G.PULLAIAH COLLEGE OF ENGINEERING AND TECHNOLOGY,KURNOOL
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

IV B.TECH I SEM (R13)

INFORMATION RETRIEVAL SYSTEMS

UNIT-2

UNIT-II

Retrieval Utilities

Utilities improve the results of a retrieval strategy. Most utilities add or remove terms from the initial query in an attempt to refine the query. Others simply refine the focus of the query by using subdocuments or passages instead of whole documents. The key is that each of these utilities (although rarely presented as such) are plug-and-play utilities that operate with any arbitrary retrieval strategy.

The utilities identified are:

Relevance Feedback-The top documents found by an initial query are identified as relevant. These documents are then examined. They may be deemed relevant either by manual intervention or by an assumption that the top n documents are relevant. Various techniques are used to rank the terms. The top t terms from these documents are then added back to the original query.

Clustering-Documents or terms are clustered into groups either automatically or manually. The query is only matched against clusters that are deemed to contain relevant information. This limits the search space. The goal is to avoid non-relevant documents before the search even begins

N-grams-The query is partitioned into n-grams (overlapping or non-overlapping sequences of n characters). These are used to match queries with the document. The goal is to obtain a "fuzzier" match that would be resilient to misspellings or optical character recognition (OCR) errors. Also, n-grams are language independent.

Thesauri-Thesauri are automatically generated from text or by manual methods. The key is not only to generate the thesaurus, but to use it to expand either queries or documents to improve retrieval.

Regression Analysis- Statistical techniques are used to identify parameters that describe characteristics of a match to a relevant document. These can then be used with a regression analysis to identify the exact parameters that refine the similarity measure.

2.1 Relevance Feedback

A popular information retrieval utility is relevance feedback. The basic premise is to implement retrieval in multiple passes. The user refines the query in each pass based on results of previous queries. Typically, the user indicates which of the documents presented in response to an initial query are relevant, and new terms are added to the query based on this selection. Additionally, existing terms in the query can be re-weighted based on user feedback. This process is illustrated in Figure.

An alternative is to avoid asking the user anything at all and to simply assume the top ranked documents are relevant. Using either manual (where the user is asked) or automatic (where it is assumed the top documents are relevant) feedback, the initial query is modified, and the new query is re-executed.

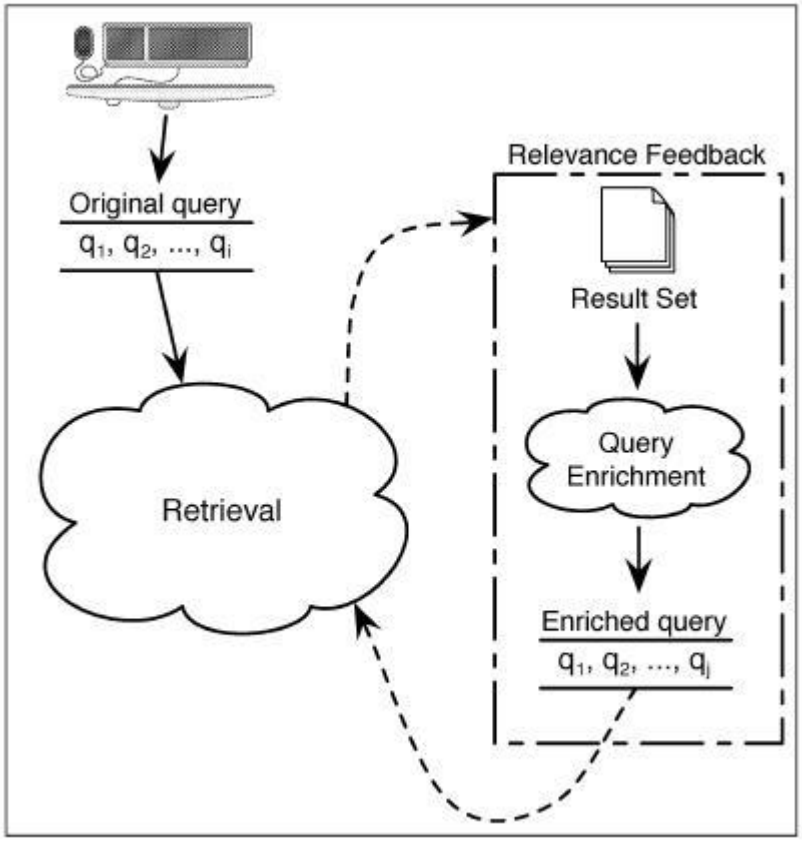


Fig: Relevance feedback process

2.1.1 Relevance Feedback in the Vector Space Model

Rocchio, in his initial paper, started the discussion of relevance feedback . Interestingly, his basic approach has remained fundamentally unchanged. Rocchio's approach used the vector space model to rank documents. The query is represented by a vector Q, each document is represented by a vector Di, and a measure of relevance between the query and the document vector is computed as SC(Q, Di), where SC is the similarity coefficient. As discussed the SC is computed as an inner product of the document and query vector or the cosine of the angle between the two vectors. The basic assumption is that the user has issued a query Q and retrieved a set of documents. The user is then asked whether or not the documents are relevant. After the user responds, the set R contains the nl relevant document vectors, and the set S contains the n2 non-

relevant document vectors. Rocchio builds the new query Q' from the old query Q using the equation given below:

$$Q' = Q + \frac{1}{n_1}\sum_{i=1}^{n_1} R_i - \frac{1}{n_2}\sum_{i=1}^{n_2} S_i$$

Ri and Si are individual components of R and S, respectively.

The document vectors from the relevant documents are added to the initial query vector, and the vectors from the non-relevant documents are subtracted. If all documents are relevant, the third term does not appear. To ensure that the new information does not completely override the original query, all vector modifications are normalized by the number of relevant and non-relevant documents. The process can be repeated such that Qi+1 is derived from Qi for as many iterations as desired. The idea is that the relevant documents have terms matching those in the original query. The weights corresponding to these terms are increased by adding the relevant document vector. Terms in the query that are in the nonrelevant documents have their weights decreased. Also, terms that are not in the original query (had an initial component value of zero) are now added to the original query. In addition to using values n1 and n2, it is possible to use arbitrary weights.

The equation now becomes:

$$Q' = \alpha Q + \beta \sum_{i=1}^{n_1} \frac{R_i}{n_1} - \gamma \sum_{i=1}^{n_2} \frac{S_i}{n_2}$$

Not all of the relevant or non-relevant documents must be used. Adding thresholds na and nb to indicate the thresholds for relevant and non-relevant vectors results in:

$$Q' = \alpha Q + \beta \sum_{i=1}^{min(n_a,n_1)} \frac{R_i}{n_1} - \gamma \sum_{i=1}^{min(n_b,n_2)} \frac{S_i}{n_2}$$

The weights a, ,8, and, are referred to as Rocchio weights and are frequently mentioned in the annual proceedings of TREe. The optimal values were experimentally obtained, but it is considered common today to drop the use of nonrelevant documents (assign zero to ,) and only use the relevant documents. This basic theme was used by Ide in follow-up research to Rocchio where the following equation was defined:

$$Q' = \alpha Q + \beta \sum_{i=1}^{n_1} R_i - S_1$$

Another intresting case when q retrieves only non-relevant documents then an arbitrary weight should be added to most frequently occurring term.This increases weight of term .By increasing weight of term it yields some relevant documents.This approach is applied only in manual relevance feedback and not in automatic relevance feedback.

