To compute Q(*gold* ⋁ *silver* ⋁ *truck*), we look at the documents which contain each of those terms. *Gold* is in $D_1$ and $D_3$ with a strength of membership of 0.143. *Silver* is only in $D_2$ with a strength of membership of 0.25. Similarly, *truck* is in $D_2$ with a membership of 0.125 and $D_3$ with 0.143. Applying the maximum set membership to implement the fuzzy OR, we obtain:

Q(*gold* ⋁ *silver* ⋁ *truck*) = $\{(D_1, 0.143), (D_2, 0.25), (D_3, 0.143)\}$

The documents would then be ranked, $D_2$, $D_1$, $D_3$ based on strength of membership for each document. As another example, consider the query: (*truck* ⋀ (*gold* ⋁ *silver*)). For this query, we determine D(*truck*) and D(*gold* ⋁ *silver*)—we will refer to these two sets as set A and set B.

A = D(*truck*) = $\{(D_2, 0.125), (D_3, 0.143)\}$

For D(*gold* ⋁ *silver*) we proceed as before, taking the maximum value of each degree of membership for each document in which either term appears. From our previous computation, we determine:

B = D(*gold* ⋁ *silver*) = $\{(D_1, 0.143), (D_2, 0.25), (D_3, 0.143)\}$

Taking the fuzzy intersection of set A with set B we use the minimum strength of membership. This yields:

A ⋂ B = D(*truck* ⋀ (*gold* ⋁ *silver*)) = $\{(D_2,0.125), (D_3, 0.143)\}$

At this point, we have not incorporated any query weights. We now modify the example to multiply each strength of membership by the *idf* for each query term. We use the following query term weights:

$$\text{gold} = \log \frac{3}{2} = 0.176$$

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

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

We now compute D(*gold* ⋁ *silver* ⋁ *truck*). $D_1$ includes only *gold* with a strength of 0.143. *Gold* has a query term weight of 0.176, so $D_1$ has a weighted strength of membership of (0.143)(0.176) = 0.025. *Silver* and *truck* are found in $D_2$. *Silver* has a strength of membership of 0.25 and a weight of 0.477; so the weighted strength of (0.25)(0.477) = 0.119. Similarly, for *truck*, the weighted strength is (0.125)(0.176) = 0.022. Since we are taking the union, we take the maximum value so $D_2$ will have a strength of membership of 0.119.

For $D_3$, both *gold* and *truck* are present with a strength of 0.143 and both terms are weighted by 0.176. Hence, the weighted strength is (0.143)(0.176) = 0.025. The fuzzy set D(*gold* $\bigvee$ *silver* $\bigvee$ *truck*) = $\{(D_1, 0.025), (D_2, 0.119), (D_3, 0.025)\}$ For the query, D(*truck* $\bigwedge$ (*gold* $\bigvee$ *silver*)) we must again determine D(*gold* $\bigvee$ *silver*) and D(*truck*). Using the weighted strength of membership yields:

A = D(*truck*) = $\{(D_2, 0.022), (D_3, 0.025)\}$
B = D(*gold* $\bigvee$ *silver*) = $\{D_1, 0.025), (D_2, 0.119), (D_3, 0.025)\}$

Again, taking the minimum strength of membership to compute the intersection, we obtain:

A $\bigcap$ B = D(*truck* $\bigwedge$ (*gold* $\bigvee$ *silver*)) = $\{(D_2, 0.022), (D_3, 0.025)\}$

### 2.9.2 Using a Concept Hierarchy

An approach using fuzzy logical inference with a concept hierarchy was used in the FIRST system (Fuzzy Information Retrieval SysTem) [Lucarella and Morara, 1991].

A concept network is used to represent concepts found in documents and queries and to represent the relationships between these concepts. A concept network is a graph with each vertex representing a concept and a directed edge between two vertices representing the strength of the association of the two concepts. A document can then be represented as a fuzzy set of concepts:

$$d_1 = \{(C_1, w_1), (C_2, w_2), \ldots, (C_3, w_3)\}$$

This indicates that document one contains the concepts $(C_1, C_2, C_3)$ and the strength of each concept is given by $(w_1, w_2, w_3)$. The link relationships are defined as *fuzzy transitive* so that if $C_i$ is linked to $C_j$, and $C_j$ is linked to $C_k$, and the strength of $C_i$ to $C_k$ is defined as:

$$F(C_i, C_k) = Min(F(C_i, C_j), F(C_j, C_k))$$

To compute the strength between two concepts, take the minimum value of all edges along the path, then add query Q at the root of the concept hierarchy. For each concept linked to the query, it is possible to obtain the strength of that concept for a given document. To do so, find all paths through the concept graph from the concept to the document and take the minimum of all edges that connect the source concept to the document. Each of these paths results in a single value. An aggregation rule is then applied to compute the strength of the source concept as the maximum of the value returned by each distinct path. A detailed example is given in [Lucarella and Morara, 1991]. Note for queries

involving more than one initial concept, the appropriate fuzzy operations are applied to each pair of arguments in a Boolean operator.

A comparison of the vector space model to this approach was done. A 300 document Italian test collection was used with a handbuilt concept graph that contained 175 concepts. Ten queries were chosen, and FIRST had comparable precision to vector space and had higher recall.

