1.5} = 0.523$

gold $W_4 = \log\left[\frac{\frac{(1+0.5)}{(2-1+0.5)}}{\frac{(2-1+0.5)}{(3-2-2+1+0.5)}}\right] = \log\frac{1}{0.333}$ ~~$= 0.477$~~ -0.477

silver ~~W_4~~

$W_4 = \log\left[\frac{\frac{(1+0.5)}{(2-1+0.5)}}{\frac{(1-1+0.5)}{3-1+2+1+0.5}}\right] = \log\frac{1}{0.333} = 0.477$

truck ~~W_4~~

$W_4 = \log\left[\frac{\frac{(2+0.5)}{(2-2+0.5)}}{\frac{(2-2+0.5)}{(3-2-2+2+0.5}}\right] = \log\left[\frac{5}{0.333}\right] = 1.176$

Term Weights

	W_1	W_2	W_3	W_4
gold	-0.079	-0.176	-0.176	-0.477
Silver	0.097	0.301	0.176	0.477
truck	0.143	0.523	0.523	1.176

	W_1	W_2	W_3	W_4
D1	-0.079	-0.176	-0.176	-0.477
D2	0.240	0.824	0.699	1.653
D3	0.064	0.347	0.347	0.699

Document Weights

The Similarity coefficient for a given document is obtained by summing the weights of the terms present.

for the document weights

⇒ For D1, gold is the only term to appear so the weight for D1 is just the weight for gold, which is -0.079.

⇒ For D2, silver and truck appears so the weight for D2 is the sum of the weights for silver and truck, which is 0.097 + 0.143 = 0.240.

⇒ For D3, gold and truck appears so the weight for D3 is the sum for gold and truck, which is -0.079 + 0.143 = 0.064.

Non-Binary Independence Model:-

The non-binary independence model developed by Yu, Meng, and Park incorporates term frequency and document length, somewhat naturally, into the calculation of term weights. Once the term weights are computed, the vector space model is used to compute an inner product for obtaining a final similarity coefficient.

The simple term weight approach estimates a term's weight based on whether or not the term appears in a relevant document. Instead of estimating the probability that a given term will identify a relevant document, the probability that a term which appears tf times will appear in a relevant document is estimated.

With non-binary independence model, we calculate a separate probability for each term frequency. Hence, we compute the probability that blue will occur one time $P(1/R) = 0.1$ because it did occur one time in document one.

The probability that blue will occur ten times is $P(10/R) = 0.1$ because it did occur ten times in one out of ten documents. Compute the probability that blue occurs only once in a relevant document to the probability that blue occurs 0.5 times in a relevant document.

The probability that a term will result in a non-relevant document is also used. The final weight is computed as the ratio of the probability that a term will occur tf times in relevant documents to the probability that the term will occur tf times in non-relevant documents.

More formally $\log \frac{P(di/R)}{P(di/N)}$

Where $P(di/R)$ is the probability that a relevant document will contain di occurrences of i^{th} term and $P(di/N)$ is the probability that a non-relevant document has di occurrences of the i^{th} term.

Example:-

Q: "gold Silver truck"

D1: "Shipment of gold damaged in a fire"

D2: "Delivery of Silver arrived in a Silver truck"

D3: "Shipment of gold arrived in a truck.

Table: Term to Document Mapping

docid	a	arrived	damaged	delivery	fire	gold	in	of	shipment	Silver	truck
D1	1	0	1	0	1	1	1	1	1	0	0
D2	1	1	0	1	0	0	1	1	0	2	1
D3	1	1	0	0	0	1	1	1	1	0	1
Q	0	0	0	0	0	1	0	0	0	1	1

Thus three documents with eleven terms and a single query The training data include both relevant and non-relevant documents. we assume that documents two and three are relevant and document one is not relevant. Normalizing by documents length yields as shown below table.

docid	a	arrived	damaged	delivery	fire	gold	in	of	shipment	Silver	truck
D1	1/7	0	1/7	0	1/7	1/7	1/7	1/7	1/7	0	0
D2	1/8	1/8	0	1/8	0	0	1/8	1/8	0	1/4	1/8
D3	1/7	1/7	0	0	0	1/7	1/7	1/7	1/7	0	1/7

Here do not normalize the query. The terms present in the query are gold, Silver. and truck For D1 the weight of gold is

$$\log\left[\frac{P(1/4/R)}{P(1/7/N)}\right] = \log\frac{1/2}{1} = -0.3010$$

Of the two relevant documents, one has a frequency of 1/7 and one does not, so $P(\frac{1}{7}|R) = 1/2$. However, the only one relevant document has gold with a frequency of 1/7 so $P(\frac{1}{7}/N) = 1$.

For silver in D1 we obtain:

$$\log\left[\frac{P(0|R)}{P(0/N)}\right] = \log\frac{1}{2} = -0.3010$$

Language Model:-

A statistical language model is a probabilistic mechanism for "generating" a piece of text. It thus a distribution over all the possible word sequences. The simplest language model is the unigram language model. which is essential a word distribution

The language models for speech recognition and language translation, their use ~~of~~ for information retrieval started only in 1998 [Ponte and Croft]. The core idea is that documents can be ranked on their likehood of generating the query.

The similar coefficient is simply:

$$Sc.(Q, Di) = P(Q/M_{Di})$$

where M_{Di} is the language model implicit in document Di.

The probability of the query is calculated as the product of probabilities for both the terms in the query and terms absent. that is

$$S.C(Q, Di) = \prod_{tj \in Q} P(tj/M_{Di}) \prod_{tj \notin Q} (1 - P(tj/M_{Di}))$$

$$P(tj/M_{Di}) = P_{ml}(tj/M_{Di})$$

where $P_{ml}(tj/M_{Di})$ is the maximum likehood estimate of the term distribution (i.e the relative term frequency)

$$P_{ml}(tj/M_{Di}) = \frac{tf(tj, Di)}{dl_{Di}}$$

dl_{Di} - is the document length of document Di.

Smoothing:

To avoid the problem caused by terms in the query that are not present in a document, various smoothing approaches exist which estimate non-zero values for these terms. One approach assumes that the query term could occur in this model, but simply at no higher a rate than the chance of it occuring in any other document.

The ratio $\boxed{\frac{cf_t}{cs}}$ was initially proposed where

cf_t is the number of occurrences of term t in the collection

cs is the number of terms in the entire collection.

An additional adjustment is made to account for the reality that these document models are based solely on individual documents. These are relatively small sizes from which to build a model. To use a larger sample (the entire collection) the following estimate is proposed.

$$\boxed{P_{avg}(t) = \frac{\sum_{d(t \in d)} P_{ml}(t/M_d)}{df_t}}$$

To improve the effectiveness of the estimates for term weights it is possible to minimize the risk involved in our estimate. First define $\boxed{\bar{f}_t}$ as the mean term frequency of term t in the document.

$$\boxed{\bar{f}_t = P_{avg}(t) \times dl_d}$$

The risk can be obtained using a geometric distribution as:

$$\boxed{R_{t,d} = \left(\frac{1.0}{1.0 + \bar{f}_t}\right) \times \left(\frac{\bar{f}_t}{1.0 + \bar{f}_t}\right)^{tf_{t,d}}}$$

$$\boxed{P(t,d) = \begin{cases} P_{ml}(t,d)^{(1-R(t,d))} \times P_{avg}(t)^{R(t,d)} & \text{if } tf(t,d) > 0 \\ \alpha_d \times P_{avg}(t) & \text{otherwise} \end{cases}}$$

Language Model Example:

Q: "gold Silver truck"

D1: "Shipment of gold damaged in a fire"

D2: "Delivery of Silver arrived in a Silver truck"

D3: "Shipment of gold arrived in a truck"

There are three documents in a test collection. There are 22 tokens so $CS = 22$. The total number of tokens in documents D1, D2 and D3 are 7, 8 and 7. respectively The below table contains values for dl_d (document length

Term occurrence

	D1	D2	D3
dl_d	7	8	7

df_t, the document frequency of t,

	a	arrived	damaged	delivery	fire	gold	in	of	shipment	silver	truck
df_t	3	2	1	1	1	2	3	3	2	1	2

cf_t, the raw count of token t in the collection is given in below table.

	a	arrived	damaged	delivery	fire	gold	in	of	shipment	silver	truck
cf_t	3	2	1	1	1	2	3	3	2	2	2

$tf_{t,d}$ the raw term frequency of term t in document d

First need to calculate $P_{ml}(t/M_d)$ the maximum likehood estimate of probability of term t under the distribution for document d.

For each term t, $P_{ml}(t/M_d) = \frac{tf_{td}}{dl_d}$

Table: Raw term Frequency

	a	arrived	damaged	delivery	fire	gold	in	of	shipment	Silver	truck
D_1	1	0	1	0	1	1	1	1	1	0	0
D_2	1	1	0	1	0	0	1	1	0	2	1
D_3	1	1	0	0	0	1	1	1	1	0	1

Table: Maximum likehood for each term

$P_{ml}(t/M_d)$	D_1	D_2	D_3
a	0.143	0.125	0.143
arrived	0	0.125	0.143
damaged	0.143	0	0
delivery	0	0.125	0
fire	0.143	0	0
gold	0.143	0	0.143
in	0.143	0.125	0.143
of	0.143	0.125	0.143
shipment	0.143	0	0.143
Silver	0	0.250	0
truck	0	0.125	0.143

$$P_{ml}(a/D_1) = \frac{tf_{a,D_1}}{dl_{D_1}} = \frac{1}{7} = 0.143$$

Second calculate the mean probability of term t in documents which contain the term The equation is:

$$P_{avg}(t) = \frac{\sum_{d(t \in d)} P_{me}(t/M_d)}{df_t}$$

eg:

$$P_{avg}(arrived) = \frac{P_{me}(arrived/MD_2) + P_{me}(arrived/MD_3)}{df_{arrived}}$$

$$= \frac{0.125 + 0.143}{2} = 0.134$$

$$P_{avg}(a) = \frac{P_{me}(a/MD_1) + P_{me}(a/MD_2) + P_{me}(a/MD_3)}{df_a}$$

$$= \frac{0.143 + 0.125 + 0.143}{3} = 0.137.$$

Table: Average Probability for each term.

	a	arrived	damaged	delivery	fire	gold	in	of	shipment	silver	truck
$P_{avg}(t)$	0.137	0.134	0.143	0.125	0.143	0.143	0.137	0.137	0.143	0.250	0.134

Third: calculate the risk for a term t in a document d. To do that $\bar{f}_t$, the mean term frequency of term t in a document is computed by the following equation

$$\bar{f}_t = P_{avg}(t) \times dl_d.$$

$\bar{f}_t$ of each term is given following table.