2.1.2 Relevance Feedback in the Probabilistic Model

In probabilistic model the terms in the document are treated as evidence that a document is relevant to a query. Given the assumption of term independence, the probability that a document is relevant is computed as a product of the probabilities of each term in the document matching a term in the query. The probabilistic model is well suited for relevance feedback because it is necessary to know how many relevant documents exist for a query to compute the term weights. Typically, the native probabilistic model requires some training data for which relevance information is known. Once the term weights are computed, they are applied to another collection. Relevance feedback does not require training data. Viewed as simply a utility instead of a retrieval strategy, probabilistic relevance feedback "plugs in" to any existing retrieval strategy. The initial query is executed using an arbitrary retrieval strategy and then the relevance information obtained during the feedback stage is incorporated.

For example, the basic weight used in the probabilistic retrieval strategy is:

$$w_i = \log \frac{\frac{r_i}{R-r_i}}{\frac{n_i-r_i}{(N-n_i)-(R-r_i)}}$$

where:

W_i -weight of term i in a particular query

R -number of documents that are relevant to the query

N -number of documents in the collection

r_I - number of relevant documents that contain term i

n_i -number of documents that contain term i

R and r cannot be known at the time of the initial query unless training data with relevance information is available

2.1.2.1 Initial Estimates

The initial estimates for the use of relevance feedback using the probabilistic model have varied widely. Some approaches simply sum the idf as an initial first estimate. Wu and Salton proposed an interesting extension which requires the use of training data. For a given term t, it is necessary

to know how many documents are relevant to term t for other queries. The following equation estimates the value of r i prior to doing a retrieval:

$$ri = a + b\log f$$

where f is the frequency of the term across the entire document collection.

After obtaining a few sample points, values for a and b can be obtained by a least squares curve fitting process. Once this is done, the value for ri can be estimated given a value of f, and using the value of ri, an estimate for an initial weight (IW) is obtained. The initial weights are then combined to compute a similarity coefficient. In the paper [Wu and Salton, 1981] it was concluded (using very small collections) that idf was far less computationally expensive, and that the IW resulted in slightly worse precision and recall.

2.1.2.2 Computing New Query Weights

For,query Q,Document D and t terms in D,D_i is binary.If the term is present then place 1 otherwise place 0.

$$SC(Q, D_i) = \sum_{i=1}^{t} d_i \log \frac{p_i(1-u_i)}{u_i(1-p_i)} + K$$

Where k is constant.

$$\hat{p_i} = \frac{r_i+0.5}{R+1} \text{ and } \hat{u_i} = \frac{n_i-r_i+0.5}{N-R+1}.$$

After substituting we get

$$\frac{\frac{r_i+0.5}{R+1}}{\frac{n_i-r_i+0.5}{N-R+1}}$$

Using relevance feedback, a query is initially submitted and some relevant documents might be found in the initial answer set. The top documents are now examined by the user and values for r i and R can be more accurately estimated (the values for ni and N are known prior to any retrieval). Once this is done, new weights are computed and the query is executed again. Wu and Salton tested four variations of composing the new query:

1. Generate the new query using weights computed after the first retrieval.

2. Generate the new query, but combine the old weights with the new. Wu suggested that the weights could be combined as:

$$Q' = \frac{1-\beta}{Q} + (1-\beta)(T)$$

Where

Q-old weights

T-weights computed during first pass

β-scaling factor that inducates importance of initial weights

The ratio of relevant documents retrieved to relevant documents available collection-wide is used for this value $(\beta = \frac{r_i}{R})$.

A query that retrieves many relevant documents should use the new weights more heavily than a query that retrieves only a few relevant documents.

3. Expand the query by combining all the terms in the original query with all the terms found in the relevant documents. The weights for the new query are used as in step one for all of the old terms (those that existed in the original query and in the relevant documents). For terms that occurred in the original query, but not in any documents retrieved in the initial phase, their weights are not changed. This is a fundamental difference from the work done by

4. Expand the query using a combination of the initial weight and the new weight. This is similar to variation number two above. Assuming ql to qm are the weights found in the m components of the original query, and m - n new

terms are found after the initial pass, we have the following:

$$Q' = \frac{1-\beta}{Q}(q_1, q_2, \ldots, q_m, 0, 0, \ldots, 0) + \beta(q_1, q_2, \ldots, q_m, q_{m+1}, \ldots, q_{m+n})$$

$$p_i = \frac{r_i + \frac{n_i}{N}}{R+1}$$
$$u_i = \frac{n_i - r_i + \frac{n_i}{N}}{N-R+1}$$

Here the key element of the idf is used as the adjustment factor instead of the crude 0.5 assumption.

2.1.2.3 Partial Query Expansion

The initial work done by Wu and Salton in 1981 either used the original query and reweighted it or added all of the terms in the initial result set to the query and computed the weights for them. The idea of using only a selection of the terms found in the top documents was presented. Here the top ten documents were retrieved. Some of these documents were manually identified as relevant. The question then arises as to which terms from these documents should be used to expand the initial query. Harman sorted the terms based on six different sort orders and, once the terms were sorted, chose the top twenty terms. The sort order had a large impact on effectiveness. Six different sort orders were tested on the small Cranfield collection.

In many of the sort orders a noise measure, n, is used. This measure, for the k_{th} term is computed as:

$$n_k = \sum_{i=1} N \frac{tf_{ik}}{f_k} \log_2 f_k tf_{ik}$$

t_{fik} -number of occurrences of term i in document k

f_k -number of occurrences of term k in the collection

N -number of terms in the collection

This noise value increases for terms that occur infrequently in many documents, but frequently across the collection. A small value for noise occurs if a term occurs frequently in the collection. It is similar to the idf, but the frequency within individual documents is incorporated.

Additional variables used for sort orders are:

P_k number of documents in the relevant set that contain term k

rt f_k number of occurrences of term k in the relevant set

A modified noise measure, rn_k. is defined as the noise within the relevant set.

This is computed as:

$$rn_k = \sum_{i=1} p_k \frac{tf_{ik}}{f_k} \log_2 f_k tf_{ik}$$

Various combinations of rnk, nk. and Pk were used to sort the top terms. The six sort orders tested were:

- n_k
- P_k
- r_{nk}
- n_k x rtf_k
- n_k x f_k x P_k
- n_k x f_k

Six additional sort orders were tested.

The sorts tested were:

- $\frac{(RT_j)(df_i)}{N}$

where RTj - total number of documents retrieved for query j,

df_i - document frequency or number of documents in the collection that contain term

i, N - number of documents in the collection.

This gives additional weight to terms that appear in multiple documents of the initial answer set.

- $\frac{r_{ij}}{R} - \frac{df_i}{N}$

r_{ij} - number of retrieved relevant documents for query j that have term

i. R_j-number of retrieved relevant documents for query j.

This gives additional weight to terms that occur in many relevant documents and which occur infrequently across the entire document collection.

- $W_{ij} = \log_2 \frac{p_{ij}(1-q_{ij})}{(1-p_{ij})q_{ij}}$

W_{ij} - term weight for term i in query j.

P_{ij}-The probability that term i is assigned within the set of relevant documents to query j

q_{ij} -The probability that term i is assigned within the set of non-relevant documents for query j

is. These are computed as:

$$p_{ij} = \frac{r_{ij} + 0.5}{R_j + 1.0} \qquad q_{ij} = \frac{df_i - r_i + 0.5}{N - R_j + 1.0}$$

- $idf_j(p_{ij} - q_{ij})$

where the theoretical foundation is based on the presumption that the term i's importance is computed as the amount that it will increase the difference between the average score of a relevant document and the average score of a nonrelevant document. The means of identifying a term weight are not specified in this work, so for this sort order, idf_j is used.

- $W_{ij}(p_{ij} - q_{ij})$

where the term weight is computed as given above.

- $\log(RTF_i + 1)(p_{ij} - q_{ij})$

where RT F_i is the number of occurrences of term i in the retrieved relevant documents. Essentially, sort three was found to be superior to sorts four, five, and six, but there was little difference in the use of the various sort techniques. Sorts one and two were not as effective.