### 2.9.3 Allowing Intervals and Improving Efficiency

In [Chen and Wang, 1995], Lucarella's model was extended through the use of intervals rather than real values as concept weights. Additionally, an efficiency improvement was added in that a concept matrix was developed.

The concept matrix is a C x C matrix that is represented such that M($C_i$, $C_j$) indicates the strength of the relationships between $C_i$ and $C_j$. The strength is defined as a single value or an interval such that the strength occurs somewhere inside of the interval. The transitive closure T of M is computed via successive matrix multiplications of M. Once the T matrix is computed, an entry $T(C_i, C_j)$ indicates the strength of the relationship of $C_i$ to $C_j$, where the strength is computed as the maximum of all paths from $C_i$ to $C_j$.

Although the initial computation of T is expensive for a concept network with a high number of concepts, T efficiently computes a similarity coefficient. First, a new matrix of size $D \times C$ maps all of the documents to concepts. An entry, $t_{ij}$, in this matrix indicates the strength of the relationship between document $i$ and concept $j$. Values for $t_{ij}$ are obtained either directly, if the concept appears in the given document, or indirectly, if the concept does not appear in the given document—in this case the T matrix is used to obtain the weight for that concept.

Given a document, $D_i$, with concepts $(c_{i1}, c_{i2}, \ldots, c_{in})$ and a query Q with concepts $(x_1, x_2, \ldots, x_n)$, a similarity coefficient is computed for all concepts that exist in the query (a concept that does not exist in the query has the value of "-"):

$$SC(Q, D_i) = \sum_{q(j) \neq \text{“}-\text{”} \wedge j=1}^{n} T(t_{ij}, x_j)$$

where T(x,y) = 1 - $|x - y|$.

The function T measures the difference between the strength of the concept found in the document and the user input strength of the concept found in the query. The document strength is computed as the minimum of the strengths found on the path from the document to the concept. A small difference results in a high value of T, and the similarity coefficient simply sums these differences for each concept given in the query.

For intervals, a new function, S, computes the average of the distance between the high and low ends of the interval. For example, an interval of [3, 5] compared with an interval of [2, 6] results in a distance of $\frac{|3-2|+|5-6|}{2} = 1$. These differences are then summed for a similarity coefficient based on intervals.

## 2.10 Summary

We described nine different strategies used to rank documents in response to a query. The probabilistic model, vector space model, and inference networks all use statistical measures that essentially rely upon matching terms between a query and a document.

The vector space model represents documents and queries as vectors. Similarity among documents, and between documents and queries is defined in terms of the distance between two vectors. For example, one of the common similarity measures is the cosine similarity, which treats the difference between two documents or a document and a query as the cosine of the angle between these two vectors.

The probabilistic model uses basic probability theory with some key assumptions to estimate the probability of relevance. It is often criticized as requiring pre-existing relevance information to perform well. We described various means of circumventing this problem—including Kwok's novel idea of using self-relevance to obtain an initial estimate.

A relatively new strategy, language modeling is an approach that has previously worked well in speech recognition. Numerous papers have been written on this strategy in the last few years and it has now clearly taken its place among the most popular strategies discussed today.

Using *evidence*, inference networks use Bayesian inference to infer the probability that a document is relevant. Since inference networks are capable of modeling both vector space and probabilistic models, they may be seen as a more powerful model. In fact, different inference network topologies have yet to be fully explored.

Latent semantic indexing is the only strategy we presented that directly addresses the problem that relevant documents to a query, at times, contain numerous terms that are identical in meaning to the query but do not share the same syntax. By estimating the "latent semantic" characteristics of a term-term matrix, LSI is able to accurately score a document as relevant to the query even though the term-mismatch problem is present.

Neural networks and genetic algorithms are both used commonly for machine learning applications and have only initially been used in document ranking. Computational resource limitations have prevented their widespread use—but the potential for these strategies to retrieve documents that differ from other strategies makes them intriguing.

The fuzzy set and extended Boolean are older strategies that extend the widespread use of Boolean requests into relevance ranks. Other strategies require users to submit a list of terms—instead of a Boolean request—and a ranking is then obtained. In some applications, users prefer Boolean requests. This is particularly true for those users who have relied on Boolean requests for years.

So, which strategy is best? This is still an area for debate, and therefore, for further investigation. To our knowledge, no head-to-head comparison of all of these strategies has been implemented. TREC (Text REtrieval Conference) activities evaluate complete information retrieval systems, but a system includes a strategy and a set of utilities, as well as, a variety of other implementation details. Thus far, no decisive conclusion can be deduced about the contribution of one strategy given a fixed framework of all other system components. Additional thoughts and comments related to these strategies are described in [Salton, 1989, Kowalski and Maybury, 2000], and some of the original papers describing these efforts were reprinted in [Sparck Jones and Willett, 1997].

## 2.11 Exercises

1 Show how inference networks can be used to model either the vector space model or the probabilistic model.

2 Download *Alice in Wonderland* from the internet. Write some code to identify the *idf* of each term. Identify how closely this work matches the Zipfian distribution.