$\bar{f}_a = 0.13$ ~~$P_{avg}(a) \times dl_{D_1} = 0.137 \times 7 = 0.958$~~

$\bar{f}_a = P_{avg}(a) \times dl_{D_2} = 0.137 \times 8 = 1.096$

$\bar{f}_a = P_{avg}(a) \times dl_{D_3} = 0.137 \times 7 = 0.958$

Table : Mean Term Frequency for each term:

f_t	a	arrived	damaged	delivery	fire	gold	in	of	shipment	silver	truck
D_1	0.958	0.938	1.000	0.875	1.000	1.000	0.958	0.958	1.000	1.750	0.938
D_2	1.096	1.071	1.143	1.000	1.143	1.143	1.096	1.096	1.143	2.000	1.071
D_3	0.958	0.938	1.000	0.875	1.000	1.000	0.958	0.958	1.000	1.750	0.938

The risk values per term per document as shown in table

$$R_{t,d} = \left(\frac{1.0}{1.0+f_t}\right) \times \left(\frac{f_t}{1.0+f_t}\right)^{tf_{t,d}}$$

$$R_{a,D_1} = \left(\frac{1.0}{1.0+0.958}\right) \times \left(\frac{0.958}{1.0+0.958}\right)^{1} = \underline{0.250}$$

$$R_{fire,D_1} = \left(\frac{1.0}{1.0+f_{fire}}\right) \times \left(\frac{f_{fire}}{1.0+f_{fire}}\right)^{tf_{fire,D_1}}$$

$$= \left(\frac{1}{1+1}\right) \times \left(\frac{1}{1+1}\right)^{1} = \underline{0.250}$$

Table: Risk for each term

$R_{t,d}$	D_1	D_2	D_3
a	0.250	0.249	0.250
arrived	0.516	0.250	0.250
damaged	0.250	0.467	0.500
delivery	0.533	0.250	0.533
fire	0.250	0.467	0.500
gold	0.250	0.467	0.250
in	0.250	0.249	0.250
of	0.250	0.249	0.250
shipment	0.250	0.467	0.250
silver	0.364	0.148	0.364
truck	0.516	0.249	0.250

Now use the risk value as a mixing parameter to calculate $P(Q/M_d)$, the probability of producing the query for a given document model as mentioned before.

$$P(t/M_{di}) = \begin{cases} P_{mle}(t,d)^{(1-R(t,d))} \times P_{avg}(t)^{R(t,d)} & \text{if } tf(t,d) > 0 \\ \frac{cf_t}{cs} & \text{otherwise} \end{cases}$$

See table Raw term frequency if term frequency is > 0 then use above equation in case Term frequency (tf) less than zero use the equation $P(t/M_d) = \frac{cf_t}{cs}$

for document D_1 term consist $tf > 0$ is a, damaged, fire, gold, in, of shipment use above equation remain term use $\frac{cf_t}{cs}$

$$P(a/M_{D_1}) = P_{mle}(a/M_{D_1})^{(1.0-R_{AD_1})} \times P_{avg}(a)^{R_{AD_1}}$$

$$= (0.143)^{(1-0.250)} \times (0.137)^{0.250} = \underline{0.141}$$

$$P(a/M_{D_2}) = P_{mle}(a/M_{D_2})^{(1.0-R_{AD_2})} \times P_{avg}(a)^{R_{AD_2}}$$

$$= (0.125)^{1-0.249} \times (0.137)^{0.249} = \underline{0.128}$$

$$P(\text{delivery}/M_{D_1}) = \frac{cf_{\text{delivery}}}{cs} = \frac{1}{22} = 0.045$$

$\because tf_{\text{deliver D}} = 0$ so use $P(t/M_d) = \frac{cf_t}{cs}) =$

like that all term calculate $P(t/M_d)$

final measure of similarity $P(Q/M_d)$.

The actual values are given below table.

Table: Expected Probability for each term

P(t/Md)	D1	D2	D3
a	0.141	0.128	0.141
arrived	0.091	0.127	0.141
damaged	0.143	0.045	0.045
delivery	0.045	0.125	0.045
fire	0.143	0.045	0.045
gold	0.143	0.091	0.143
in	0.141	0.128	0.141
of	0.141	0.128	0.141
shipment	0.143	0.091	0.143
silver	0.091	0.250	0.091
truck	0.091	0.127	0.141

$$P(Q/M_d) = \prod_{t \in Q} P(t/M_d) \times \prod_{t \notin Q} (1.0 - P(t/M_d))$$

for Document D1 Query terms only gold

Remaining terms of document.

$$P(Q/M_{D1}) = P(gold/M_{D1}) \times (1 - P(a/M_{D1}) \times (1 - P(damaged/M_{D1}) \times (1 - P(fire/M_{D1})) \times (1 - P(in/M_{D1})) \times (1 - P(of/M_{D1}) \times (1 - P(Shipment/M_{D1}))$$

$$= 0.143 \times (1 - 0.141) \times (1 - 0.143) \times (1 - 0.143) \times (1 - 0.141) \times (1 - 0.141) \times (1 - 0.143)$$

$$= 0.143 \times 0.859 \times 0.857 \times 0.857 \times 0.859 \times 0.859 \times 0.857$$

$$= 0.000409$$

$P(Q/MD_2)$ & $P(Q/MP_3)$ Similarly calculat the similarity values is shown below table.

Table: Similarity using Language Model

	D_1	D_2	D_3
$P(Q/M_d)$	0.000409	0.001211	0.000743

Hence, as with all other strategies that describing the final ranking is D_2, D_3 and D_1.

CROSS-LANGUAGE INFORMATION RETRIEVAL

CROSS-language Information Retrieval (CLIR) is to allow a user to issue a query in language "L" and have that query retrieve document in language "L'"

The key difference between CLIR and monolingual IR is that the query and the document cannot be matched directly.

The CLIR most approaches use bilingual term lists, term dictionaries, a comparable corpus, or a parallel corpus-

A Comparable corpus is a collection of documents in language L and another collection about the same topic in language L'. The key here is that the documents are not literal translation of each other. that documents only comparable each other.

A Parallel Corpus: is a document in one language L that are then direct translation of language L' or vice versa.

Crossing the Language Barrier:

* What should be translated? Either the queries may be translated, the document, or both queries and documents may be translated to some internal representation
* Which tokens should be used to do a translation (e.g. stems, words, phrases, etc).
* How should we use translation? In other words, a single term in language L may map to several terms in language L', may use of these terms. Some of these terms, or all of these terms. Additionally, might weight some terms higher than other terms
* How can we remove translation? Typically, there are spurious translation that can lead to poor retrieval.

i) Query Translation:-

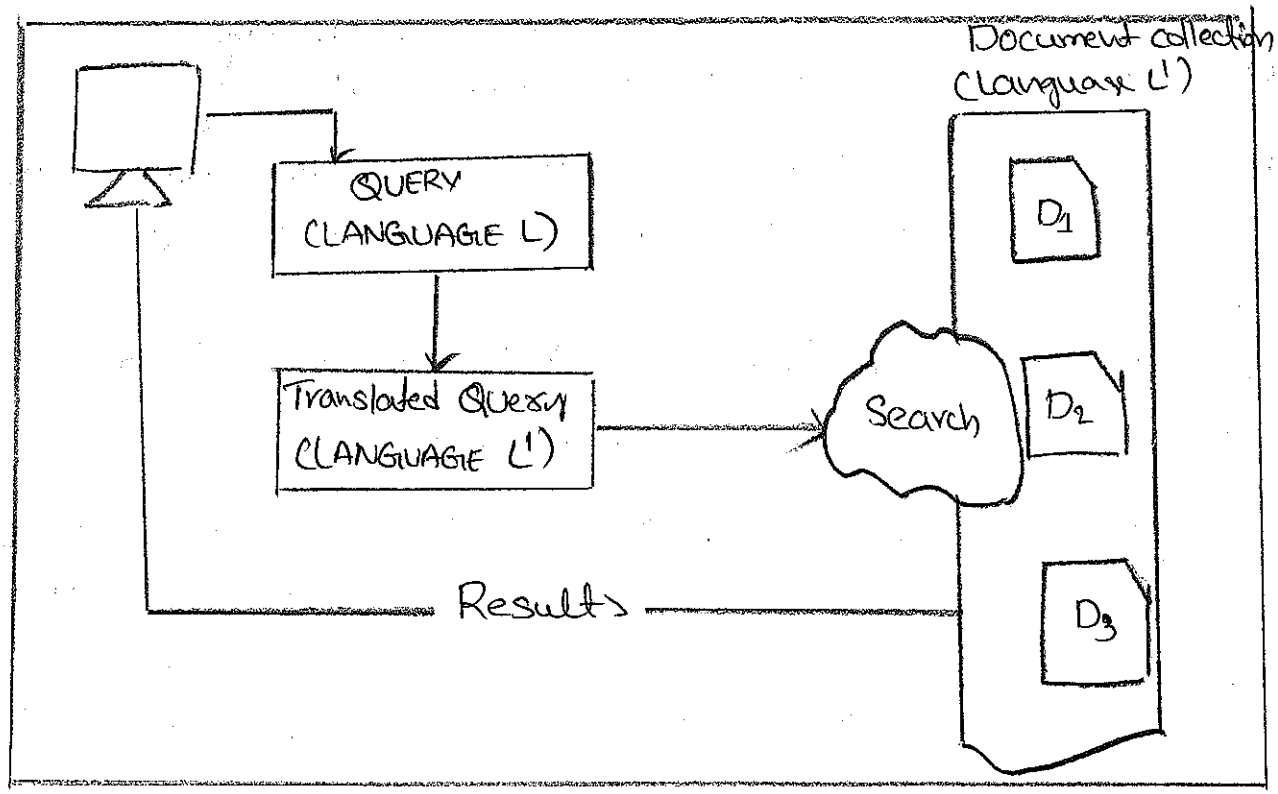


fig: Translate the Query.

User specified keywords were used to represent documents and a dictionary was used to translate English keywords to German keywords.

Query translation approaches use machine translation, language specific stemmers, dictionaries, thesauri, and automatically generated bilingual term list to implement the necessary translation of the user query in language L to the target query language L'.

ii) Document Translation:-

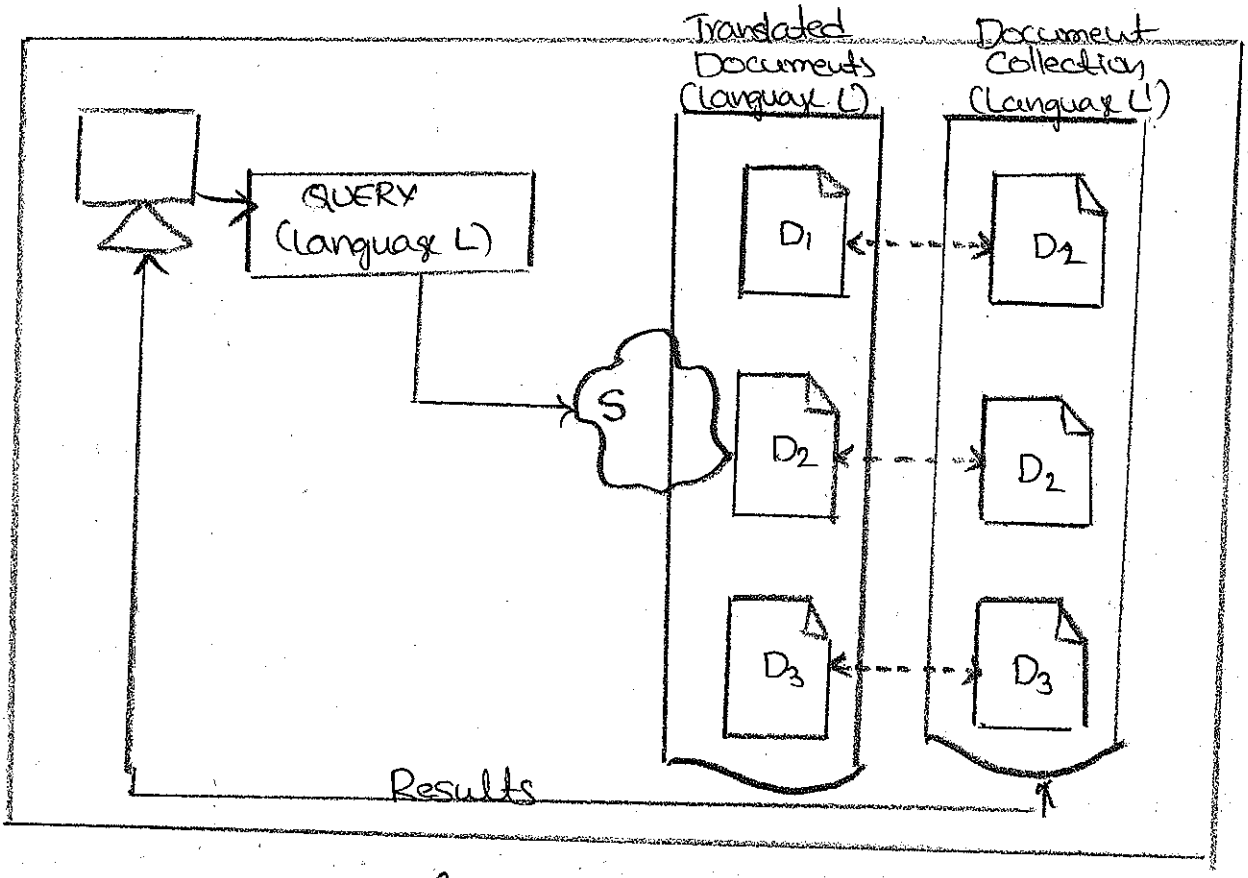


fig: Translate the document.

In document translation use one or more bilingual term lists may be used to translate a document from language L into language L'.