2.1.2.4 Number of Feedback Iterations

The number of iterations needed for successful relevance feedback was initially tested in 1971 by Salton. His 1990 work with 72 variations on relevance feedback assumed that only one iteration of relevance feedback was used. Harman investigated the effect of using multiple iterations of relevance feedback . The top ten documents were initially retrieved. A count of the number of relevant documents was obtained, and a new set of ten documents was then retrieved. The process continued for six iterations. Searching terminates if no relevant documents are found in a given iteration. Three variations of updating term weights across iterations were used based on whether or not the counting of relevant documents found was static or cumulative. Each iteration used the basic strategy of retrieving the top ten documents, identifying the top 20 terms, and reweighting the terms.

The three variations tested were:

• Cumulative count-counts relevant documents and term frequencies within relevant documents. It accumulates across iterations

• Reset count-resets the number of relevant documents and term frequencies within relevant documents are reset after each iteration

• Reset count, single iteration term---counts are reset and the query is reset such that it only contains terms from the current iteration

In each case, the number of new relevant documents found increased with each iteration. However, most relevant documents were found in the first two iterations.On average, iterations 3, 4, 5, and 6 routinely found less than one new relevant document per query.

2.1.2.5 User Interaction

The initial work in relevance feedback assumed the user would be asked to determine which documents were relevant to the query. Subsequent work assumes the top n documents are relevant and simply uses these documents. An interesting user study, done by Spink, looked at the question of using the top documents to suggest terms for query expansion, but giving the user the ability to pick and choose which terms to add . Users were also studied to determine how much relevance feedback is used to add terms as compared to other sources. The alternative sources for query terms were:

• Original written query

• User interaction-discussions with an expert research user or "intermediary" prior to the search to identify good terms for the query

• Intermediary-suggestion by expert users during the search

• Thesaurus

• Relevance feedback-selection of terms could be selected by either the user or the expert intermediary

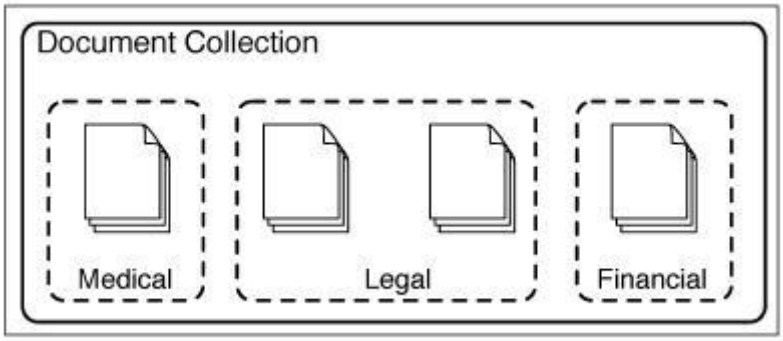
Users chose forty-eight terms (eleven percent) of their search terms (over forty queries) from relevance feedback. Of these, the end-user chose fifteen and the expert chose thirty-three. This indicates a more advanced user is more likely to take advantage of the opportunity to use relevance feedback.

Additionally, the study identified which section of documents users found terms for relevance feedback. Some eighty-five percent of the relevance feedback terms came from the title or the descriptor fields in the documents, and only two terms came from the abstract of the document. This study concluded that new systems should focus on using only the title and descriptor elements of documents for sources of terms during the relevance feedback stages.

2. 2 Clustering

Document clustering attempts to group documents by content to reduce the search space required to respond to a query. For example, a document collection that contains both medical and legal documents might be clustered such that all medical documents are placed into one cluster, and all

legal documents are assigned to a legal cluster. A query over legal material might then be directed (either automatically or manually) to the legal document cluster.



Document clustering

Several clustering algorithms have been proposed. In many cases, the evaluation of clustering algorithms has been challenging because it is difficult to automatically point a query at a document cluster. Viewing document clustering as a utility to assist in ad hoc document retrieval, we now focus on clustering algorithms and examine the potential uses of these algorithms in improving precision and recall of ad hoc and manual query processing. Another factor that limits the widespread use of clustering algorithms is their computational complexity. Many algorithms begin with a matrix that contains the similarity of each document with every other document. For a 1,000,000 document collection, this matrix has $\frac{1{,}000{,}000^2}{2}$ different elements. Each of these pair-wise similarity calculations is computationally expensive due to the same factors found in the traditional retrieval problem. Initial work on a Digital Array Processor (DAP) was done to improve run-time performance of clustering algorithms by using parallel processing Subsequently, these algorithms were implemented on a parallel machine with a torus interconnection network. Clusters are formed with either a top-down or bottom-up process. In a top-down approach, the entire collection is viewed as a single cluster and is partitioned into smaller and smaller clusters. The bottom-up approach starts with each document being placed into a separate cluster of size one and these clusters are then glued to one another to form larger and larger clusters. The bottom up approach is referred to as hierarchical agglomerative because the result of the clustering is a hierarchy (as clusters are pieced together, a hierarchy emerges). Other clustering algorithms, such as the popular K-Means algorithm, use an iterative process that begins with random cluster centroids and iteratively adjusts them until some termination condition is met. Some studies have found that hierarchical algorithms, particularly those that use group-average cluster merging schemes, produce better clusters because of their complete

document-to-document comparisons . More recent work has indicated that this may not be true across all metrics and that some combination of hierarchical and iterative algorithms yields improved effectiveness .As these studies use a variety of different experiments, employ different metrics and (often very small) document collections, it is difficult to conclude which clustering method is definitively superior.

2.2.1 Result Set Clustering

Clustering was used as a utility to assist relevance feedback.In those cases only the results of a query were clustered (a much smaller document set), and in the relevance feedback process, by only new terms from large clusters were selected.Recently, Web search results were clustered based on significant phrases in the result set . First, documents in a result set are parsed, and two term phrases are identified. Characteristics about these phrases are then used as input to a model built by various learning algorithms (e.g.; linear regression, logistic regression, and support vector regression are used in this work). Once the most significant phrases are identified they are used to build clusters. A cluster is initially identified as the set of documents that contains one of the most significant phrases. For example, if a significant phrase contained the phrase "New York", all documents that contain this phrase would be initially placed into a cluster. Finally, these initial clusters are merged based on document-document similarity.

2.2.2 Hierarchical Agglomerative Clustering

First the N x N document similarity matrix is formed. Each document is placed into its own cluster. The following two steps are repeated until only one cluster exists.

• The two clusters that have the highest similarity are found.

• These two clusters are combined, and the similarity between the newly formed cluster and the remaining clusters recomputed.

As the larger cluster is formed, the clusters that merged together are tracked and form a hierarchy.

Assume documents A, B, C, D, and E exist and a document-document similarity matrix exists. At this point, each document is in a cluster by itself:

{{A} {B} {C} {D} {E}}

We now assume the highest similarity is between document A and document B. So the contents of the clusters become:

{{A,B} {C} {D} {E}}

After repeated iterations of this algorithm, eventually there will only be a single cluster that consists of {A,B,C,D,E}. However, the history of the formation of this cluster will be known.

The node {AB} will be a parent of nodes {A} and {B} in the hierarchy that is formed by clustering since both A and B were merged to form the cluster {AB}.

Hierarchical agglomerative algorithms differ based on how {A} is combined with {B} in the first step. Once it is combined, a new similarity measure is computed that indicates the similarity of a document to the newly formed cluster {AB}

2.2.2.1 Single Link Clustering

The similarity between two clusters is computed as the maximum similarity between any two documents in the two clusters, each initially from a separate cluster. Hence, if eight documents are in cluster A and ten are in cluster B, we compute the similarity of A to B as the maximum similarity between any of the eight documents in A and the ten documents in B.