3 Devise a new strategy that allows users to implement a Boolean retrieval request. The results should be ranked in order of the similarity to the query. Compare your new strategy with the extended Boolean retrieval strategy.

4 Describe the effect of adding new or changing existing documents to the vector space strategy. Which values must be recomputed? How can the strategy be slightly modified so that it is more resilient to the addition of new documents?

5 Develop a detailed example (as done with our standard query: *gold, silver, truck*) and our standard document collection to compute the similarity between the query and each document using Term Components.

6 Develop a detailed example (as done with our standard query: *gold, silver, truck*) and our standard document collection to compute the similarity between the query and each document using language models with Jelinek-Mercer smoothing.

7 It has been suggested that one or more strategies could be merged to form an improved result set. Give two general heuristics to merge results from two

arbitrary retrieval strategies. Describe the advantages and disadvantages inherent in your approach.

# Chapter 3

# RETRIEVAL UTILITIES

Many different 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.

- **Passage-based Retrieval**—The premise is that most relevant documents have a non-relevant portion, and the relevant passage is somewhat concentrated. Hence, queries are matched to passages (either overlapping or non-overlapping) of documents, and the results for each passage are then combined into a single similarity coefficient. The size of each passage is either fixed or varied based on the passage finding algorithm. Other approaches simply rank each sentence, paragraph, or other naturally occurring subdivision of a document.

- **Parsing** (noun phrase processing, stemming, etc.): Simply matching terms does not always yield good results. The identification and use of phrases is computationally much easier than the use of proximity operators. Parsing rules or lists of known phrases are used to identify valid phrases like "New York." These phrases are then treated as single terms. Other parsing techniques avoid common prefixes or suffixes to allow for matches between query and document terms that share a common root but have different prefixes or suffixes.

- **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.

- **Semantic Networks**—Concept hierarchies exist in which individual concepts are linked to other related concepts. The strength of the relationship is associated with the link. One such network is Wordnet [Beckwith and Miller, 1990], but others exist. Attempts to automatically construct such a network were pursued. The challenge is to use the network to expand queries or documents to contain more terms describing the contents of the query.

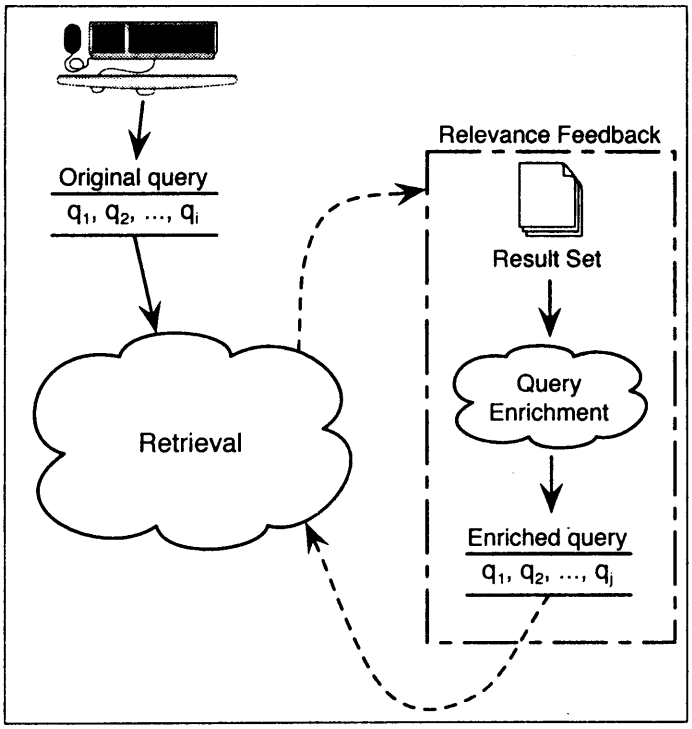
- **Regression Analysis**— Statistical techniques are used to identify parameters that describe characteristics of a match to a relevant document. These can then be used with a regression analysis to identify the exact parameters that refine the similarity measure.

## 3.1 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 3.1.

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.

*Figure 3.1.* Relevance Feedback Process




For example, an initial query "find information surrounding the various conspiracy theories about the assassination of John F. Kennedy" has both useful keywords and noise. The most useful keywords are probably *assassination* and *John F. Kennedy.* Like many queries (in terms of retrieval) there is some meaningless information. Terms such as *various* and *information* are probably not stop words (i.e., frequently used words that are typically ignored by an information retrieval system such as *a, an, and, the*), but they are more than likely not going to help retrieve relevant documents. The idea is to use all terms in the initial query and ask the user if the top ranked documents are relevant. The hope is that the terms in the top ranked documents that are said to be relevant will be "good" terms to use in a subsequent query.

Assume a highly ranked document contains the term *Oswald.* It is reasonable to expect that adding the term *Oswald* to the initial query would improve both precision and recall. Similarly, if a top ranked document that is deemed relevant by the user contains many occurrences of the term *assassination*, the weight used in the initial query for this term should be increased.

With the vector space model, the addition of new terms to the original query, the deletion of terms from the query, and the modification of existing term weights has been done. With the probabilistic model, relevance feedback initially was only able to re-weight existing terms, and there was no accepted means of adding terms to the original query. The exact means by which relevance feedback is implemented is fairly dependent on the retrieval strategy being employed. However, the basic concept of relevance feedback (i.e., run a query, gather information from the user, enhance the query, and repeat) can be employed with any arbitrary retrieval strategy.