Since the documents contain natural language text it is also possible to run machine translation algorithms to translate from language L to language L'.

c. Phrase Translation:-

Instead of simply using terms for translation, phrase-based approaches were shown to yield substantial improvement. manually translating phrases instead of a term based translation was shown to yield improvement in. A bilingual phrase list extracted from a parallel text. Phrase translation using the collins machine readable dictionary. A core problem with any phrase translation approach is that there is far more likehood that a phrase will not be found in a bilingual list used for translation than a single term.

Choosing Translations:-

Once a bilingual term list or term lists are in place, the next challange is to choose the appropriate translation.

There are four different approaches: unbalanced, structured queries, balanced, and the use of a pivot language.

i) Quality of Bilingual Term List.

Most cross-language systems use a variety of techniques to improve accuracy. At the core of systems are bilingual lexicons. A lexicon is used to map a source term in language L to a myriad of terms in language L'. Key problems with the lexicon are ambiguities, incomplete coverage of the vocabulary

Consider a query with the term duck in English. This term can be translated to many different terms in a target language because there is more than one meaning to the word. Duck can be defined as a "type of bird that floats on lakes or it can be a command on move out of the way.

ii) Unbalanced

This is the most naive form of using a bilingual-term list for a given word t in language L all terms in language L' that are valid translations of t are used. The problem with this is that a query term that a query term that is general may have numerous translations and they are all added to the query.

Meanwhile a query term that is very specific may only have one translation and hence it may well be given less weight that the more general term.

iii) Balanced Queries:-

A term t is translated into multiple terms and these terms are assigned a weight that is then averaged across all translations for the term. Hence a general term with multiple translations will result in all of the translated terms having a lower weight than a very specific translation to only single term.

The weight is a combination of the term frequency (tf) the document frequency (df) and the document length.

w_{ij} – weight of a term t_i in document D_j.

tf_{ij} – term frequency term i in document j

$|D_j|$ — document length

To compute the weight of query term q_i language L. Assume we have k translations of term q_i in language L'. For each of their translation t'_i. We can compute the ~~weight~~ tf'_{ij} and $|D'_j|$ for each document in language L'. Calculate collection frequency, df'_i in language L'. can compute weight w'_i, for each of the translation in language L'.

initial query term in language L we simple average the corresponding translation as:

$$q_i = \frac{\sum_{i=1}^{k} w'_i}{k}$$

iv) Structured Queries:

For Structured Queries, the overall term frequency for a term in a document is computed as the sum of the term frequencies for all of the translations. The corresponding document frequencies for all translations of a query term are combined Assume we again have k translations for query term q_i:

$$tf_i = \sum_{j=1}^{k} tf_{ij}' \qquad df_i = \sum_{j=1}^{k} df_{ij}'$$

These modified term and document weights are then used to compute w_i.

v) Pivot Language:

There are cases when a translation dictionary from language L to L' is not be available. In these cases, a pivot language P can be used to translate from L to P and then from P to L'. This is sometimes referred to as transitive translation.

An approach which uses two different pivot languages is given translating from German to English is done via two routes First translations are obtained by translating from German to Spanish and then to English. Next, these are combined with those from German to Dutch to English.

For example, Consider a query with terms t_1 and t_2 and assume that one translation route resulted in translations e_1, e_2, e_3 and another route obtained e_2 and e_4. The only common term, e_2, would be used against the target language. The remaining terms, e_1, e_3, e_4, would be removed as potentially erroneous translations.

Cross-Language Retrieval Strategies

Cross-language Retrieval strategies develop a similarity measures for a query in language L against documents in language L'.

Language Models for CLIR:

A language modeling approach may be used for cross language retrieval. A simple monolingual language model will compute the likehood of generating a query given a document

$$P(q/D_i \in R) = \prod_{j=1}^{\theta} \alpha P(t_j/C) + (1-\alpha) P(t_j/D_i)$$

Given a query $(t_1, t_2 \ldots t_{|q|})$ where t_j is a query term α is estimated , C – is document collection.

A Good overview on Hidden Markov Models may be found. The remaining probabilities are computed as:

$$P(t_j/C) = \frac{t_i}{|C|}$$

Where t_i – frequency of occurences of terms in the document collection

$$P(t_j/D) = \frac{tf_{ij}}{|D_i|}$$

tf_{ij} – term frequency of term j in document i

$|D_i|$ – Document length.

For cross-language retrieval, the model is extended so a document written in language L generated a query in language L'.

t_L – term in language "L"

$t_{L'}$ – term in language "L'"

D_{Li} – Document i in language "L"

$D_{L'i}$ – Document i in language "L'"

$$P(q_L | D_{L'i} \in R) = \prod_{j=1}^{q} \alpha P(t_{Lj} | C_L) + (1-\alpha) P(t_{Lj} | D_{L'i}) \quad \text{——— (1)}$$

Compute the probability that a query in language L will be generated with a document in language L'.

$$P(t_{Lj} | D_{L'i}) = \sum_{K=1}^{|D_{L'}|} P(t_{L'k} | D_{L'i}) \, P(t_{Lj} | t_{L'k}) \quad \text{——— (2)}$$

$$P(t_{L'k} | D_{L'i}) = \frac{tf_{L'k}}{|D_{L'i}|}$$

The probability $P(t_{Lj} / t_{L'k})$ is the translation probability that a given a term in language L' there will be a term in language L.

Example!

This approach can be used to compute a measure of relevance for our English Query: "gold silver truck" and our three test documents in Spanish. All translation probabilities were derived from the the use of two online bilingual English–Spanish dictionaries

Document collection

De_1: "Shipment of gold ~~silver~~ damaged in a fire"

De_2: "Delivery of silver arrived in a silver truck".

De_3: "Shipment of gold arrived in a truck"

DS_1: El envío del oro daño en un fuego

DS_2: La entrega de la plata llego en un camion de plata

DS_3: El envio del oro llego en un camion.

To find the document in the spanish collection that is most relevant to the English query "gold silver truck", find the translation ~~word~~ of "gold", "Silver", truck. ~~Using~~ ~~Equation~~ ② obtain ~~the~~

$tf_{t,d}$	a	arrived	damaged	delivery	fire	gold	in	of	shipment	Silver	truck
De_1	1	0	1	0	1	1	1	1	1	0	0
De_2	1	1	0	1	0	0	1	1	0	2	1
De_3	1	1	0	0	0	1	1	1	1	0	1

Table1 Raw Terem Frequeng for English collection.

Table 2: Raw Teerm Frequency for Spanish collection

$tf_{t,d}$	camion	dano	de	del	el	en	entrega	envio	fuego	la	llego	oso	plata	un
DS_1	0	1	0	1	1	1	0	1	1	0	0	1	0	1
DS_2	1	0	2	0	0	1	1	0	0	2	1	0	2	1
DS_3	1	0	0	1	1	1	0	1	0	0	1	1	0	1

Table 3: P(t|Ce) for All Terms in the English collection

a	assived	damaged	delivery	five	gold	in	of	shipment	silver	truck
3/22	2/22	1/22	1/22	1/22	2/22	3/22	3/22	2/22	2/22	2/22

Ce - Total terms in English document = 22
t - term frequency.

Table 4: P(t/Cs) for All Terms in the Spanish collection

Camion	dano	de	del	el	en	entrega	envio	fuego	la	llego	oxo	plata	un
2/27	1/27	2/27	2/27	2/27	3/27	1/27	2/27	1/27	2/27	2/27	2/27	2/27	3/27

Cs - Total no of terms in Spanish collection = 27
t - Term frequency.

Table 5: Term Frequency for spanish collection

$tf_{t,d}$	Camion	dano	de	del	el	en	entrega	envio	fuego	la	llego	oxo	plata	un
$\frac{tf}{Ds_1}$	0	1/8	0	1/8	1/8	1/8	0	1/8	1/8	0	0	1/8	0	1/8
$\frac{tf}{Ds_2}$	1/11	0	2/11	0	0	1/11	1/11	0	0	2/11	1/11	0	2/11	1/11
$\frac{tf}{Ds_3}$	1/8	0	0	1/8	1/8	1/8	0	1/8	0	0	1/8	1/8	0	1/8

Table 6: Translation Probabilities P(t_{wj}/t_{ik})

	Camion	dano	de	del	el	en	entrega	envio	fuego	la	llego	oxo	plata	un
gold	0/2	0	0	0	0	0	0	0	0	0	0	1	0/1	0
silver	0/2	0	0	0	0	0	0	0	0	0	0	0/1	1	0
truck	1/2	0	0	0	0	0	0	0	0	0	0	0/1	0/1	0

English - spanish Dictionary
Gold - oxo Silver = plata truck = camion

Using This Equation

$$P(t_{Lj}/D_{L'i}) = \sum_{k=1}^{|D_{L'}|} P(t_{L'k}|D_{L'i})\, P(t_{Lj}|t_{L'k})$$

obtain the probability of a query term appearing in a document from the spanish collection.

$P(Gold|Ds_1) = P(El|Ds_1)\cdot P(gold|El) + P(envio/Ds_1)\cdot P(Gold|envio) +$
$P(del/Ds_1)\cdot P(gold/del) + P(oso/Ds_1)\cdot P(gold/oso) +$
$P(dano/Ds_1)\cdot P(gold/dano) + P(en/Ds_1)\cdot P(gold/en) +$
$P(un/Ds_1)\cdot P(gold/un) + P(fuego/Ds_1)\cdot P(gold/fuego)$

$= \frac{1}{8}\cdot\frac{1}{8} = 1/8.$

$P(gold/Ds_2) =$
$P(la/Ds_2)\cdot P(gold/la) + P(entsega/Ds_2)\cdot P(gold/entrega) +$
$P(de/Ds_2)\cdot P(gold/de) + P(la/Ds_2)\ P(gold/la) +$
$P(plata/Ds_2)\cdot P(gold/plata) + P(llego/Ds_2)\cdot P(gold/llego) +$
$P(en/Ds_2)\cdot P(Gold/en) + P(un/Ds_2)\cdot P(gold/un) +$
$P(camion/Ds_2)\cdot P(gold/camion) = 0$

$P(Gold/Ds_3) = P(El/Ds_3)\cdot P(gold/El) + P(envio/Ds_3)\ P(gold/envio) +$
$P(del/Ds_3)\cdot P(gold/del) + P(oso/Ds_3)\cdot P(gold/oso) +$
$P(llego/Ds_3)\cdot P(gold/llego) + P(en/Ds_3)\cdot P(gold/en) +$
$P(un/Ds_3)\cdot P(gold/un) + P(camion/Ds_3)\cdot P(gold/camion)$

$= 1/8\cdot 1/8 = 1/8.$

$P(Silvex/Ds_1) = P(El/Ds_1)\cdot P(Silvex/El) + P(envio/Ds_1)\cdot P(Silvex/envio) +$
$P(del/Ds_1)\cdot P(Silvex/del) + P(oso/Ds_1)\cdot P(Silvex/oso) +$
$P(dano/Ds_1)\cdot P(Silvex/dano) + P(en/Ds_1)\cdot P(Silvex/en) +$
$P(un/Ds_1)\cdot P(Silvex/un) + P(fuego/Ds_1)\cdot P(Silvex/fuego)$

$= 0$

$P(Silvex/Ds_2) = P(la/Ds_2)\cdot P(Silvex/la) + P(entsega/Ds_2)\cdot P(Silvex/entrega) +$
$P(de/Ds_2)\cdot P(Silvex/De) + P(la/Ds_2)\cdot P(Silver/la) +$
$P(plata/Ds_2)\cdot P(Silvex/plata) + P(llego/Ds_2)\cdot P(Silvex/llego) +$
$P(en/Ds_2)\cdot P(Silver/en) + P(un/Ds_2)\cdot P(silver/un) + P(camion/Ds_2)\cdot P(silver/camion)$

$= \frac{2}{11}\cdot 1/1 = 2/11$

$P(\text{silver}/DS_3) =$

$P(El/DS_3) \cdot P(\text{silver}/El) + P(\text{envio}/DS_3) \cdot P(\text{silver}/\text{envio}) +$
$P(del/DS_3) \cdot P(\text{silver}/del) + P(\text{oso}/DS_3) \cdot P(\text{silver}/\text{oso}) +$
$P(\text{llego}/DS_3) \cdot P(\text{silver}/\text{llego}) + P(en/DS_3) \cdot P(\text{silver}/en) +$
$P(un/DS_3) \cdot P(\text{silver}/un) + P(\text{camion}/DS_3) \cdot P(\text{silver}/\text{camion})$