2.2.2.2 Complete Linkage

Inter-cluster similarity is computed as the minimum of the similarity between any documents in the two clusters such that one document is from each cluster.

2.2.2.3 Group Average

Each cluster member has a greater average similarity to the remaining members of that cluster than to any other cluster. As a node is considered for a cluster its average similarity to all nodes in that cluster is computed. It is placed in the cluster as long as its average similarity is higher than its average similarity for any other cluster.

2.2.2.4 Ward's Method

Clusters are joined so that their merger minimizes the increase in the sum of the distances from each individual document to the centroid of the cluster containing it. The centroid is defined as the average vector in the vector space. If a vector represents the i^{th} document,Di =< tl, t2, ... , tn >, the centroid C is written as C =< CI, C2, ... , Cn >.The j^{th} element of the centroid vector is computed as the average of all of the j^{th} elements of the document vectors:

$$c_j = \frac{\sum_{i=1}^{n} t_{ij}}{n}$$

Hence, if cluster A merged with either cluster B or cluster C, the centroids for the potential cluster AB and AC are computed as well as the maximum distance of any document to the centroid. The cluster with the lowest maximum is used.

2.2.2.5 Analysis of Hierarchical Clustering Algorithms

Ward's method typically took the longest to implement these algorithms, with single link and complete linkage being somewhat similar in run-time .A summary of several different studies on

clustering is given in . Clusters in single link clustering tend to be fairly broad in nature and provide lower effectiveness. Choosing the best cluster as the source of relevant documents resulted in very close effectiveness results for complete link, Ward's, and group average clustering. A consistent drop in effectiveness for single link clustering was noted.

2.2.3 Clustering Without a Precomputed Matrix

Other approaches exist in which the N x N similarity matrix indicates that the similarity between each document and every other document is not required.These approaches are dependent upon the order in which the input text is received, and do not produce the same result for the same set of input files.

2.2.3.1 One-Pass Clustering

One approach uses a single pass through the document collection. The first document is assumed to be in a cluster of size one. A new document is read as input, and the similarity between the new document and all existing clusters is computed. The similarity is computed as the distance between the new doc ument and the centroid of the existing clusters. The document is then placed into the closest cluster, as long as it exceeds some threshold of closeness. This approach is very dependent on the order of the input. An input sequence of documents 1,2, ... ,10 can result in very different clusters than any other of the (10! - 1) possible orderings.

Since resulting clusters can be too large, it may be necessary to split them into smaller clusters. Also, clusters that are too small may be merged into larger clusters.

2.2.3.2 Rocchio Clustering

Rocchio developed a clustering algorithm, in which all documents are scanned and defined as either clustered or loose. An unclustered document is tested as a potential center of a cluster by examining the density of the document and thereby requiring that n_1 documents have a similarity coefficient of at least P_1 and at least n_2 documents have a correlation of P_2. The similarity coefficient Rocchio most typically used was the cosine coefficient. If this is the case, the new document is viewed as the center of the cluster and the old documents in the cluster are checked to ensure they are close enough to this new center to stay in the cluster. The new document is then marked as clustered If a document is outside of the threshold, its status may change from clustered to loose. After processing all documents, some remain loose. These are added to the cluster whose centroid the document is closest to (revert to the single pass approach).

Several parameters for this algorithm were described . These included:

- Minimum and maximum documents per cluster
- Lower bound on the correlation between an item and a cluster below which an item will not be placed in the cluster. This is a threshold that would be used in the final cleanup phase of unclustered items.
- Density test parameters(n_1, n_2, P_1, P_2)

• Similarity coefficient

2.2.3.3 K-Means

The popular K-means algorithm is a partitioning algorithm that iteratively moves k centroids until a termination condition is met. Typically, these centroids are initially chosen at random. Documents are assigned to the cluster corresponding to the nearest centroid. Each centroid is then recomputed. The
algorithm stops when the centroids move so slightly that they fall below a user-defined threshold or a required information gain is achieved for a given iteration.

2.2.3.4 Buckshot Clustering

Buckshot clustering is a clustering algorithm which runs in O(kn) time where k is the number of clusters that are generated and n is the number of documents. For applications where the number of desired clusters is small, the clustering time is close to 0 (n) which is a clear improvement over the 0 (n^2) alternatives that require a document -document similarity matrix.

Buckshot clustering works by choosing a random sample of √kn documents.These √kn documents are then clustered by a hierarchical clustering algorithm (anyone will do). Using this approach, k clusters can be identified from the cluster hierarchy. The hierarchical clustering algorithms all require a DOC-DOC similarity matrix, so this step will require $O(\sqrt{kn}^2) = O(kn)$ time. Once the k centers are found, the remaining documents are then scanned and assigned to one of the k centers based on the similarity coefficient between the incoming document and each of the k centers. The entire algorithm requires on the order of 0 (kn) time, as 0 (kn) is required to obtain the centers and O(kn) is required to scan the document collection and assign each document to one of the centers. Note that buckshot clustering can result in different clusters with each running because a different random set of documents can be chosen to find the initial k centers.

2.2.3.5 Non-negative Matrix Factorization

A more recent clustering algorithm uses non-negative matrix factorization (NMF). This provides a latent semantic space where each axis represents the topic of each cluster. Documents are represented as a summation of each axis and are assigned to the cluster associated with the axis for which they have the greatest projection value .

2.2.4 Querying Hierarchically Clustered Collections

Once the hierarchy is generated, it is necessary to determine which portion of the hierarchy should be searched. A top-down search starts at the root of the tree and compares the query

vector to the centroid for each subtree. The subtree with the greatest similarity is then searched. The process continues until a leaf is found or the cluster size is smaller than a predetermined threshold. A bottom-up search starts with the leaves and moves upwards. Early work showed that starting with leaves, which contained small clusters, was better than starting with large clusters. Subsequently three different bottom-up procedures were studied :

• Assume a relevant document is available, and start with the cluster that contains that document.

• Assume no relevant document is available. Implement a standard vector space query, and assume the top-ranked document is relevant. Start with the cluster that contains the top-ranked document.

• Start with the bottom level cluster whose centroid is closest to the query.

Once the leaf or bottom-level cluster is identified, all of its parent clusters are added to the answer set until some threshold for the size of the answer set is obtained.

These three bottom-up procedures were compared to a simpler approach in which only the bottom is used. The bottom-level cluster centroids are compared to the query and the answer set is obtained by expanding the top n clusters.

2.2.5 Efficiency Issues

Although the focus of this chapter is on effectiveness, the limited use of clustering algorithms compels us to briefly mention efficiency concerns. Many algorithms begin with a matrix that contains the similarity of each document with every other document. For a 1,000,000 document collection, this matrix has $\frac{1{,}000{,}000^2}{2}$ elements. Algorithms designed to improve the efficiency of clustering are given in , but at present, no TREC participant has clustered the entire document collection.

2.2.5.1 Parallel Document Clustering

Another means of improving run-time performance of clustering algorithms is to implement them on a parallel processor. Initial work on a Digital Array Processor (DAP) was done to improve the run-time of clustering algorithms by using parallel processing. These algorithms were implemented on a parallel machine with a torus interconnection network . A parallel version of the Buckshot clustering algorithm was developed that showed near-linear speedup on a network of sixteen workstations. This enables Buckshot to scale to significantly larger collections and provides a parallel hierarchical agglomerative algorithm There exists some other work specifically focused on parallel hierarchical clustering , but these algorithms often have large computational overhead or have not been evaluated for document clustering. Some work was done in developing parallel algorithms for hierarchical document clustering, however these algorithms were developed for several types of specialized interconnection networks, and it is

unclear whether they are applicable to the simple bus connection that is common for many current parallel architectures.