Section 3.1.1 discusses the initial use of the vector space model to implement relevance feedback. Section 3.1.2 discusses the probabilistic means by which relevance feedback is added.

Relevance feedback has been fertile ground for research, as many tuning parameters are immediately apparent. Most feedback algorithms start with the premise that within the top $x$ ranked documents, the top $t$ terms will be used. Finding correct values for $x$ and $t$, as well as examining the number of iterations required to obtain good results, has been the subject of a fair amount of research. Recently, a summer workshop was held which focused on identifying good values for $x$ and $t$ [Harman and Buckley, 2004].

### 3.1.1 Relevance Feedback in the Vector Space Model

Rocchio, in his initial paper, started the discussion of relevance feedback [Rocchio, 1971]. 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 $D_i$, and a measure of relevance between the query and the document vector is computed as $SC(Q, D_i)$, where SC is the similarity coefficient. As discussed in Section 2.1, 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 $n_1$ relevant document vectors, and the set S contains the $n_2$ non-relevant document vectors. Rocchio builds the new query $Q'$ from the old query $Q$ using the equation given below:

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

$R_i$ and $S_i$ 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 $Q_{i+1}$ is derived from $Q_i$ 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 non-relevant 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 $n_1$ and $n_2$, 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 $n_a$ and $n_b$ 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 $\alpha$, $\beta$, and $\gamma$ are referred to as Rocchio weights and are frequently mentioned in the annual proceedings of TREC. The optimal values were experimentally obtained, but it is considered common today to drop the use of non-relevant documents (assign zero to $\gamma$) 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:

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

Only the top ranked non-relevant document is used, instead of the sum of all non-relevant documents. Ide refers to this as the *Dec-Hi* (decrease using highest ranking non-relevant document) approach. Also, a more simplistic weight is described in which the normalization, based on the number of document vectors is removed, and $\alpha$, $\beta$, and $\gamma$ are set to one [Salton, 1971a]. This new equation is:

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

An interesting case occurs when the original query retrieves only non-relevant documents. Kelly addresses this case in [Salton, 1971b]. The approach suggests that an arbitrary weight should be added to the most frequently occurring

*concept* in the document collection. This can be generalized to increase the component with the highest weight. The hope is that the term was important, but it was drowned out by all of the surrounding noise. By increasing the weight, the term now rings true and yields some relevant documents. Note that this approach is applied only in manual relevance feedback approaches. It is not applicable to automatic feedback as the top $n$ documents are assumed, by definition, to be relevant.

### 3.1.2 Relevance Feedback in the Probabilistic Model

We described the basic probabilistic model in Section 2.2. Essentially, 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:

$$\begin{aligned} w_i &= \text{weight of term } i \text{ in a particular query} \\ R &= \text{number of documents that are relevant to the query} \\ N &= \text{number of documents in the collection} \\ r_i &= \text{number of relevant documents that contain term } i \\ n_i &= \text{number of documents that contain term } i \end{aligned}$$

$R$ and $r$ cannot be known at the time of the initial query unless training data with relevance information is available. Realistically, the presence of domain-independent training data is unlikely. Some other retrieval strategy such as

the vector space model could be used for the initial pass, and the top $n$ documents could be observed. At this point, $R$ can be estimated as the total relevant documents found in the top $n$ documents, and $r$ is the number of occurrences in these documents. The problem of requiring training data before the probabilistic retrieval strategy can be used is eradicated with the use of relevance feedback.

#### 3.1.2.1 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:

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

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

This equation results in a curve that maps frequency to an estimated number of relevant documents. Frequency is an indicator of the number of relevant documents that will occur because of a given term. 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 $r_i$ can be estimated given a value of $f$, and using the value of $r_i$, 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. However, we are unaware of work done with IW on the TREC collection.

#### 3.1.2.2 Computing New Query Weights

Some variations on the basic weighting strategy for use with relevance feedback were proposed in [Robertson and Sparck Jones, 1976]. The potential for using relevance feedback with the probabilistic model was first explored in [Wu and Salton, 1981]. Essentially, Wu and Salton applied Sparck Jones' equation for relevance information [Robertson and Sparck Jones, 1976]. They modified the approach by using the similarity coefficient found in the equation below. Given a vector $Q$ representing the query and a vector $D_i$ representing the documents with a collection of $t$ terms, the following equation computes the similarity coefficient. The components of $d_i$ are assumed to be binary. A one indicates the term is present, and a zero indicates the term is absent, and K

is a constant.

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

Using this equation requires estimates for $p_i$ and $u_i$. The simplest estimate uses $p_i = \frac{r_i+0.5}{R+1}$ and $u_i = \frac{n_i-r_i+0.5}{N-R+1}$. That is, the ratio of the number of relevant documents retrieved that contain term $i$ to the number of relevant documents is a good estimate of the evidence that a term $i$ results in relevance. The 0.5 is simply an adjustment factor. Similarly, the ratio of the number of documents that contain term $i$ that were not retrieved to the number of documents that are not relevant is an estimate of $u_i$. Substituting these probabilities into the equation yields one of the conventional weights [Robertson and Sparck Jones, 1976], $w_2$ we described in Section 2.2.1.1:

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

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 $n_i$ 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 $Q$ contains the old weights and $T$ contains the weights computed by using the initial first pass. $\beta$ is a scaling factor that indicates the importance of the 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

Sparck Jones because it allows for expansion as well as reweighting. Before this proposal, work in probabilistic retrieval relied upon the reweighting of old terms, but it did not allow for the addition of new terms.