$= 0$

$P(\text{truck}/DS_1) =$

$P(El/DS_1) \cdot P(\text{truck}/El) + P(\text{envio}/DS_1)\ P(\text{truck}/\text{envio}) +$
$P(del/DS_1) \cdot P(\text{truck}/del) + P(\text{oso}/DS_1) \cdot P(\text{truck}/\text{oso}) +$
$P(\text{dano}/DS_1) \cdot P(\text{truck}/\text{dano}) + P(en/DS_1) \cdot P(\text{truck}/en) +$
$P(un/DS_1) \cdot P(\text{truck}/un) + P(\text{fuego}/DS_1) \cdot P(\text{truck}/\text{fuego})$

$= 0.$

$P(\text{truck}/DS_2) =$

$P(la/DS_2) \cdot P(\text{truck}/la) + P(\text{entrega}/DS_2) \cdot P(\text{truck}/\text{entrega}) +$
$P(del/DS_2) \cdot P(\text{truck}/del) + P(la/DS_2) \cdot P(\text{truck}/la) + P$
$P(\text{plata}/DS_2) \cdot P(\text{truck}/\text{plata}) + P(\text{llego}/DS_2) \cdot P(\text{truck}/\text{llego}) +$
$P(en/DS_2) \cdot P(\text{truck}/en) + P(un/DS_2) \cdot P(\text{truck}/un) +$
$P(\text{camion}/DS_2) \cdot P(\text{truck}/\text{camion})$

$= \frac{1}{11} \cdot \frac{1}{2} = 1/22$

$P(\text{truck}/DS_3) = P(El/DS_3) \cdot P(\text{truck}/El) + P(\text{envio}/DS_3) \cdot P(\text{truck}/\text{envio}) +$
$P(del/DS_3) \cdot P(\text{truck}/del) + P(\text{oso}/DS_3) \cdot P(\text{truck}/\text{oso}) +$
$P(\text{llego}/DS_3) \cdot P(\text{truck}/\text{llego}) + P(en/DS_3) \cdot P(\text{truck}/en) +$
$P(un/DS_3) \cdot P(\text{truck}/un) + P(\text{camion}/DS_3) \cdot P(\text{truck}/\text{camion})$

$= \frac{1}{8} \cdot \frac{1}{2} = 1/16.$

	DS_1	DS_2	DS_3
Gold	1/8	0	1/8
Silver	0	2/11	0
Truck	0	1/22	1/16

Now apply equation

$$P(q_k | D_{i_e} \in R) = \prod_{j=1}^{q} \alpha P(t_{kj} | C_L) + (1-\alpha) P(t_{kj} | D_{i_e})$$

find the probability of the query appearing in each Spanish documents. Here $\alpha = 0.3$

$P(Gold, silver, truck | D_{S_1}) =$

$[0.3) P(gold/Ce) + (1-0.3) P(gold/D_{S_1})] \times$

$[(0.3) P(Silver/Ce) + (1-0.3) P(Silver | D_{S_1})] \times$

$[(0.3) P(truck/Ce) + (1-0.3) P(truck | D_{S_1})]$

$= (0.3 \cdot (2/22) + 0.7 (1/8)) (0.3 (2/22) + 0.7(0)) (0.3 (2/22) + 0.7(0))$

$= 0.0000854$

$P(Gold, Silver, truck / D_{S_2}) =$

$[(0.3] P(gold/Ce) + (1-0.3) P(Gold | D_{S_2})] \times$

$[(0.3) P(Silver/Ce) + (1-0.3) P(Silver/D_{S_2})] \times$

$[(0.3) P(truck/Ce) + (1-0.3) P(truck/D_{S_2})]$

$= (0.3 \times \frac{2}{22} + 0.7 \times 0) (0.3 \times \frac{2}{22} + 0.7 \times \frac{2}{11}) [0.3 \times \frac{2}{22} + 0.7 \times 1/22]$

$= 0.0002491$

$P(Gold, Silver, truck / D_{S_3}) =$

$[(0.3) P(gold/Ce) + (1-0.3) P(gold/D_{S_3})] \times$

$[(0.3) P(Silver/Ce) + (1-0.3) P(Silver/D_{S_3})] \times$

$[(0.3) P(truck/Ce) + (1-0.3) P(truck/D_{S_3})]$

$= (0.3 \times \frac{2}{22} + 0.7 \times \frac{1}{8}) (0.3 \times \frac{2}{22} + 0.7 \times 0) (0.3 \times \frac{2}{22} + 0.7 \times 1/16)$

$= 0.0002223.$

D_{S_2} gives the highest probability, therefore, it is most likely to be relevant to the query.

EFFICIENCY

Sequential information retrieval algorithms are difficult to analyze in detail, as their performance is often based on the selectivity of an information retrieval query.

Inverted Index:-

Since many document collections are reasonably static it is feasible to build an inverted index to quickly find terms in the document collection. Instead of scanning the entire collection, the text is preprocessed and all unique terms are identified. This list of unique terms is referred to as the index. For each term, a list of documents that contain the term is also stored. This list is referred to as a "posting list".

An entry in the list of documents can also contain the location of the term in the document to facilitate Proximity Searching. Additionally, an entry can obtain a manually or automatically assigned weight for the term in the document. This weight is frequently used in computations that generate a measure of relevance to the query. Once this measure is computed, the document retrieval algorithm identifies all the documents that are "relevant" to the query by sorting the co-efficient and presenting a ranked list to the user.

Upon receiving a query, the index is consulted, the corresponding posting list are retrieved, and the algorithm ranks the documents based on the contents of the posting list.

~~Building an~~ I

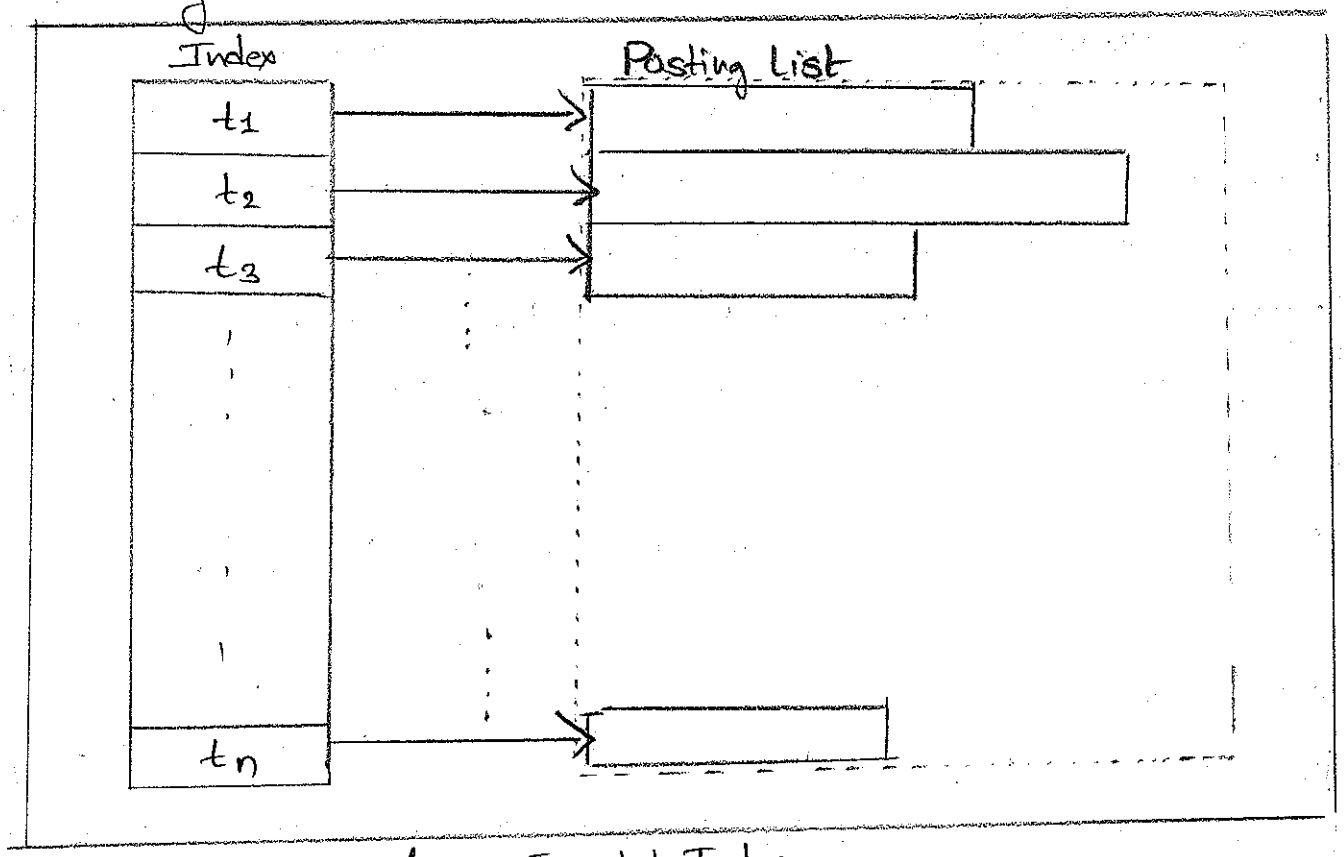


fig : Inverted Index

Building an Inverted Index:-

An Inverted index consists of two components, a list of each distinct term referred to as an index and a set of lists referred to as posting lists. To compute relevance ranking, the term frequency or weight must be maintained. Thus, a posting list contains a set of tuples for each distinct term in the collection. The set of tuples is of the form <doc_id, ff)

The entries in the posting list are stored in ascending order by document numbers. An inverted index is constructed by stepping through the entire document collection, one term at a time. The output of the index construction algorithm is a set of files written to disk. These files are.

* Index file:- Contain the actual posting list for each distinct term in the collection. A term, t that occurs in "i" different documents will have a posting list of the form:

$$t \rightarrow (d_1, tf_{1j}), (d_2, tf_{2j}) \ldots\ldots (d_i, tf_{ij})$$

Where d_i indicates the document identifiers of document "j"

tf_{ij} indicates the number of times term "j" occurs in document i.

* document file:- Contains information about each distinct document - document identifier, document name, etc.

* Weight file:- contains the weight for each document

The construction of inverted index is implemented by scanning the entire collection, one term at a time. A hash function is used to quickly locate the term in an array. collisions caused by the hash function are resolved via linear linked list. Once the posting list corresponding to this term is identified, the first entry of the list is checked to see if its document identifier matches the current document.

A typical uncompressed index spends four bytes on the document identifiers and two bytes on the term frequency.

Compressing an Inverted Index:-

A key objective in the development of inverted index files is to develop algorithms that reduce I/O bandwidth and storage overhead. Since large index files demand greater I/O bandwidth to read them, the size also directly affects the processing times.

Fixed Length Index Compression:-

In a posting list are in ascending order by document identifier, run length encoding is applicable for document identifiers. For any document identifier, only the offset between the current identifier and the identifier immediately preceding it are computed. a compressed version of the document identifier is stored.

This scheme effectively reduces the domain of the identifiers, allowing them to be stored in a more concise format. Subsequently, the following method is applied to compress the data.

For any given input value, the two left-most bits are reserved to store count for the number of bytes that are used in storing the value. thus, a two bit length indicator is used for all document identifiers. Integers are stored in either 6, 14, 22, or 30 bits. Optimally, a reduction of each individual data record size by a factor of four is obtained by this method since, in the best case, all values are less than $2^6 = 64$ and can be stored in single byte. without Compression, four bytes are used for all document identifier.

Example: Fixed length compression:-

Consider an entry for an arbitary term t_1, which indicates that t_1 occurs in document 1, 3, 7, 70, 8250. Byte-aligned (BA) Compression uses the leading two high order bits to indicate the number of bytes required to represent the value. The differences between entries in the posting list must be computed. The difference of 250 − 70 = 180 is all that must be computed for this final value. The values and their corresponding compressed bit strings are shown below tables.

Using no compression, the five entries in the posting list require four bytes each for a total of twenty bytes. This example, uncompressed data require 160 bits, while BA Compression requires only 48 bits.