Additional proposals use clustering as a utility to assist relevance feedback . Only the results of a query are clustered (a much smaller document set), and relevance feedback proceeds by only obtaining new terms from large clusters.

2.2.5.2 Clustering with Truncated Document Vectors

The most expensive step in the clustering process occurs when the distance between the new document and all existing clusters is computed. This is typically done by computing the centroid of each cluster and measuring the cosine of the angle between the new document vector and the centroid of each cluster.

Later, it was shown that clustering can be done with vectors that use only a few representative terms from a document .

One means of reducing the size of the document vector is to use Latent Semantic Indexing to identify the most important components.Another means is to simply truncate the vector by removing those terms with a weight below a given threshold. No significant difference in effectiveness was found for a baseline of no truncation, or using latent semantic indexing with twenty, fifty, and one hundred and fifty terms or simple truncation with fifty terms.

2.4 N-grams

Term-based search techniques typically use an inverted index or a scan of the text . Additionally, queries that are based on exact matches with terms in a document perform poorly against corrupted documents. This occurs regardless of the source of the errors-either OCR (optical character recognition)

errors or those due to misspelling. To provide resilience to noise, n-grams were proposed. The premise is to decompose terms into word fragments of size n, then design matching algorithms that use these fragments to determine whether or not a match exists.

N-grams have also been used for detection and correction of spelling errors and text compression. A survey of automatic correction techniques is found in . Additionally, n-grams were used to determine the authorship of documents.

2.4.1 D' Amore and Mah

Initial information retrieval research focused on n-grams as presented in. The motivation behind their work was the fact that it is difficult to develop mathematical models for terms since the potential for a term that has not been seen before is infinite. With n-grams, only a fixed number of n-grams can exist for a given value of n. A mathematical model was developed to estimate the noise in indexing and to determine appropriate document similarity measures.

D' Amore and Mah's method replaces terms with n-grams in the vector space model. The only remaining issue is computing the weights for each n-gram. Instead of simply using n-gram frequencies, a scaling method is used to normalize the length of the document. D' Amore and Mah's contention was that a large document contains more n-grams than a small document, so it should be scaled based on its length.

To compute the weights for a given n-gram, D' Amore and Mah estimated the number of occurrences of an n-gram in a document. The first simplifying assumption was that n-grams occur with equal likelihood and follow a binomial distribution. Hence, it was no more likely for n-gram "ABC" to occur than "DEF." The Zipfian distribution that is widely accepted for terms is not true for n-grams. D' Amore and Mah noted that n-grams are not equally likely to occur, but the removal of frequently occurring terms from the document collection resulted in n-grams that follow a more binomial distribution than the terms.

D' Amore and Mah computed the expected number of occurrences of an ngram in a particular document. This is the product of the number of n-grams in the document (the document length) and the probability that the n-gram occurs. The n-gram's probability of occurrence is computed as the ratio of

its number of occurrences to the total number of n-grams in the document. D' Amore and Mah continued their application of the binomial distribution to derive an expected variance and, subsequently, a standard deviation for n-gram occurrences. The final weight for n-gram i in document j is:

$$w_{ij} = \frac{f_{ij} - e_{ij}}{\sigma_{ij}}$$

where:

f_{ij}= frequency of an n-gram i in document j

e_{ij}= expected number of occurrences of an n-gram i in document

j σ_{ij} =standard deviation

The n-gram weight designates the number of standard deviations away from the expected value. The goal is to give a high weight to an n-gram that has occurred far more than expected and a low weight to an n-gram that has occurred only as often as expected.

D' Amore and Mah did several experiments to validate that the binomial model was appropriate for n-grams. Unfortunately, they were not able to test their approach against a term-based one on a large standardized corpus.

2.4.2 Damashek

Damashek expanded on D' Amore and Mah's work by implementing a five-gram- based measure of relevance Damashek's algorithm relies upon the vector space model, but computes relevance in a different fashion.Instead of using stop words and stemming to normalize the expected

occurrence of n-grams, a centroid vector is used to eliminate noise. To compute the similarity between a query and a document, the following cosine measure is used:

$$SC(Q,D) = \frac{\sum_{j=1}^{t}(w_{qj} - \mu_Q)(w_{dj} - \mu_D)}{\sqrt{\sum_{j=1}^{t}(w_{qj} - \mu_q)^2 \sum_{j=1}^{t}(w_{dj} - \mu_D)^2}}$$

Here μ_q and μ_d represent centroid vectors that are used to characterize the query language and the document language. The weights, W_{qj} and W_{dj} indicate the term weight for each component in the query and the document vectors. The centroid value for each n-gram is computed as the ratio of the total number of occurrences of the n-gram to the total number of n-grams. This is the same value used by D' Amore and Mah. It is not used as an expected probability for the n-grams, but merely as a characterization of the n-gram's frequency across the document collection. The weight of a specific n-gram in a document vector is the ratio of the number of occurrences of the n-gram in the document to the total number of all of the n-grams in the document. This "within document frequency" is used to normalize based on the length of a document, and the centroid vectors are used to incorporate the frequency of the n-grams across the entire document collection. By eliminating the need to remove stop words and to support stemming, (the theory is that the stop words are characterized by the centroid so there was no need to eliminate them), the algorithm simply scans through the document and grabs n-grams without any parsing. This makes the algorithm language independent. Additionally, the use of the centroid vector provides a means of filtering out common n-grams in a document. The remaining n-grams are reverse engineered back into terms and used as automatically assigned keywords to describe a document.

2.4.3 Pearce and Nicholas

An expansion of Damashek's work uses n-grams to generate hypertext links . The links are obtained by computing similarity measures between a selected body of text and the remainder of the document.

After a user selects a body of text, the five-grams are identified, and a vector representing this selected text is constructed. Subsequently, a cosine similarity measure is computed, and the top rated documents are then displayed to the user as dynamically defined hypertext links. The user interface issues surrounding hypertext is the principal enhancement over Damashek's work. The basic idea of constructing a vector and using a centroid to eliminate noise remains intact.

2.4.4 Teufel

Teufel also uses n-grams to compute a measure of similarity using the vector space model . Stop words and stemming algorithms are used and advocated as a good means of reducing noise in the set of n-grams. However, his work varies from the others in that he used a measure of relevance that is intended to enforce similarity over similar documents. The premise was that if document A is similar to B, and B is similar to C, then A should be roughly similar to C. Typical

coefficients, such as inner product, Dice, or Jaccard , are non-transitive. Teufel uses a new coefficient, H, where:

$$H=X+Y-(XY)$$

and X is a direct similarity coefficient (in this case Dice was used, but Jaccard, cosine, or inner product could also have been used) and Y is an "indirect" measure that enforces transitivity. With the indirect measure, document A is identified as similar to document C. A more detailed description of the indirect
similarity measure is given . Good precision and recall was reported for the INSPEC document collection.
Language independence was claimed in spite of reliance upon stemmingand stop words.

2.4.5 Cavnar and Vayda

Most of this work involves using n-grams to recognize postal addresses. Ngrams were used due to their resilience to errors in the address. A simple scanning algorithm that counts the number of n-gram matches that occur between a query and a single line of text in a document was used. No weighting of any kind was used, but, by using a single text line, there is no need to normalize for the length of a document. The premise is that the relevant portion of a document appears in a single line of text. Cavnar's solution was the only documented approach tested on a large standardized corpus. For the entire TIPSTER document collection, average precision of between 0.06 and 0.15 was reported. It should be noted that for the AP portion of the collection an average precision of 0.35 was obtained. These results on the AP documents caused Cavnar to avoid further tuning. Unfortunately, results on the entire collection exhibited relatively poor performance. Regarding these results, the authors claimed that,"It is unclear why there should be such variation between the retrievability of the AP documents and the other document collections."