4. Expand the query using a combination of the initial weight and the new weight. This is similar to variation number two above. Assuming $q_1$ to $q_m$ 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:

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

Additionally, a modified estimate for $p_i$ and $u_i$ was computed. These new values are given below:

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

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

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

Wu and Salton found the fourth variation, which combines results of reweighting and term expansion, to be the most effective. Relatively little difference was observed with the modified $p_i$ and $u_i$.

Salton and Buckley give an excellent overview of relevance feedback. Twelve variations on relevance feedback were attempted [Salton and Buckley, 1990]. These included: stemmed Rocchio, Ide, conventional Sparck Jones probabilistic equation (see Section 2.2.1), and the extended probabilistic given in Wu and Salton [Wu and Salton, 1981]. All twelve were tested against six small collections (CACM, CISI, CRAN, INSPEC, MED, and NPL). All of these collections were commonly used by many researchers prior to the development of the larger TIPSTER collection. Different parameters for the variations were tried, such that there were seventy-two different feedback methods in all. Overall Ide Dec-Hi (decrease using the highest ranking elements) performed the best—having a ranking of one in three of the six collections and a ranking of two and six in the others.

#### 3.1.2.3 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 [Wu and Salton, 1981]. The idea of using only a selection of the terms found in the top documents was presented in [Harman, 1988]. In this paper, 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:

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

where:

$$\begin{aligned} tf_{ik} &= \text{number of occurrences of term } i \text{ in document } k \\ f_k &= \text{number of occurrences of term } k \text{ in the collection} \\ N &= \text{number of terms in the collection} \end{aligned}$$

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:

$$\begin{aligned} p_k &= \text{number of documents in the relevant set that contain term } k \\ rtf_k &= \text{number of occurrences of term } k \text{ in the relevant set} \end{aligned}$$

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 $rn_k$, $n_k$, and $p_k$ were used to sort the top terms. The six sort orders tested were:

- $n_k$
- $p_k$
- $rn_k$
- $n_k \times rtf_k$
- $n_k \times f_k \times p_k$
- $n_k \times f_k$

The sort order: $n_k \times f_k \times p_k$, resulted in the highest improvement in average precision (9.4%). This is very similar to $p_k \times idf$ which is a reasonable measure given that $p_k$ is an intuitively good value to use (i.e., a term that appears frequently in the relevant set is probably a good term to add to the query). However, this will not be the case for noise terms that occur frequently across the collection. This explains why the $p_k$ value did not perform as well as when it was combined with $n_k$.

Six additional sort orders were tested in a follow-up paper [Harman, 1992]. The sorts tested were:

- $\frac{(RT_j)(df_i)}{N}$ where $RT_j$ is the total number of documents retrieved for query $j$, $df_i$ is the document frequency or number of documents in the collection that contain term $i$, and N is the number of documents in the collection. This gives additional weight to terms that appear in multiple documents of the initial answer set.
- $\frac{r_{ij}}{R} - \frac{df_i}{N}$ where $r_{ij}$ is the number of retrieved relevant documents for query $j$ that have term $i$. $R_j$ is the 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.
- $W_{ij} = \log_2 \frac{p_{ij}(1-q_{ij})}{(1-p_{ij})q_{ij}}$ where $W_{ij}$ is the term weight for term $i$ in query $j$. This is based on Sparck Jones probabilistic weights given in Section 2.2.1. The probability that term $i$ is assigned within the set of relevant documents to query $j$ is $p_{ij}$. The probability that term $i$ is assigned within the set of non-relevant documents for query $j$ is $q_{ij}$. 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. Additional details are given in [Robertson, 1990].
- $W_{ij}(p_{ij} - q_{ij})$ where the term weight is computed as given above.
- $\log(RTF_i + 1)(p_{ij} - q_{ij})$ where $RTF_i$ 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.

Once the sort order was identified, the number of terms to add to the new query was studied. A peak at twenty terms was identified. At TREC, similar differences were observed in which some groups engaged in "massive query expansion" in which all terms in the first phase are added to the query, while other groups use only a subset of those terms [Buckley et al., 1994, Salton and Buckley, 1990]. Some groups at TREC have used fifty terms and twenty phrases and obtained good results.

In [Lundquist et al., 1997] additional sort techniques were explored using the TIPSTER collection, and it was found that $p_k \times nidf$ performs well. The variable, $nidf$, is a normalized $idf$ using pivoted document length normalization (see Section 2.1.2). Additionally, it was shown that the use of the top ten items (either terms or phrases) resulted in a thirty-one percent improvement in average precision over the use of the top fifty terms and twenty phrases.