Table: Storage Requirements Based on Integer size

Length	Number of Bytes Required
$0 \le x < 64$	1
$64 \le x < 16,384$	2
$16,384 \le x < 4,194,304$	3
$4,194,304 \le x < 1,073,741,824$	4

Table: Byte Aligned Compression

Value	Compressed Bit string
1	00 000001
2	00 000010
4	00 000100
63	00 111111
180	01 000000 10110100

Table: Baseline: No Compression

value	Uncompressed Bit string
1	00000000 00000000 00000000 0000 0001
3	00000000 00000000 00000000 0000 0011
7	00000000 00000000 00000000 0000 0111
70	00000000 00000000 00000000 01000110
250	00000000 00000000 00000000 11111010

Variable length Index Compression:-

Moffat an Zobel also use the differences in the posting list. They first mention that patterns can be seen in these differences and that Huffman encoding provides the best compression.

Moffat an Zobel use a family of universal codes described. This code represents an integer x with $2\lfloor \log_2 x \rfloor + 1$ bits. The first $\lfloor \log_2 x \rfloor$ bits are the unary representation of $\lfloor \log_2 x \rfloor$. (unary representation is a base one representation of integer using only the digit one). After the leading unary representation, the next bit is a single stop bit of ZERO. At this point, the highest power of two that doesn't exceed x has been represented. The next $\lfloor \log_2 x \rfloor$ bits represent the remainder of $x - 2^{\lfloor \log_2 x \rfloor}$ in binary.

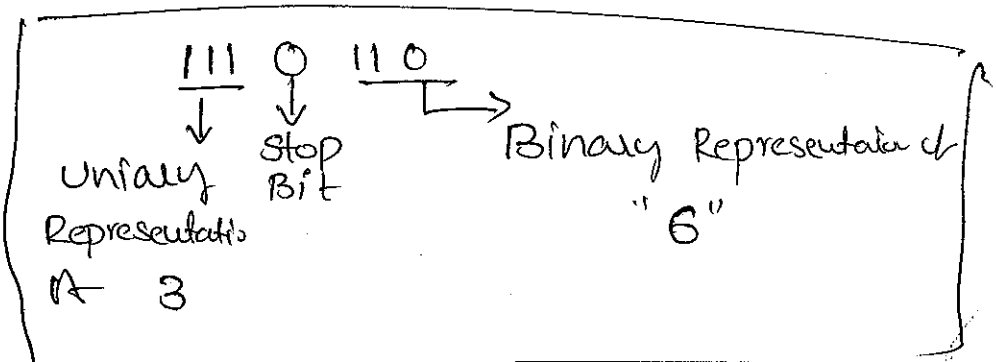
Example: Compression of the decimal 14.

First $\lfloor \log_2 x \rfloor = \lfloor \log_2 14 \rfloor = 3$ is represented in unary as "111".

Next, the stop bit is used. Subsequently, the remainder of $x - 2^{\lfloor \log_2 x \rfloor} = 14 - 2^3 =$

$$14 - 8 = 6$$

is stored in binary using ~~log~~ "110"



Example : Variable length Compression:

Consider an entry for an arbitary term t_1, which indicates that t_1 occurs in documents 1, 3, 7, 70, & 250. The differences of 1, 2, 4, 63, and 180 are compressed values as given below table. This requires only 35 bits. thirteen less than the simple BA Compression.

Table: Gamma Compression

Value	Compressed Bit String
1	0
2	10 0
4	11 0 00
63	11111 0 11111
180	1111111 0 0110100

INDEX Pruning:-

The lossless approaches for inverted index compression, A lossy approach is called static index pruning. Static pruning simply elimnates posting list entries in a uniform-fashion - regardless of the term.

Reordering Documents Prior to Indexing:-

Index compression efficiency can also be improved if we use an algorithm to reorder documents prior to compressing the inverted index. Since the compression effectiveness of many encoding schemas is largely dependent upon the gap between document identifiers, the idea is that if we can feed documents to the algorithm correctly we reduce the average gap, thereby maximizing compression.

Consider documents d_1, d_{99}, and d_{1000} all which contain the same term t. For these documents we obtain a posting list entry for t of $t \rightarrow d_1, d_5, d_{101}$.

Each algorithm uses the Jaccard similarity co-efficient to obtain a measure of document similarity. Two basic approaches have been proposed. Top-down or Bottom-up

Top-Down:-

The Top-Down algorithms consist of four main phases. In the first phase, called Center Selection, two groups of documents are selected from the collection and used as pattions in subsequent phases. In the redistribution phase, all remaining documents are divided among the selected centers according to their similarity. In the recursion phase, the previous phases are repeated recursively over the two resulting partitions until each one becomes a singleton. Finally, in the merging phase, the pattions formed from each recursive call are merged bottom-up, creating an ordering.

The first of the two proposed top-down algorithm is called transactional B&B. as it is an implentation of the Bellochand Blandford algorithm.

The second top-down algorithm is called Bisecting, So named because its center selection phase consist of choosing two random documents as centers, thereby dramatically reducing the cost of this phase.

Bottom-up:-

The bottom-up algorithm begin by considering each document in the collection separately and they progressively group documents based on their similarity. The bottom-up inspired by the popular k-means approach to document clustering. The second uses k-scan:

This K-means algorithm initially chooses k documents as cluster representatives, and assigns all remaining documents to choose clusters based on a measure of similarity. At the end of this first ~~phase~~ pass, the cluster centroid are recomputed and the documents are reassigned according to their similarity to the new centroids. This iteration continues until the cluster centroid stabilize. The single pass version of this algorithm only performs the first pass of this algorithm. The K-scan algorithm is a simplified version of single-pass K-means.

Query Processing:-

The query performance can be improved by modifying the inverted index to support fast scanning of a posting list. The reasonable precision and recall can be obtained by retrieving fewer terms in the query.

Inverted Index Modification:

Moffat and Zobel show how an inverted index can be segmented to allow for quick search of a posting list to locate a particular document. The typical ranking algorithm scans the entire posting list for each term in the query. An array of document scores is updated for each entry in the positing list. Moffat and Zobel suggest that the least frequent term should be procesed first. The less frequent terms carry the most meaning and ~~pos~~ probably have the most significant contribution to a high-ranking documents. The entire positing list for these terms are processed.

The high-frequency terms in the query will simply be generating scores for document that will not end up in the final top t documents, where "t" is the number of documents that are displayed to the user.

A suggested improvement is to continue processing all the terms in the query, but only update the weights found in the d documents. In other words after some threshold of d scores has been reached, the remaining query terms become part of an AND instead of the usual vector space OR.

Prior to reaching d scores the basic algorithm is:

```
For each term t in the Query Q
    Obtain the posting list entries for t
    For each posting list entry that indicates "t" is in doc "i"
    Update score for document i.
```

For query terms with small posting list, the inner loop is small: term are very frequent are examined, long posting lists are prevalent. After d documents are accessed, there is no need to update the score for every document.

To avoid scanning very long Posting lists, the algorithm is modified to be:

```
For each term t in the query Q
    Obtain posting list, p, for document that contain t
    For each document "x" in the reversed list of "d" documents
    Scan posting list p for x
    If x exists
        update score for documents x.
```

Partial Result Set Retrieval:-

Another way to improve run-time performance is to stop processing after some threshold of computational resources is expended. One approach counts disk I/O operations and stops after a threshold of disk I/O operations is reached. Three measures used to characterize a posting list are now described.

i) Cutoff Based on Document frequency:-

To simplest measures of term quality is to rely on document frequency. This was using between twenty-five to seventy-five percent of the query terms after they were sorted by document frequency resulted in almost no degradation in precision and recall improves with fewer terms because lower ranked terms have long posting lists that result in scoring thousands of documents and do little to improve the quality of the result.

ii) Cutoff Based on Maximum Estimated weight:-

Two other measures of sorting the query terms are described The first computes the maximum term frequency of a given term as tf_{max} and uses the following as a means of sorting the query.

$$tf_{max} \times idf$$

iii) Cutoff Based on the weight of a Disk Page in the Posting List:

The cutoffs based on term weights can be used to characterize posting lists and choose which posting list to process first. The problem is that a posting list can be quite long and might have substantial size. At index creation time, the posting lists are sorted by decreasing order by term frequency and instead of just a

a pointer to the first entry in the posting list, the index contains an entry for each page of the posting list.

The posting list pages are then sorted by:

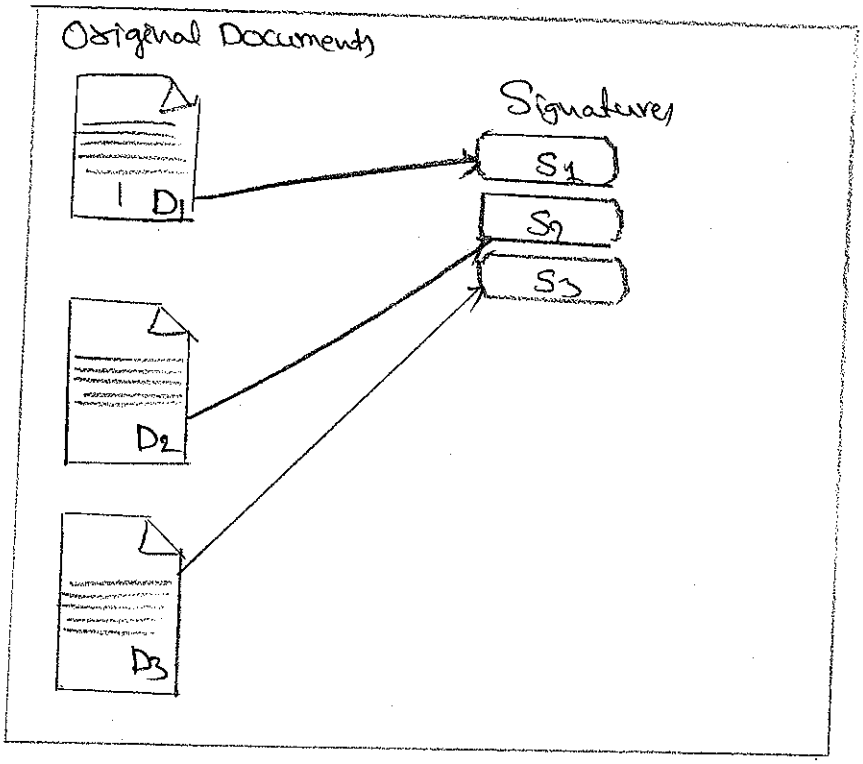
$$tf_{max} \times idf \times f(I)$$

Where $f(I)$ is a function that indicates the number of entries on a page. This is necessary since some pages will not be full and a normalization is needed such that they are not sorted in exactly the same way as a full page. One value that is used for $f(I)$ is e^a where $0 < e < 1$.

Signature Files:-

The use of Signature files lies between a sequential scan of the original text and the construction of an inverted index. A signature is an encoding of a document. The idea is to encode all documents as relatively small signatures. Once this is done, the signatures can be scanned instead of the entire documents. Typically, signatures do not uniquely represent a document. so it is usually necessary to implement a retrieval in two phases. The first phase scans all of the signatures and identifies possible hits, and second phase scans the original text of the documents in the possible hit list to ensure that they are correct matches.

Construction of a signature is often done with different hashing functions. One or more hashing functions are applied to each word in the document.



term	h(term)
t_1	0101
t_2	1010
t_3	0011

Building a Signature

The hasing function is used to set a bit in the Signature. for example, if the terms Information and retrieval were in a document and h(information) and h(retrieval) corresponded to bits one and four respectively, a four bit binary signature for this document would appear as 1001.

Consider a terms t_1 ~~that~~, t_2 and t_3 has function respectely shown above table. document d_1 that contains t_1, document d_2 contains t_1 and t_3 and document d_3 contain t_1 and t_2.

The signatures for these three documents

Table: Document Signatures.

Document	Signature
d_1	0101
d_2	0111
d_3	1111

Hence a query that is searching for term t_3 will obtain a false match on document d_3 even though it does not contain t_3.

To implement document retrieval, a signature is constructed for the query. A Boolean AND is executed between the query signature and each document signature. If the AND returns TRUE, the document is added to the possible hit list. Similarly, a Boolean OR can be executed if it is only necessary for any word in the query to be in the document. To minimize false positives, multiple hashing functions are applied to the same word.

Signature are useful if they can fit into memory. Also, it is easier to add or delete document in a signature file to an inverted index, and the order of an entry in the signature file does not matter.

Scanning to Remove False Positives

Once a signature has found a match, scanning algorithms are employed to verify whether or not the match is a false positive due to collisions.