2.5 Regression Analysis

Another approach to estimating the probability of relevance is to develop variables that describe the characteristics of a match to a relevant document. Regression analysis is then used to identify the exact parameters that match the training data. For example, if trying to determine an equation that predicts a
person's life expectancy given their age:

Age	Life Expectancy
45	72
50	74
70	80

A simple least squares polynomial regression could be implemented, that would identify the correct values of a and (3 to predict life expectancy (LE) based on age (A):

$$LE = \alpha A + \beta$$

For a given age, it is possible to find the related life expectancy. Now, if we wish to predict the likelihood of a person having heart disease, we might obtain the following data:

Age	Life Expectancy	Heart Disease
45	72	yes
50	74	no
70	80	yes

We now try to fit a line or a curve to the data points such that if a new person shows up and asks for the chance of their having heart disease, the point on the curve that corresponds to their age could be examined. This second example is more analogous to document retrieval because we are trying to identify characteristics in a query-document match that indicate whether or not the document is relevant. The problem is that relevance is typically given a binary (l or 0) for training data-it is rare that we have human assessments that the document is "kind of" relevant. Note that there is a basic independence assumption that says age will not be related to life expectancy (an assumption we implied was false in our preceding example). Logistic regression is typically used to estimate dichotomous variables-those that only have a small set of values, (i.e., gender, heart disease present, and relevant documents).

Focusing on information retrieval, the problem is to find the set of variables that provide some indication that the document is relevant.

Matching Terms	Size of Query	Size of Document	Relevant?
5	10	30	yes
8	20	45	no

Six variables used are given below:

- The mean of the total number of matching terms in the query.
- The square root of the number of terms in the query.
- The mean of the total number of matching terms in the document.
- The square root of the number of terms in the document.
- The average idf of the matching terms.
- The total number of matching terms in the query.

A brief overview of polynomial regression and the initial use of logistic regression is given . However, the use of logistic regression requires the variables used for the analysis to be independent. Hence, the logistic regression is given in two stages. Composite clues which are composed of independent variables are first estimated. Assume clues 1-3 above are found in one composite clue and 4-6 are in the second composite clue. The two stages proceed as follows:

Stage 1:

A logistic regression is done for each composite clue.

$$
\begin{aligned}
logO(R|C_1) &= c_0 + c_1X_1 + c_2X_2 + c_3X_3 \\
logO(R|C_2) &= d_0 + d_1X_4 + d_2X_5 + d_3X_6
\end{aligned}
$$

At this point the coefficients Co, C1, C2, C3 are computed to estimate the relevance for the composite clue C1. Similarly, do, d1, d2 , d3 estimate the relevance of C2.

Stage 2:

The second stage of the staged logistic regression attempts to correct for errors induced by the number of composite clues. As the number of composite clues grows, the likelihood of error increases. For N composite clues, the following logistic regression is computed:

$$
logO(R|C_1, C_2, \ldots, C_N) = e_0 + e_1Z + e_2N
$$

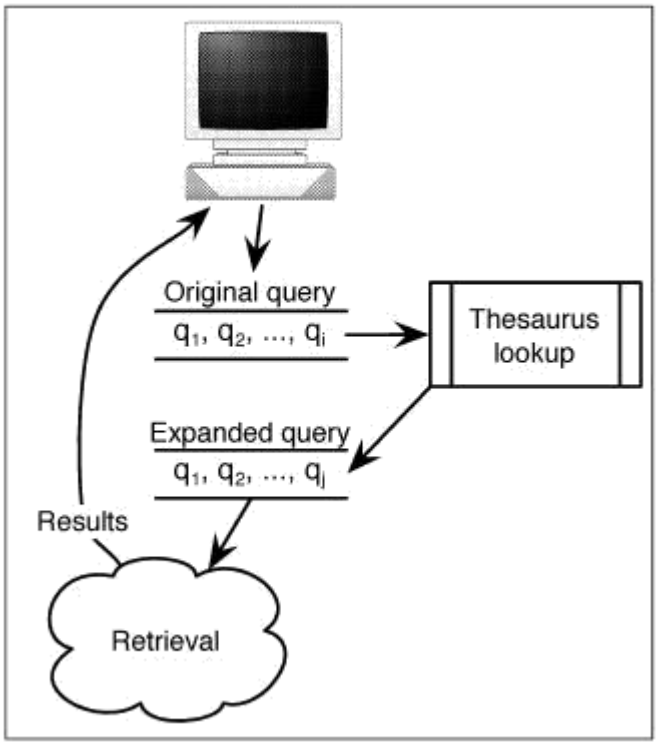
where Z is computed as the sum of the composite clues or:

$$
Z = \sum_{i=1}^{N} \log O(R|C_i)
$$

The results of the first stage regression are applied to the second stage. It should be noted that further stages are possible. Once the initial regression is completed, the actual computation of similarity coefficients proceeds quickly. Composite clues are only dependent on the presence or absence of terms in the document and can be precomputed. Computations based on the number of matches found in the query and the document are done at query time, but involve combining the coefficients computed in the logistic regression with the precomputed segments of the query. The question is whether or not the coefficients can be computed in a generic fashion that is resilient to changes in the document collection. The appealing aspects of this approach are that experimentation can be done to identify the best clues, and the basic independence assumptions are avoided. Additionally, the approach corrects for errors incurred by the initial logistic regression.

2.6 Thesauri

One of the most intuitive ideas for enhancing effectiveness of an information retrieval system is to include the use of a thesaurus. Almost from the dawn of the first information retrieval systems in the early 1960's, researchers focused on incorporating a thesaurus to improve precision and recall. The process of using a thesaurus to expand a query is illustrated in Figure



A thesaurus, at first glance, might appear to assist with a key problem-two people very rarely describe the same concepts with the same terms (i.e., one person will say that they went to a party while another person might call it a gathering). This problem makes statistical measures that rely on the number of matches between a query term and the document terms somewhat brittle when confronted with semantically equivalent terms that happen to be syntactically distinct. A query that asks for information about dogs is probably also interested in documents about canines. A document relevant to a query might not match any of the terms in the query. A thesaurus can be used either to assign a common term for all syn onyms of a term, or to expand a query to include all synonymous terms. Intuitively this should work fine, but unfortunately, results have not been promising. This section describes the use of hand-built thesauri, a very labor intensive means of building a thesaurus, as well as the quest for a sort of holy grail of information retrieval, an automatically generated thesaurus.

2.6.1 Automatically Constructed Thesauri

A hand-built thesaurus might cover general terms, but it lacks domain specific terms. A medical document collection has many terms that do not occur in a general purpose thesaurus. To avoid the need for numerous hand-built domain-specific thesauri, automatic construction methods were implemented.

2.6.1.1 Term Co-occurrence

An early discussion of automatic thesaurus is to represent each term as a vector. The terms are then compared using a similarity coefficient that measures the Euclidean distance, or angle, between the two vectors. To form a thesaurus for a given term t, related terms for t are all those terms u such that SC(t, u) is above a given threshold. Note, this is an $O(t^2)$ process so it is often common to limit the terms for which a related term list is built. This is done by using only those terms that are not so frequent that they become stop terms, but not so infrequent that there is little chance they have many synonyms.

Consider the following example:

D1 : "a dog will bark at a cat in a tree"

D2 : "ants eat the bark of a tree"

This results in the term-document occurrence matrix found in Table 3.1 This results in the term-document occurrence matrix found in Table .