#### 3.1.2.4 Number of Feedback Iterations

The number of iterations
needed for successful relevance feedback was initially tested in 1971 by Salton [Salton, 1971d]. 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 in [Harman, 1992].

In her work, 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. All three variations of implementing relevance feedback across iterations performed comparably.

#### 3.1.2.5 User Interaction

As earlier stated, 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 [Spink, 1994, Spink, 1995]. 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

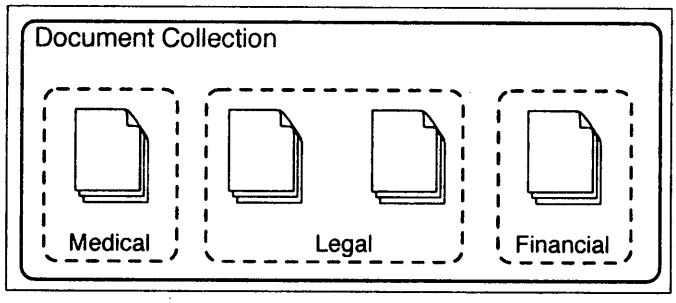
Users chose forty-eight terms (eleven percent) of their search terms (over forty queries) from relevance feedback. Of these, the end-user chose fifteen and the expert chose thirty-three. This indicates a more advanced user is more likely to take advantage of the opportunity to use relevance feedback.

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.

## 3.2 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 (see Figure 3.2). A query over legal material

*Figure 3.2.* Document Clustering




might then be directed (either automatically or manually) to the legal document cluster.

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. Namely, when considering each document as a matrix with size corresponding to the number of terms in the lexicon, calculating the similarity between a pair of documents requires comparison across the union of each of their non-zero matrix components. This is complicated by the fact that the document matrices are exceedingly sparse, as many terms appear in only a single document.

This may be an area where clustering will be computationally feasible enough to implement on a large scale. Also, initial work on a Digital Array Processor (DAP) was done to improve run-time performance of clustering algorithms by using parallel processing [Rasmussen and Willett, 1989]. Subsequently, these algorithms were implemented on a parallel machine with a torus interconnection network [Ruocco and Frieder, 1997].

A detailed review of clustering algorithms is given in [Salton, 1989]. 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 [Larsen and Aone, 1999, Willet, 1988, Dubes and Jain, 1988]. 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 [Steinbach et al., 2000, Zhao and Karypis, 2002]. 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

### 3.2.1 Result Set Clustering

Clustering was used as a utility to assist relevance feedback [Lu et al., 1996]. 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 [Zeng et al., 2004]. 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.

### 3.2.2 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}.

#### 3.2.2.1 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.

#### 3.2.2.2 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.

#### 3.2.2.3 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.

#### 3.2.2.4 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 [El-Hamdouchi and Willett, 1986]. The centroid is defined as the average vector in the vector space. If a vector represents the $i^{th}$ document, $D_i =< t_1, t_2, \ldots, t_n >$, the centroid C is written as $C =< c_1, c_2, \ldots, c_n >$. 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.

#### 3.2.2.5 Analysis of Hierarchical Clustering Algorithms

A paper that describes the implementation of all of these algorithms found that Ward's method typically took the longest to implement, with single link and complete linkage being somewhat similar in run-time [El-Hamdouchi and Willett, 1989].

A summary of several different studies on clustering is given in [Burgin, 1995]. In most studies, clusters found 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.

### 3.2.3 Clustering Without a Precomputed Matrix

Other approaches exist in which the $N \times N$ 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.

#### 3.2.3.1 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 $1, 2, \ldots, 10$ 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.

#### 3.2.3.2 Rocchio Clustering

Rocchio developed a clustering algorithm in 1966 [Rocchio, 1966], 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 in 1971 by Grauer and Messier [Grauer and Messier, 1971]. 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

#### 3.2.3.3 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 [Willett, 1990].

#### 3.2.3.4 Buckshot Clustering

Buckshot clustering is a clustering algorithm designed so that it 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 $O(n)$ which is a clear improvement over the $O(n^2)$ alternatives that require a document-document similarity matrix.

Buckshot clustering works by choosing a random sample of $\sqrt{kn}$ documents. These $\sqrt{kn}$ documents are then clustered by a hierarchical clustering algorithm (any one 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 $O(kn)$ time, as $O(kn)$ 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. Details of the buckshot clustering algorithm and its analysis are given in [Cutting et al., 1992].

#### 3.2.3.5 Non-negative Matrix Factorization

A more recent clustering algorithm uses non-negative matrix factorization (NMF). This provides a latent semantic space (see Section 2.6) 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 [Xu et al., 2003].

### 3.2.4 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 [Willett, 1988]:

- 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.

### 3.2.5 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 [Voorhees, 1986], but at present, no TREC participant has clustered the entire document collection.

#### 3.2.5.1 Parallel Document Clustering

Another means of improving run-time performance of clustering algorithms is to implement them on a parallel processor (see Chapter 7). Initial work on a Digital Array Processor (DAP) was done to improve the run-time of clustering algorithms by using parallel processing [Rasmussen and Willett, 1989]. These algorithms were implemented on a parallel machine with a torus interconnection network [Ruocco and Frieder, 1997]. A parallel version of the Buckshot clustering algorithm (see Section 3.2.3.4) 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 [Jensen et al., 2002]. There exists some other work specifically focused on parallel hierarchical clustering [Zhang et al., 1996, Guha et al., 1988], 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 [Olson, 1995], however these algorithms were developed for several types of specialized interconnection networks, and it is unclear whether they are appli-

cable to the simple bus connection that is common for many current parallel architectures.