Signature algorithms can be employed without scanning for false drops and no degradation in precision and recall occurs

Pattern matching algorithms are related to the use of scanning in information retrieval since they strive to find a pattern in a string of text characters. Pattern matching is defined as finding all positions in the input text that contain the start of a given pattern. If the pattern is of size P and the text is of size S, the naive nested loop pattern match requires O(PS) comparisons.

Aho and Corasick's algorithms implement deterministic finite state automata to identify matches in the text. Knuth, Morris, and Pratt (KMP) also describe an algorithm that runs in O(S) time that scans forward along the text, but uses preprocessing of the pattern to determine appropriate skips in the string that can be safely taken.

The Boyer-Moore algorithm is another approach that preprocesses the pattern, but starts at the last character of the pattern and works backward towards the beginning of the string.
The FSA scans text from right to left, as done in Boyer-Moore.

Duplicate Document Detection:-

A method to improve both efficiency and effectiveness of an information retrieval system is to remove duplicate or near duplicates. Duplicates can be removed either at the time documents are added to an inverted index or upon retrieving the result of a query.

The difficult is that wish to remove exact duplicates, and also removing near duplicate as well. The user users should only be shown two hyperlinks, but instead is shown thousands, these redundant pages can effect term and document weighting schemes. Additionally, they can increase indexing time and reduce search efficiency.

I) Finding Exact Duplicates.

Duplicate detection is often implemented by calculating a unique hash value for each document. Each document is then examined for duplication by looking up the hash value in either an in-memory hash or persistent lookup systems. Several common hash functions used are MD2, MD5, or SHA. ~~These~~ They can be calculated on arbitrary document lengths, they are easy to compute, and they have very low probabilities of collisions

While this approach is both fast and easy to implement, the problem is that it will find only exact duplicates.

The Several metrics for determining the similarity of a document to another. The first is resemblance. If a document contains roughly the same semantic content, it is a duplicate whether or not it is a precise syntatic match.

$$r(D_i, D_j) = \frac{|S(D_i) \cap S(D_j)|}{|S(D_i) \cup S(D_j)|}$$

The resemblance r of document D_i and document D_j is defined. This metric can be used to calculate a fuzzy similarity between two documents.

For example assume D_i and D_j share fifty percent of their terms and each document has 10 terms.

Their resemblance would be $\frac{5}{15} = 0.33$

Using resemblance to provide a threshold "t" to find duplicate documents. Where if t was exceeds the documents could be considered duplicate.

a. shingles:-

A shingles is simply a set of contiguous terms in a document. Shingling techniques, such as COPS, and DSC, essentially all compare the number of matching shingles.

The hash values for each document subsection / feature set and filters those hash values to reduce the number of comparisons the algorithm must perform. This filtration of features, therefore, improves the runtime performance. In the shingling approaches, rather than comparing documents, subdocuments are compared, and thus, each document can produce many potential duplicates.

To combat the inherent efficiency issues, several optimization techniques were proposed to reduce the number of comparisons made.

The DSC Algorithm reduces the number of shingles used. Frequently occuring shingles are removed. Every 25th shingles is saved in. This reduction, however, hinder/ effectiveness. The implementation of the DSC algorithm took 10 CPU days to process 30 million documents

The DSC algorithm has a more efficient alternative, DSC-SS, which uses Super Shingles. This algorithm takes several shingles and combine them into a Super Shingle. The result is a document with a few Super Shingles instead of many shingles.

b Duplicate Detection via Similarity:-

Another approach is to simply compute the similarity coefficient between two documents. If the document similarity exceeds a threshold, the documents can be deemed duplicates of each other. These approaches are similar to work done in document clustering. In this approach require all pairs of documents to be compared, i.e., each document is compared to every other document and a similarity weight is calculated. A document to document similarity comparsion approach is thus computationally prohibitive given the theoretical $O(d^2)$ runtime, where "d" is the number of documents.

c Treating the Document as a Query:-

The Result document as a new query and looks for other documents that match this document. This approach is not computationally feasible for large collections or dynamic collections since each document must be queried against the entire collection. For large collections, where a common term can occur in millions of documents, this is not scalable.

d: I-Match

I-Match uses a hashing scheme that uses only some terms in a document. The decision of which terms to use is key to the success of the algorithm. I-match is a hash of the document that uses collection statistics, for example idf, to identify which terms should be used as the basis for comparison. The use of collection statistics allows one to determine the usefulness of terms for duplicate document detection.

I-Match assumes that removal of very infrequent terms or very common terms result in good document representation for identifying duplicates.

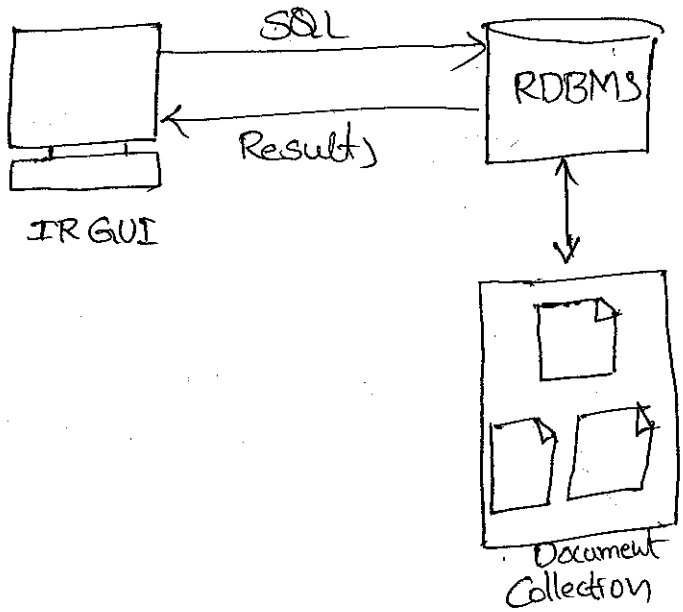
Pseudo-code for the algorithm.

* Get document
* Parse document into a token steam, removing format tags
* Using term thresholds (idf), retain only significant tokens.
* Insert relevant tokens into unicode ascending ordered tree of unique tokens.
* Loop through token tree adding each unique token to the SHA1(4) digest. Upon completion of loop, a (doc-id, SHA1 digest) tuple is defined.
* The tuple (doc-id, SHA1 digest) is inserted into the storage data structures using the key.
* If there is a collision of digest values then the document are similar.

INTEGRATING STRUCTURED DATA AND TEXT

Essential problems associated with searching and retrieving relevant documents were discussed in the preceding.

Airline reservation systems, automated teller machines, and credit card validation devices are all systems that pervade everyday life. Each replete with structured data. Most production systems implemented with a relational database management systems (RDBMS) have some text — such as a comment field — which allows users to enter a free text comment a particular order. Commercial database systems allow users to store these unstructured fields in Binary large objects (BLOB) or character large OBJects (CLOB) datatypes Unstructured data to be stored in a relation.



IR as an application of a RDBMS.

Review of the Relational Model:-

The relational model was initially described by Codd [Codd 1970] Prior data models were navigational. in that application developers had to indicate the means by which the database should be traversed. The relational model stores data in relations and enables the developer to simply describe what data are required not how to obtain the data.

Relational Database, Primitives and Nomenclature:-

A relational database system stores data in set-theoretic relations. An atribute within a relation is any symbol from a finite set $\mathcal{L} = \{A_0, A_1, A_2 \ldots A_n\}$ A Relation R on the set $\mathcal{L}$ is a subset of the cartesian product $dom(A_0) \times dom(A_1) \times dom(A_2) \ldots dom(A_n)$ where $dom(A_i)$ is the domain of A_i.

EMP_PROJ (Emp_no, Project, delivery_date, budget)

PROJECT (Project, delivery_date, Budjet)

Above relations exist one to represent the EMP entitity, one to represent the PROJECT entity, and one to represent the relation EMP_PROJ that exists between the two relations.

Normalization is the process of ensuring the data base design satisfies very specific rules developed to reinforce the consistency and integrity of the data. First Normal Form (1NF) simply indicates that data are stored in single-valued attributes.

A relation is in ~~stored~~ second normal form (2NF) if all attributes are fully dependent on the primary key of the relation. An entity is in third normal form (3NF) if all attributes of the entity are dependent on the primary key of the relation and are not also dependent on another key. The primary key is one or more attributes that uniquely identifies a tuple in a relation. A database should satisfy at least 3NF.

A Good overview of SQL can be found in has the structure.

SELECT <list of attributes>
FROM <List of Relations>
WHERE <list of conditions>
ORDER BY <list of attributes>
GROUP BY <list of attributes>
HAVING <list of conditions>

Example Query's

> SELECT emp-no FROM EMP WHERE emp_no = 400;

> SELECT emp-no, COUNT(*) FROM EMP_PROJ GROUP BY emp-no

> SELECT emp-no, COUNT(*) FROM EMP_PROJ
GROUP BY emp-no HAVING COUNT(*) > 3

> SELECT emp-no FROM EMP_PROJ ORDER BY emp-no

> SELECT a.emp_no, a.age FROM EMP a, EMP-PROJ b
WHERE a.emp-no = b.emp-no AND
b.project = 'A'

A Historical Progression:-

That Combine information retrieval and DBMS together, or Systems that extend relational DBMS to include information retrieval functionality

Combining Separate Systems:-

Several ~~app~~ Integrated solutions which consist of writing a central layer of software to send requests to underlying DBMS and information retrieval system. Queries are passed and the structured portions are submitted as a query to the DBMS, while text search portions of the query are submitted to an information retrieval systems. The results are combined and presented to the user.

The advantage of this approach is that the DBMS & IRS are commercial products that are continuously improved upon by vendors. Additionally, software development costs are minimized. The disadvantage include Poor data integrity, portability, and run-time performance.

Data Integrity:-

Data integrity is sacrificed because the DBMS transaction log and the IRS transaction 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 IRS log would not know about the DBMS log. Hence, the umbrella application that coordinates work between the two systems must handle all recovery.

An update that must take place in both systems can succeed in the DBMS, but fail in the IRS. A partial update is clearly possible, but is logically flawed.

ii) 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 develop 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 systems is brought in.

iii) Performance:-

Run-time Performance suffers because of the lack of parallel processing and query optimization. Although most commercial DBMS have parallel implementation, most information retrieval systems do not.

User-defined Operators:-

User defined operators that allow users to modify SQL by adding their own functions to the DBMS engine ~~were described~~. An example query that uses the user-defined area function that accepts a single arguments. The datatype of the argument is given as rectangle. Hence this example uses both a user defined function and a user defined datatype.

Ex. 1. SELECT MAX(AREA(rectangle)) FROM SHAPE.

In the information retrieval domain, an operator such as proximity() could be defined to compute the result set for a proximity search.

The following query obtains all documents that contain the terms term1, term2, and term3.

Ex 2: SELECT Doc-Id FROM DOC

WHERE PROXIMITY (Text, 6, Term1, Term2, Term3).

The advantages of user-defined operators are that they not only solve the problem for text, but also solve it for spatial data, image processing etc.

The key problems with userdefined operators against are integrity, portability, and runtime Performance.

i) 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 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.

To protect the DBMS from a faulty user-defined operator, a remote procedure call (RPC) can be invoke the user-defined operator.

ii) Portability:-

A user defined operator implemented at SITE A may not be present at SITE B. Worse, the operator may appear to exist, but it may perform an entirely different function. Without user-defined operators, anyone with an RDBMS may write an application and expect it to run at any site that runs that RDBMS.

iii) Performance:

Query optimization, by default, does not know much about the specific user defined operators. Optimization is often based on substantial information about the query. A query with an EQUAL operator can be expected to retrieve fewer rows than a LESS THAN Operator. This knowledge assists the optimizer in choosing an access path.

Without knowing the semantics of a user-defined operator the optimizer is unable to efficiently use it. Some user-defined operators might require a completely different access structure like an inverted index.

Bibliographic Search with Unchanged SQL.

Blair explored the potential of relational systems to provide typical information retrieval functionality. Blair's work included queries using structured data with unstructured data. The following relations model the document collection.

- DIRECTORY (name, institution) – identifies the author's name and the institution the author is affiliated with.
- AUTHOR (name, DocId) – indicates who wrote a particular doc-
- INDEX (term, DocId) – identifies terms used to index a particular document.