To compute the similarity of term i with term j, a vector of size N, where N is the number of documents, is obtained for each term. The vector corresponds to a row in the following table. A dot product similarity between "bark" and "tree" is computed as:

$$SC(bark, tree) = < 1\ 1 > \bullet < 1\ 1 > = 2$$

The corresponding term-term similarity matrix is given in Table. The matrix is symmetric as SC(tl, t2) is equivalent to SC(t2, tl). The premise is that words are similar or related to the company they keep. Consider "tree" and "bark"; in our example, these terms co-occur twice in two documents. Hence, this pair has the highest similarity coefficient. Other simple extensions to this approach are the use of word stems instead of whole terms . The use of stemming is important here so that the term cat will not differ from cats. The tf-idf measure can be

term	D_1	D_2
a	3	1
ants	0	1
at	1	0
bark	1	1
cat	1	0
dog	1	0
eat	0	1
in	1	0
of	0	1
the	0	1
tree	1	1
will	1	0

Term-Document matrix

term	a	ants	at	bark	cat	dog	eat	in	of	the	tree	will
a	0	1	3	4	3	3	1	3	1	1	4	3
ants	1	0	0	1	0	0	1	0	1	1	1	0
at	3	0	0	1	1	1	0	1	0	0	1	0
bark	4	1	1	0	1	1	1	1	1	1	2	1
cat	3	0	1	1	0	1	0	1	0	0	1	1
dog	3	0	1	1	1	0	0	1	0	1	1	1
eat	1	1	0	1	0	0	0	0	1	0	1	0
in	3	0	1	1	1	1	0	0	0	0	1	1
of	1	1	0	1	0	0	1	0	0	1	1	0
the	1	1	0	1	0	0	1	0	1	0	1	0
tree	4	1	1	2	1	1	1	1	1	1	0	1
will	3	0	1	1	1	1	0	1	0	0	1	0

Term-term similarity matrix

used in the term-term similarity matrix to give more weight to co-occurrences between relatively infrequent terms. This summarizes much of the work done in the 1960's using term clustering, and provides some additional experiments . The common theme of these papers is that the term-term similarity matrix can be constructed, and then various clustering algorithms can be used to build clusters

of related terms. Once the clusters are built, they are used to expand the query. Each term in the original query is found in a cluster that was included in some portion or all (depending on a threshold) elements of its cluster. Much of the related work one during this time focused on different clustering algorithms and different thresholds to identify the number of terms added to the cluster. The conclusion

was that the augmentation of a query using term clustering did not improve on simple queries that used weighted terms.

A domain-specific thesaurus was constructed on information about the Caenorhabditis elegans worm in support of molecular biologists . A term-term similarity measure was built with phrases

and terms. A weight that used tf-idfbut also included another factor Pi, was used where Pi indicated the number of terms in phrase i. Hence, a two-term phrase was weighted double that of a single term. The new weight was:

$$w_{ij} = tf_{ij} \times \log\left(\frac{N}{df_i} \times p_i\right)$$

Using this new weight, an asymmetric similarity coefficient was also developed. The premise was that the symmetric coefficients are not as useful for ranking because a measurement between t_i t_j can become very skewed if either t_i or t_j occurs frequently. The asymmetric coefficient allows for a ranking

of an arbitrary term t_i, frequent or not, with all other terms. Applying a threshold to the list means that for each term, a list of other related terms is generated-and this can be done for all terms.

The measurement for $SC(t_i,t_j)$ is given as:

$$SC(t_i, t_j) = \left(\frac{\sum_{k=1}^{n} min(tf_{ik}, tf_{jk}) \log\left(\frac{N}{df_{ij}} \times p_j\right)}{\sum_{k=1}^{n} w_{ik}} \right) \times W_j$$

where df_{ij} is the number of co-occurrences of term i with term j. Two additional weights make this measure asymmetric: Pj and Wj . As we have said Pj is a small weight included to measure the size of term j. With all other weights being equal, the measure: SC(food, apple pie) > SC(food, apple) since phrases are weighted higher than terms. The weighting factor, Wj , gives additional preference to terms that occur infrequently without skewing the relationship between term i and term j. The weight Wj is given as:

$$W_j = \left(\frac{\log\left(\frac{N}{df_j}\right)}{\log N} \right)$$

Consider the term york and its relationship to the terms new and castle. Assume new occurs more frequently than castle. With all other weights being equal, the new weight, Wj, causes the following to occur:

$$SC(york, castle) > SC(york, new)$$

This is done without regard for the frequency of the term york. The key is that we are trying to come up with a thesaurus, or a list of related terms, for a given term (i.e., york). When we are deriving the list of terms for new we might find that york occurs less frequently than castle so we would have:

SC(new, york) > SC(new, castle)

Note that we were able to consider the relative frequencies of york and castle with this approach. In this case:

SC(new, york) = SC(york, new)

The high frequency of the term new drowns out any real difference between york and castle-or at least that is the premise of this approach. We note in our example, that new york would probably be recognized as a phrase, but that is not really pertinent to this example. Hence, at this point, we have defined SC(ti,tj). Since the coefficient is asymmetric we now give the definition of SC(tj, ti):

$$SC(t_j, t_i) = \left(\frac{\sum_{k=1}^{n} min(tf_{ik}, tf_{jk}) \log\left(\frac{N}{df_{ij}} \times p_i\right)}{\sum_{k=1}^{n} w_{jk}} \right) \times W_i$$

A threshold was applied so that only the top one hundred terms were used for a given term. These were presented to a user. For relatively small document collections, users found that the

thesaurus assisted their recall. No testing of generic precision and recall for automatic retrieval was measured.

2.6.1.2 Term Context

Instead of relying on term co-occurrence, some work uses the context (surrounding terms) of each term to construct the vectors that represent each term]. The problem with the vectors given above is that they do not differentiate the senses of the words. A thesaurus relates words to different senses. In the example given below, "bark" has two entirely different senses. A typical thesaurus lists "bark" as:

bark-surface of tree (noun)
bark-dog sound (verb

Ideally an automatically generated thesaurus would have separate lists of synonyms. The term-term matrix does not specifically identify synonyms, and Gauch and Wang do not attempt this either. Instead, the relative position of nearby terms is included in the vector used to represent a term .

The key to similarity is not that two terms happen to occur in the same document; it is that the two terms appear in the same context-that is they have very similar neighboring terms. Bark, in the sense of a sound emanating from a dog, appears in different contexts than bark, in the sense of a tree surface. Consider the following three sentences:

s 1: "The dog yelped at the cat."
s2 : "The dog barked at the cat."
s3 : "The bark fell from the tree to the ground."

In sentences s1 and s2, yelped is a synonym for barked, and the two terms occur in exactly the same context. It is unlikely that another sense of bark would appear in the same context. "Bark" as a suiface of tree more commonly would have articles at one position to the left instead of two positions to the left, etc.

To capture the term's context, it is necessary to identify a set of context terms. The presence or absence of these terms around a given target term will determine the content of the vector for the target term. The authors assume the highest frequency terms are the best context terms, so the 200 most frequent terms (including stop terms) are used as context terms. A window of size seven was used. This window includes the three terms to the left of the target term and the three terms to the right of the target term. The new vector that represents target term i will be of the general form:

$$T_i = < v_{-3} v_{-2} v_{-1} v_1 v_2 v_3 >$$

where each vector, Vi, and i = -3, -2, -1, 1,2, and 3 corresponds to a 200 element vector that represents the context of the target term for a given position. The vector V_{-3} contains a component for each of the 200 context terms that occur three terms to the left of the target term. Similarly, the vector V3 contains
a component for each of the 200 context terms that occur three terms to the right of the target.