Additional proposals use clustering as a utility to assist relevance feedback [Lu et al., 1996]. 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.

#### 3.2.5.2 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 [Schutze and Silverstein, 1997].

One means of reducing the size of the document vector is to use Latent Semantic Indexing (see Section 2.6) 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.

## 3.3 Passage-based Retrieval

Passage-based retrieval [Callan, 1994], is based on the premise that only a small portion of each relevant document (i.e., the relevant passage within the document) contains the information that is relevant to the query. By computing metrics that compare the entire document to the query, the noisy parts of the document (the sections that are nonrelevant) potentially mask the relevant segment of the document.

For instance, consider this book. This section is the only section that contains relevant information in response to a query that searches for *passage-based retrieval.* If the entire book was viewed as a single document, this section might contribute very little to the overall similarity coefficient between the book and the passage.

Since documents often are naturally segmented into chapters, sections, and subsections, it is reasonable to use each of these author-determined boundaries and simply rank the passages to the original query. A similarity coefficient must then merge the passage-based results and obtain a final coefficient.

Consider a document $D_1$ with sections A, B, C, and D. Further assume section C is the only section that mentions anything about the query. A similarity coefficient $SC(Q, D_1)$ could result in a coefficient that is heavily biased towards nonrelevance because sections A, B, and D have many terms that do

not match with terms in the query. The similarity coefficient reflects this and given the length of the document and the relatively small proportion of matching terms, or even terms that are semantically related, the document would have a low similarity coefficient. With passage-based retrieval, four separate coefficients are computed: SC(Q,A), SC(Q,B), SC(Q,C), and SC(Q,D). The four different similarity coefficients would then be merged. Several different techniques for merging these components are presented.

Passage-based research focuses on determining how to delimit a passage and combine each passage into a single similarity coefficient. The following sections discuss each of these problems and demonstrate some initial work in each area.

### 3.3.1 Marker-based Passages

Marker-based passages use section headers or paragraph indentation and vertical space as a means of partitioning passages. SGML tags found in long Federal Register documents were used in [Zobel et al., 1995].

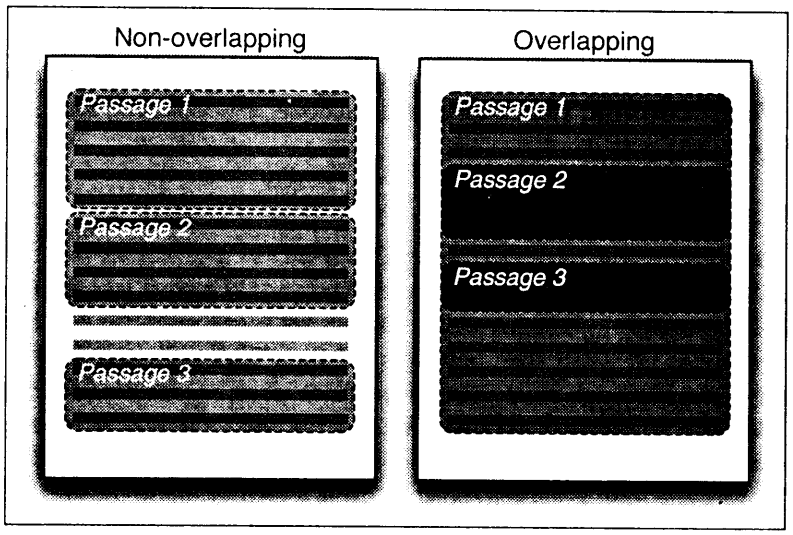
In similar work, paragraph markers were used. To avoid very long or short paragraphs, long paragraphs were partitioned based on size and short paragraphs were glued together. The passages were bounded such that no passage contained fewer than fifty terms or was larger than 200 terms [Callan, 1994]. In [Zobel et al., 1995] passages were glued together until a size of $p$ was exceeded. In both papers, modest improvement occurred, but results given with the Federal Register should be viewed with care as there are comparatively few relevant documents in this particular collection. The reason given for the limited success of this intuitively appealing approach is that the paragraph markers and section markers are prone to error on the part of the author and may not have resulted in a good semantic partitioning (i.e., one passage might have described numerous concepts).

### 3.3.2 Dynamic Passage Partitioning

Different approaches have been used to automatically find good partitions. These approaches attempt to partition documents differently based on the particular query [Callan, 1994]. One means of doing this is to find a term that matches the query and then build a passage around this match. If a term matches at position $n$, passage A will begin at position $n$ and continue until position $n + p$ where $p$ is a variable passage size. The next passage, B, will overlap with A and start at position $n + \frac{p}{2}$. Figure 3.3 illustrates the difference between overlapping and non-overlapping passages. For a term that matches at position ten, a small passage length of fifty results in passages around the terms [10, 60], [35, 85], [60, 110], etc. where [i, j] indicates the passage starts

at position $i$ and continues to $j$. Overlapping passages are intended to avoid splitting sections of relevant text.