The following query ranks institution based on the number of publications that contain input-term in the document.

```
Ex. SELECT UNIQUE institution, COUNT(UNIQUE name)
    FROM DIRECTORY
    WHERE name IN (SELECT name FROM AUTHOR WHERE DocId IN
                   SELECT DocId FROM INDEX WHERE term = input-term
                   ORDER BY 2 DESCENDING))
```

Blair cites several benefits for using the relational model as a foundation for document retrieval. These benefits are the basis for providing typical information retrieval functionality in the relational model.

Information Retrieval as a Relational Application:

Work with extensions to SQL started ~~first item~~ w. Intitial extensions described by Macleod are based on the use of a QUERY (term) relation that stores the term in the query and, an INDEX (DocId, term) relation that indicates which terms appear. in which documents. The following Query lists all the identities of documents that contain at least one term in QUERY:

Ex: SELECT DISTINCT (i.DocId) FROM INDEX i, QUERY q
WHERE i.term = q.term

Frequently used terms or stop terms are typically eliminated from the document collection. Therfore STOP-TERM relation may be used to store the frequently used terms. The STOP-TERM relation contains a single attribute (term).

Q: to identify documents that contain any of the terms in the query except those in the STOP-TERM relation is given below.

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

to implement a logical AND of the terms InputTerm1, InputTerm2 on inputTerm3, Macleod and Crawford proposed the following query

Ex:- SELECT DocId FROM INDEX WHERE term = InputTerm1
INTERSECT
SELECT DocId FROM INDEX WHERE term = InputTerm2
INTERSECT
SELECT DocId FROM INDEX WHERE term = InputTerm3

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

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. The markers in the working example are found in the TIPSTER collection which was previous years as the standard dataset for TREC. These markers begin with a < and end with > (e.g., <TAG>)

The first relation, DOC, contains information about each document.

The second relation, INDEX models the inverted index and indicates which term appear in which document and how often the term has appeared

The relations are:

INDEX (DocId, Term, TermFrequency)

DOC (DocId, DocName, PubDate, Dateline)

These two relations are built by the preprocessor. A Third TERM relation tracks statistics for each term based on its number of occurrences across the document collection. At a minimum, this relation contains the Document Frequency (df) and Inverse Document frequency (IDF). These ~~were described~~

The term relation is of the form: TERM(Term, Idf).

The INDEX relation has one hundred millions tuples or more. This requires one hundred million separate calls to the DBMS INSERT function. with each insert, a transaction log is updated.

All relational DBMS provide some type of bulk-load facility in which a large flat file may be quickly migrated to a relation without significant overhead.

The DOC and INDEX tables are output by the preprocessor. The TERM Relation is not output. In the initial testing of the preprocessor, it was found that this table was easier to build using the DBMS than within the preprocessor.

A WORKING Example:-

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 </DOCNO>
<HL> Italy's Commercial Vechicle 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>

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

<DATELINE> DuPont Company, Wilmington, DE </DATELINE>
<TEXT>
John A. Krol was named group vice president, Agriculture Products department, of this diversified chemicals company, succeeding Dale E. Wolt, who will retire May 1. Mr. Krol was formerly vice president in the Agricultural Products department.
</TEXT>
</DOC>

The preprocessor accepts these two documents as input and create the two files that are then loaded in to the relational DBMS.

The corresponding DOC and INDEX relations are given follo Tables

Table DOC

DocId	DocName	PubDate	Dateline
1.	WSJ870323-0180	3/23/87	TURIN, Italy
2.	WSJ870323-0161	3/23/87	DuPont Company, Wilmington, DE

Table INDEX

DocId	Term	Term Frequ
1	Commercial	1
1	Vechicle	1
1	Sales	1
1	Italy	1
1	February	1
1	Year	1
- -	- - - -	- - -
2	Krol	2
2	President	2
2	diversified	1
2	Company	1
2	Succeeding	1
- -	- - -	- - -

INDEX model an inverted index by storing the occurrences of a term in a document. Simply storing the entire document in a Binary Large OBject (BLOB) removes the storage Problem, but most searching operations on BLOB's are limited.

Note that the term frequency (tf) or number of occurrences of a term within a document, is a specific characteristic of the APPEARS-IN relationship: thus it is stored in the table. the primary key for this relation is (DocId, Term), hence TF is entirely dependent upon this key.

For proximity searches such as "Find all documents in which the phrase' vice President exists", an additional Offset

Table INDEX_PROX

DocId	Term	Offset
1	Commercial	0
1	Vechicle	1
1	Sales	2
1	Italy	4
-	-	-
1	makers	26
..	..	..
2	Karol	2
...	...	...

attribute is required. without this, the INDEX relation indicates that vice and president co-occur in the same document, but no information as to their location is given. To indicate that vice is adjacent to president, the Offset attribute identifies the current term Offset in the document. The first term is given an Offset of zero, the second an Offset of one, and in general, the n^{th} is given an Offset of $n-1$.

Finally, Single-Valued information about term is required. The TERM relation below table Contains the idf for given term. To review, a term that occurs frequently has a low idf and is assumed to be unimportant. A term that occurs infrequently is assumed very important. Since each term has only on Idf, is a Single valued relationship which stored in a collection-wide single TERM relation.

Table ~~RANDOM~~ TERM

Term	IDF
accoding	0.9031
commercial	1.3802
Company	0.6021
dale	2.3856
diversified	2.5798
february	1.4472
Italy	1.9231
Kool	4.2768
....	....
Sales	1.0000
succeeding	2.6107
Vechicle	1.8808
Year	0.4771
..	...

Table. STOP_TERM

Term
a
an
and
...
...
the
...
..
..
..

To maintain a Syntactically fixed set of SQL queries for IR processing, and to reduce the Syntactic complexities of the queries themselves a Query relation is used

Table QUERY

Term	tf
Vechicle	1
Sales	1

Queries are simplified because the QUERY relation can be joined to INDEX to see if any of the terms in QUERY are found in INDEX.

Finally, STOP_TERM is used to indicate all of the terms that are omitted during the passing phase. This relation is not used ~~in this~~

"Find all documents that describe vehicles and sales written on 3/23/87". The keyword search covers unstructured data, while the publication date is an element of structured data

Ex: SELECT d.DocId FROM DOC d, INDEX i
WHERE i.Term IN ("vehicle", "sales") AND
d.PubDate = "3/23/87" AND
d.DocId = i.DocId;

Boolean Retrieval:-

A Boolean query is given with usual operators, AND, OR ~~AND~~ and NOT. The result set must contain all documents that satisfy the Boolean condition. They quickly allows users to specify their information need and returns all matches. Relevance ranking avoids this problem by ranking documents based on a measure of relevance between the documents and the query.

The following SQL query returns all documents that contain an arbitrary term, InputTerm

Ex: SELECT DISTINCT (i.DocId) FROM INDEX i
WHERE i.Term = InputTerm

If the text is found in a single large attribute the query can be extended to execute a subquery to obtain the document identifiers. Then the identifiers can be used to find the appropriate text in DOC.

```
Ex: SELECT d.Text FROM DOC d
    WHERE d.DocId IN
        (SELECT DISTINCT (i.DocId)
         FROM INDEX i
         WHERE i.Term = InputTerm)
```

It is natural to attempt to extend the query to allow for n terms. If the Boolean request is an OR, the extension is straightforward and does not increase the number of joins found in the query.

```
Ex: SELECT DISTINCT (i, DocId)
    FROM INDEX i
    WHERE i.Term = InputTerm1 OR
          i.Term = InputTerm2 OR
          i.Term = InputTerm3 OR
          . . .
          i.Term = InputTermN
```

Unfortunately, a Boolean AND results in a dramatically more complex query. For a query containing n input terms, the INDEX relation must be joined n times. This results in the following query.

```
Ex: SELECT a.DocId FROM INDEX a, INDEX b, - - - - INDEX n
    WHERE a.Term = InputTerm1 AND
          b.Term = InputTerm2 AND
          . . . .
          n.Term = InputTermn AND
          a.DocId = b.DocId AND
          b.DocId = c.DocId AND
          . . . .
          n-1.DocId = n.DocId:
```

The following query computes a Boolean AND using standard syntactically fixed SQL:

```
Ex: SELECT i.DocId FROM INDEX i, QUERY q
    WHERE i.Term = q.Term
    Group by i.DocId
    Having COUNT (i.Term) = (SELECT COUNT (*) FROM QUERY)
```

Proximity Searches:-

To briefly review, Proximity Searches are used in IRS to ensure that the terms in the query are found in a particular sequence or at least within a particular window of the document.

To implement proximity searches, the INDEX-PROX gives. The offset attribute indicates the relative position of each term in the document.

For the query given in our example "vice" and president occur in positions seven and eight, respectively. Document two would be retrieved if a window of two or larger was used.

```
Ex: SELECT a.DocId FROM INDEX_PROX a, INDEX_PROX b
    WHERE a.Term IN (SELECT q.Term FROM QUERY q) AND
    b.Term IN (SELECT q.Term FROM QUERY q) AND
    a.DocId = b.DocId AND
    (b.offset - a.offset) BETWEEN 0 AND (width-1)
    GROUP BY a.DocId, a.Term, a.offset
    HAVING COUNT (DISTINCT (b.Term)) =
    (SELECT COUNT (*) FROM QUERY)
```

Computing Relevance Using Unchanged SQL:-

Relevance ranking is critical for large document collection as a Boolean query frequently returns many thousands of documents. The distance between the query vector Q and the document vector Di is used to rank document.

The following dot product measure computes this distance

$$Sc(Q, Di) = \sum_{j=1}^{t} w_{qj} \times d_{ij}$$

Where w_{qj} is weight of the j^{th} term in the query q, and d_{ij} is the weight of the j^{th} term in the i^{th} document.

The term frequency is combined with the IDF. The following SQL implements a dot product query with the tf-idf weight.

```
Ex:- SELECT i.DocId, SUM(q.tf * t.idf * t.idf)
     FROM QUERY q, INDEX i, TERM t
     WHERE q.term = t.Term AND i.Term = t.Term
     GROUP BY i.DocId ORDER BY 2 DESC
```

The WHERE clause ensures that only terms found in QUERY are included in the computation. The idf is obtained from the TERM relation and is used to compute the tf-idf weight in the select-list. The ORDER BY clause ensures that the result is sorted by the similarity co-efficient.

The cosine coefficient id defined as:

$$Sc(Q, Di) = \frac{\sum_{j=1}^{t} w_{qj} d_{ij}}{\sqrt{\sum_{j=1}^{t} (d_{ij})^2 \sum_{j=1}^{t} (w_{qj})^2}}$$

To simplify the SQL, two separate relations are created: DOC_WT (DocId, Weight) and QUERY_WT (Weight), DOC-WT stores the size of the document vector for each document and QUERY_WT contains a single tuple that indicates the size of the query vector

```
Ex: INSERT INTO DOC_WT
    SELECT DocId, SQRT(SUM(i.tf * t.idf * i.tf * t.idf)
    FROM INDEX i, TERM t
    WHERE i.Term = t.Term
    GROUP BY DocId.
```

Ex! INSERT INTO QRY_WT
SELECT SQRT (SUM (q.tf * t.idf * q.tf * t.idf)
FROM QUERY q, TERM t
WHERE q.Term = t.Term.

The following query computes the cosine

Ex! SELECT i.DocId, SUM (q.tf * t.idf * i.tf * i.idf) /
(dw.weight * qw.weight)
FROM QUERY q, INDEX i, TERM t, DOC_WT dw, QRY_WT qw
WHERE q.Term = t.Term AND
i.Term = t.Term AND
i.DocId = dw.DocId
GROUP BY i.DocId, dw.weight, qw.Weight
ORDER BY 2 DESC.

The inner product is modified to use the normalized weight by joining the two new relations DOC_WT and QUERY_WT. An additional condition is added to the WHERE clause in order to obtain the weight for each document.

Semi-Structured Search using a Relational Schema:

Numerous propesty approaches exest for searching eXtensible Markup language (XMU documents. but these lack the ability to integrate with other structured or unstructured data,

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.

There are several popular XML query languages. Initial work on Xpath, touted as a basic path based query-language was submitted to the W3C (World Wild Web Consortium) In 1998 XML-QL, a query language developed at AT&T was designed to meet the requirements of a full featured XML query language set out by the W3C. It borrows many ideas from prior work on other semi-structured query languages such as XML-QL and XPath, as well as from relational query language like SQL.