The Vi vectors are all concatenated to form the entire Ti vector for the term. For a simple example, we build the context vectors for the terms bark and yelp based on the document collection s1, s2, and s3 . To simplify the example,we assume that stemming is done to normalize bark and barked and that the and at are the only two context terms occupying components one and two, respectively, of the context vectors. For our test document collection we would obtain:

$$T_{bark} = [< 00 >< 10 >< 10 >< 01 >< 10 >< 10 >]$$

$$T_{yelp} = [< 00 >< 10 >< 00 >< 01 >< 10 >< 00 >]$$

The matching of s1 and s2 is the driving force between the two vectors being very similar. The only differences occur because of the additional word sense that occurs in s3 .

This example uses the frequency of occurrence of a context term as the component of the context vectors. The authors use a measure that attempts to place more weight on context terms that occur
less frequently than might be expected. The actual component value of the jth component of vector Vi, is a mutual information measure. Let:

$$\begin{aligned} df_{ij} &= \text{frequency of co-occurrence of context term } j \text{ with target term } i \\ tf_i &= \text{total occurrences of context term } i \text{ in the collection} \\ tf_j &= \text{the total occurrences of context term } j \text{ in the collection} \end{aligned}$$

$$v_{ij} = \log\left(\frac{N df_{ij}}{(tf_i)(tf_j)} + 1\right)$$

This gives a higher weight to a context term that appears more frequently with a given target term than predicted by the overall frequencies of the two terms.

2.6.1.3 Clustering with Singular Value Decomposition

First a matrix, A, is computed for terms that occur 2000-5000 times. The matrix contains

the number of times these terms co-occur with a term window of size k (k is 40 in this work). Subsequently, these terms are clustered into 200 A-classes (group average agglomerative clustering is used). For example, one A-class, g_{Al}, might have terms (tl, t2, t3) and another, g_{A2}, would
have (t4,t5).

Subsequently, a new matrix, B, is generated for the 20,000 most frequent terms based on their co-occurrence between clusters found in the matrix. For example, if term tj co-occurs with term tl ten times, term t2 five times, and term t4 six times, B[I, j] = 15 and B[2, j] = 6. Note the use of clusters has reduced the size of the B matrix and provides substantially more training information. The rows of B correspond to classes in A, and the columns correspond to terms. The B matrix is of size 200 x 20,000. The 20,000 columns are then clustered into 200 B-classes using the buckshot clustering algorithm] indicates the number oftimes $term_j$ co-occurs with the B-classes. Once this is done, the C matrix is decomposed and singular values are computed to represent the matrix. This is similar to the technique used for latent semantic indexing . The SVD is more tractable at this point since only 200 columns exist. A document is represented by a vector that is the sum of the context vectors (vectors that correspond to each column in the SVD). The context vector is used to match a query.

Another technique that uses the context vector matrix, is to cluster the query based on its context vectors. This is referred to as word factorization. The queries were partitioned into three separate clusters. A query is then run for each of the word factors and a given document is given the highest rank of the three. This requires a document to be ranked high by all three factors to receive an overall high rank. The premise is that queries are generally about two or three concepts and that a relevant document has information relevant to all of the concepts.

Overall, this approach seems very promising. It was run on a reasonably good-sized collection (the Category B portion of TIPSTER using term factorization, average precision improved from 0.27 to 0.32-an 18.5% overall improvement).

2.6.1.4 Using only Document Clustering to Generate a Thesaurus

Another approach to automatically build a thesaurus is, a document clustering algorithm is implemented to partition the document collection into related clusters. A document-document similarity coefficient is used. Complete link clustering is used here, but other clustering algorithms could be used (for more details on clustering algorithms . The terms found in each cluster are then obtained. Since they occur in different documents within the cluster, different operators are used to obtain the set of terms that correspond to a given cluster. Consider documents with the following terms:

$$
\begin{aligned}
D_1 &= t_1, t_2, t_3, t_4 \\
D_2 &= t_2, t_4 \\
D_3 &= t_1, t_2
\end{aligned}
$$

The cluster can be represented by the union of all the terms {t1, t2, t3, t4}, the intersection {t2}, or some other operation that considers the number of documents in the cluster that contain the term. Crouch found that simple clustering worked the best. The terms that represented the cluster now appear as a thesaurus class, in that they form the automatically generated thesaurus. The class is first reduced to obtain only the good terms. This is done by using a term discriminating function that is based on document frequency.

Queries are expanded based on the thesaurus class. Any term that occurs in the query that matches a term in the thesaurus class results in all terms in the class being added to the query. Average precision was shown to improve ten percent for the small ADI collection and fifteen percent for the Medlars collection. Unfortunately, both of these results were for small collections, and the document clustering is computationally expensive, requiring $O(N^2)$ time, where N is the number of documents in the collection.

2.6.2 Use of Manually Generated Thesaurus

Although a manually generated thesaurus is far more time consuming to build, several researchers have explored the use of such a thesaurus to improve precision and recall.

2.6.2.1 Extended Relevance Ranking with Manual Thesaurus

A system developed in 1971 used computers to assist with the manual construction of a thesaurus at the Columbia University School of Library Service. The algorithm was essentially equivalent to a simple thesaurus editor.

Manual thesaurus construction is typically used for domain-specific thesauri. A group of experts is convened, and they are asked to identify the relationship between domain-specific terms. Ghose and Dhawle note that manual generation of these thesauri can be more difficult to build for social sciences than natural sciences given that there is more disagreement about the meaning of domain-specific terms in the social sciences . A series of handbuilt thesauri (each one was constructed by students) was described in . These thesauri were generated by the relationships between two terms-such as dog is-a animal. Ultimately the thesauri were combined into one that contained seven groups of relations. These groups were:

- Antonyms
- All relations but antonyms
- All relations
- Part-whole and set relations
- Co-location relations
- Taxonomy and synonymy relations
- Paradigmatic relations

The antonym relation identified terms that were opposites of one another (e.g., night, day) and is-part-of identifies entities that are involved in a bill-of-materials relationship (e.g., tire, automobile). Co-location contains relations between words that frequently co-occur in the same phrase or sentence. Taxonomy and synonym represent synonyms. Paradigmatic relations relate different forms of words that contain the same semantic core such as canine and dog. Experiments in adding each or all of the terms from these relations were done on a small document collection with relevance judgments obtained by the researchers conducting the study. Use of all relations, with the exception of antonyms, delivered the best average precision and recall, but there was little overall improvement.

A study done in esaurus containing three different relations: equivalence (synonym), hierarchical (is-a), and associative relationships . Recall of a fairly large (227,000) document collection composed of Finnish newspaper articles was shown to increase from 47 percent to 100 percent while precision only decreased from 62.5 percent to 51 percent. Fortunately, the work was done on a large collection, however, the thesaurus was hand-built for the test and contained only 1,011 concepts and a total of 1,573 terms. Only thirty queries were used, and the high results are clearly due to "good" terms found in the thesaurus.

Given the nature of the highly specific thesaurus, this result might be very similar in nature to the manual track of the TREC conference where participants are allowed to hand-modify the original query to include more discriminating terms. The synonym, narrower term, and related term searches all showed a 10 to 20% increase in recall from a 50% baseline. The union search (using all values) showed a rather high fifty percent increase in average precision. This work does represent one of the few studies outside of the TIPSTER collection that is run on a sizable collection. It is not clear, however, how applicable the results are to a more general collection that uses a more general thesaurus.

2.6.2.2 Extending Boolean Retrieval With a Hand Built Thesaurus

All work described attempts to improve relevance ranking using a thesaurus. We describe the extensions to the extended Boolean retrieval model as a means of including thesaurus information in a Boolean request . A description of the extended Boolean model is found Values for p were attempted, and a value of six (value suggested for standard extended Boolean retrieval by Salton in to perform the best. Results of this approach showed slightly higher effectiveness.