Static Relational Schema to support XML-QL:-

Describe a static relational schema that supports arbitary XML schemas. This was first proposed in 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 static relational storage schema stores each unique XML path and its value from each document as a separate row in a relation. This static relational schema is capable of storing an arbitary XML docume

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) ATTRIBUTE(AttributeId, attribute)

TAG_PATH (TagId, Path) DOCUMENT (DocId, fileName)

INDEX (Id, parent, path, tagId, attrId, pos, value)

<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>

<DOC>
<DOCNO> WSJ870323-0161 </DOCNO>
<HL> Who's News: DuPont Co. </HL>
<DD> 03/23/87 </DD>
<DATELINE> DuPont Company, Wilmington, DE </DATELINE>
<TEXT>
John A. Krol was named group vice president, Agriculture Products department, of this diversified chemicals company, succeeding Dale E. Wolf, who will retire May 1. Mr Krol was formerly vice president in the Agricultural Products department.
</TEXT>
</DOC>

Storing XML Metadata:-

These tables store the metadata of the XML files TAG_NAME and TAG_PATH together store the information about tags and paths within the XML file TAG_NAME stores the name of each unique tag in the XML collection.

The TAG_ATTRIBUTE 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 preprocessing stage. Similarly, attributeId is uniquely assigned as well.

Since XML-QL allows users to specify what file(s) they wish to query, ~~many times~~ Each time a new file is indexed, it receive 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.

Table: TAG_NAME

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

Table: TAG_PATH

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

Table: ATTRIBUTE

attributeId	attribute
7	LANGUAGE

Table: DOCUMENT

docId	fileName
2	doc_0_xml
3	doc_1_XML

INDEX

The INDEX table 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 id column is a unique integer assigned to each element and attribute in a document. The parent attribute indicates the id of the tag that is the parent of the current tag.

Table: INDEX

id	Parent	Path	type	tagID	AttrId	docId	pos	value
41	0	10	E	10	1	2	0	NULL
42	41	11	E	11	1	2	0	WSJ870323-0180
43	41	12	E	12	1	2	0	Italy's Commercial
44	41	13	E	13	1	2	0	03/23/87
45	41	14	E	14	1	2	0	TURIN, Italy
46	41	15	E	15	1	2	0	Commercial-Vehicle
47	46	15	A	15	7	2	0	English
48	0	10	E	10	1	3	0	NULL
49	48	11	E	11	1	3	0	WSJ870323-0161
50	48	12	E	12	1	3	0	Who's News
51	48	13	E	13	1	3	0	03/23/87
52	48	14	E	14	1	3	0	DuPont Co.-DE
53	48	15	E	15	1	3	0	John A. Kool
54	53	15	A	15	7	3	0	English

The path corresponds to Primary key value in the TagPath. The type indicates whether the path terminates with an element or an attribute (E or A) The TagId and AttrId is a foreign key to the TagId in the TagName and Attribute tables.

The Doc-Id attribute indicates the XML document for a given row. The pos tracks the order of a given tag and is used for queries that use the index expression feature of XML-QL and indicates the position of the element relative to others under the same parent. (Starting at zero). This column stores the original ordering of the input XML for explicit use in users queries. Finally, value contains the atomic unit in XML - the value inside the lowest level tags.

A Theoretical Model of Distributed Retrieval:-

A Model for a centralized IRS and then expand that model to include a distributed IRS.

Centralized Information Retrieval System Model:-

Formally, an IRS is defined as a triple, $I = (D, R, \delta)$ where D is document collection, R is the set of queries, and $\delta_j : R_j \longrightarrow 2^{D_j}$ is mapping assigning the j^{th} query to a set of relevant documents.

Many IRS rely upon a thesaurus, in which a users query is expanded, to include synonyms of the keywords to match synonyms in a document.

To include the thesaurus in the model. it was proposed in that triple be expanded to a quadruple as:

$$I = (T, D, R, S)$$

Where "T" is set of distinct terms and the relation $\rho \subset TXT$ such that $\rho(t_1, t_2)$ implies that t_1 is a synonym of t_2. Using the synonym relation, it is possible to represent document as a set of descriptors and a set of ascriptors

Consider a document D_1, the set of descriptors d consists of all terms in D_1 such that:

- Each descriptor is unique
- No descriptor is a synonym of another descriptor

* An ascriptor is defined as a term that is a synonym of a descriptor.

*Each ascriptor must be synonymous with only one descriptor. Hence, the descriptors represent a minimal description of the document.

A generalization relation over the sets of descriptors is defined as $\boxed{r \subset d \times d}$ where $r(t_1, t_2)$ implies that t_1 is more general term that "t_2". Hence r (animal, dog) is an example of a valid generalization.

* The generalization relation assumes that it is possible to construct a hierarchical knowledge base of all pairs of descriptors. Construction of such knowledge bases was attempted both automatically and manually.

The idea being that additional synonyms do not add to the semantic value of the document. The generalization relation is used to allow for the processing of a query that states "List all animals"

The generalization can then be used to define a partial ordering of documents. Let the partial ordering be denoted by "$\preceq$" and let $t(d_i)$ indicate that list of descriptors for document d_i.

Partial ordering $\preceq$ is defined as:

$$t(d_1) \preceq t(d_2) \Leftrightarrow (\forall t' \in t(d_1))(\exists t'' \in t(d_2))(r(t', t''))$$

A document d_1 whose descriptors are all generalizations of the descriptors found in d_2 will have the ordering $d_1 \preceq d_2$.

For example a document with the term animal and person will precede a document with terms dog and John. Note that this is partial ordering because two documents with terms that have no relationship between any pairs of terms will

To be inclusive, the documents that correspond to a general query q_1 must include (be a superset of) all documents that corrospond to the documents that correspond to a more specific query q_2. Where $q_1 \preceq q_2$ Formally

$$(q_1, q_2 \in Q) \wedge (q_1 \preceq q_2) \longrightarrow (\delta(q_1) \supset \delta(q_2))$$

Distributed Information Retrieval System Model! -

The centralized IRS can be partioned into n local IRS $S_1, S_2, \ldots S_n$ Each System S_i is of the form

$S_i = (T_j, D_j, R_j \cdot \delta_j)$ where

T_j : is the thesaurus

D_j: is the document collection.

R_j: Set of queries.

$\delta_j : R_j \longrightarrow 2^{D_j}$ maps the queries to documents

By taking the union of the local sites, it is possible to define the distributed IRS as:

$$S = (T, D, R \cdot \delta)$$

where

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

$$S_j = S \cap (T_j \times T_j), \quad R_j = R \cap (d_j \times d_j)$$

This states that the global δ thesaurus can be constructed from the local thesauri. and the queries at the sites j will only include descriptors at site j.

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

The Document collection, D, can be constructed by combining the document collection at each site

$$R \supset \bigcup_{j=1}^{n} R_j \preceq_j = \preceq \cap (R_i \times R_j)$$

The queries can be obtained by combining the queries at each local site. The partial ordering defined at site j will only pertain to queries at site j.

$$(\forall r \in R)\ (\delta(r) = d : d \in D \cap r \preceq t(d))$$

The hierarchy represented by r is partioned among the different sites. A query sent to the originating site would be sent to each local site and a local query would be performed. The local responses are sent to the originating site where they are combined into a final result set.

The model allows for this methodology if the local sites satisfy the criteria of being a subsystem of the IRS.

$S_1 = (T_1, D_1, R_1, \delta_1)$ is a subsystem of $S_2 = (T_2, D_2, R_2, \delta_2)$ if:

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

The thesaurus of T_1 is superset of T_2

$$D_1 \supset D_2$$

The document collection at site S_1 contains the collection D_2.

$$R_1 \in R_2 \wedge \preceq_1 = \preceq_2 \cap (R_1 \times R_2)$$

The queries at site S_1 contain those found in S_2

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

* The document collection returned by queries in S_1 will include all documents returned by queries in S_2.

Example:

r(People, Harold), r(people, Herbert), r(People, Mary)

and the second animal hierarchy:

r(animal, cat), r(animal, dog), r(cat, black-cat),

r(cat, cheshire), r(dog, doberman), r(dog, poodle)

Assume that the hierarchy is split into sites S_1 and S_2

S_1:

r(people, Harold), r(people, Mary)

r(animal, cat), r(animal, dog), r(dog, doberman), r(dog, poodle)

S_2:

r(people, Herbert), r(people, Harold)

r(animal, cat), r(animal, doberman), r(cat, cheshire), r(cat, black-cat)

Consider a set of documents with the following descriptors:

D_1 = (Mary, Harold, Herbert)

D_2 = (Herbert, dog)

D_3 = (people, dog)

D_4 = (Mary, cheshire)

D_5 = (Mary, dog)

D_6 = (Herbert, black-cat, doberman)

D_7 = (Herbert, doberman)

A query of the most general terms (people, animal) should return all documents 2 through 7. However, the hierarchy given above as S_1 will only retrieve documents D_3 and D_5, and S_2 will only retrieve documents D_6 and D_7. Hence, documents D_2 and D_4 are missing from the final result if the local results sets simply concatenated.

Web Search:-

The Search tools that access web pages via the Internet are prime examples of the implementation of many of the algorithms and heuristics discussed. These systems are, by nature, distributed in that they access data stored on Web Servers around the world. Most of these systems have a centralized index, but all of them store pointers in the form of hypertext links to various web servers.

These systems service tens of millions of user queries a day, and all of them index Several Terabytes of web pages. The Sixteen different web search engines are listed at www.Searchenginewatch.com while www.Searchengineguide.com list over 2500 ~~Specili~~ Specialized Search engines.

~~Evalul~~ Evaluation of Web Search Engines:-

In traditional information retrieval environments, individual systems are evaluated using standard queries and data. In the web environment, such evaluation conditions are unavailable. Manual evaluations on any grand scale are virtually impossible due to the vast size and dynamic nature of the web. To automatically evaluate web search engines, a method using online taxonomies that were created as part of Open Directory Project (ODP). Online directories were used as known relevant items for a query. If a query matches either the title of the item stored or the directory file name containing a known item then it is considered a match.

High Precision Search:-

Another concern in evaluating web search engines is the different measures of success as compared to traditional environment. Traditionally, precision and recall measures are the main evaluation metrics, while response time and space requirements are likely addressed. However, in the Web environment, response time is critical. Furthermore, recall estimation is very difficult, and precision is of limited concern since most users never access any links that appear beyond the first answer screen. Thus web search engine developers focus on guaranteeing that the first result screen is generated quickly, is highly accurate, and that no severe accuracy mismatch exists

Query log Analysis

Although similar and relying on much the same techniques as used in traditional IRS domains, the web environment provides for many new opportunities to revisit old issue particularly in terms of performance and accuracy optimizations and evaluation measures of search accuracy. Recently, an hourly analysis of a very large topically categorized web query log was published. Using the results presented, it is possible to generate many system optimizations.

for example, as indicated in the findings presented, user requests patterns repeat according to the time of day

and day of week. Thus, depending on the time of day and day of week, it is possible to pre-cache likely web pages in anticipation of a set of user request. Thus, page access delays are reduced increasing system throughput. Furthermore, in terms of accuracy optimization, it is likewise possible to adjust the ranking measures to better tune for certain anticipated user subject request.

Page Rank:-

This PageRank algorithm described in (Brin 2005). Page Rank is at the heart of the popular Web search engine, Google. Essentially, the PageRank algorithm uses incoming and outgoing links to adjust the score of a web page with respect to its popularity, independent of the user's query.

Hence, if a traditional retrieval strategy might have previously ranked two documents equal, the PageRank algorithm will boost the similarity measure for a popular document. Here, popular is defined as having a number of other web pages link to the document. This algorithm works well on web pages, but has no bearing on document that do not have any hyperlink.

The calculation of PageRank for page A over all pages linking to it $D_1 \cdots D_n$ is defined as follows.

$$PageRank(A) = (1-d) + d \sum_{D_1 D_2} \frac{PageRank(D_i)}{C(D_i)}$$

Where $C(D_i)$ the number of links out from page D_i and 'd' is a dampening factor from 0–1.

This dampening factor serves to give some non-zero PageRank to pages that have no links to them. The calculation is repeated using the previously calculated scores until the new scores do not change significantly.