*Figure 3.3.* Overlapping vs Non-Overlapping Passages




### 3.3.3 Merging Passage-based Similarity Measures

Passages contribute to the similarity coefficient in a number of different ways. One study tested twenty different methods of merging passage-based contributions [Wilkinson, 1994]. These methods ranged from simply taking the highest ranked passage as the similarity coefficient to combining document level contributions with passage level contributions. The work done in [Callan, 1994] also used a combination score with the document and the passage level evidence to obtain their best results. Similar results also occurred in [Wilkinson, 1994].

## 3.4 N-grams

Term-based search techniques typically use an inverted index or a scan of the text (details surrounding inverted index construction and search are given in Chapter 5). 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 [Pollock and Zamora, 1984, Thorelli, 1962, Zamora et al., 1981] and text compression [Yannakoudakis et al., 1982]. A survey of automatic correction techniques is found in [Kukich, 1992]. Additionally, n-grams were used to determine the authorship of documents [Kjell et al., 1994]. Traditional information retrieval algorithms based on n-grams are described in [D'Amore and Mah, 1985, Damashek, 1995, Pearce and Nicholas, 1993, Teuful, 1988, Cavnar and Vayda, 1993].

### 3.4.1 D'Amore and Mah

Initial information retrieval research focused on n-grams as presented in [D'Amore and Mah, 1985]. 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 n-gram 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 $j$
$\sigma_{ij}$ = 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.

### 3.4.2 Damashek

Damashek expanded on D'Amore and Mah's work by implementing a five-gram-based measure of relevance [Damashek, 1995]. 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 $\mu_q$ and $\mu_d$ 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. A description of this reverse engineering process is given in [Cohen, 1995]. Proof of language independence is given with tests covering English, German, Spanish, Georgian, Russian, and Japanese.

### 3.4.3 Pearce and Nicholas

An expansion of Damashek's work uses n-grams to generate hypertext links [Pearce and Nicholas, 1993]. 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.

### 3.4.4 Teufel

Teufel also uses n-grams to compute a measure of similarity using the vector space model [Teuful, 1988]. 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 (see Section 2.1.2), 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 in [Teuful, 1991].

Good precision and recall was reported for the INSPEC document collection. Language independence was claimed in spite of reliance upon stemming and stop words.

### 3.4.5 Cavnar and Vayda

N-grams were also proposed in [Cavnar, 1993, Cavnar and Vayda, 1993]. Most of this work involves using n-grams to recognize postal addresses. N-grams 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."

## 3.5 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 $\alpha$ and $\beta$ 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 (1 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 in [Fontaine, 1995] 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 in [Cooper et al., 1992]. 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 $c_0, c_1, c_2, c_3$ are computed to estimate the relevance for the composite clue $C_1$. Similarly, $d_0, d_1, d_2, d_3$ estimate the relevance of $C_2$.

**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. Further implementation details are found in [Fontaine, 1995].

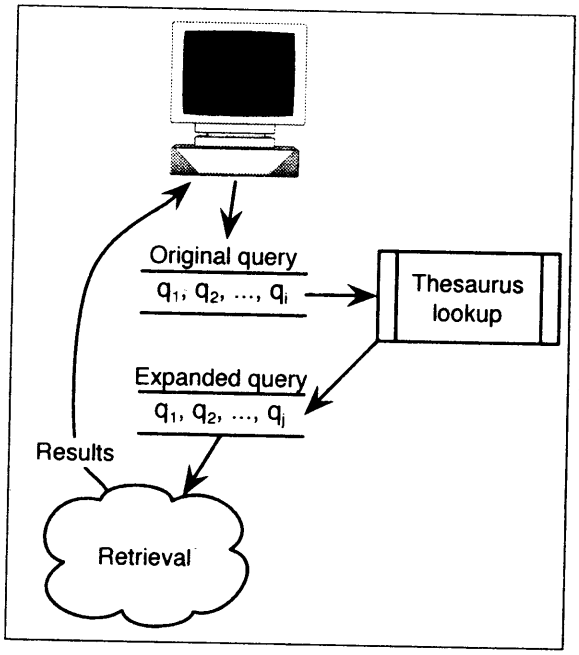
The question is whether or not the coefficients can be computed in a generic fashion that is resilient to changes in the document collection. The appealing aspects of this approach are that experimentation can be done to identify the

best clues, and the basic independence assumptions are avoided. Additionally, the approach corrects for errors incurred by the initial logistic regression.

## 3.6 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 3.4.

*Figure 3.4.* Using a Thesaurus to Expand a Query




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.

## 3.6.1 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.

### 3.6.1.1 Term Co-occurrence

An early discussion of automatic thesaurus generation is found in [Salton, 1971c]. The key to this approach 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:
$D_1$: "a dog will bark at a cat in a tree"
$D_2$: "ants eat the bark of a tree "

This results in the term-document occurrence matrix found in Table 3.1.
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 3.2.

The matrix is symmetric as $SC(t_1,t_2)$ is equivalent to $SC(t_2,t_1)$. 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 (for more on stemming see Section 3.8.1). The use of stemming is important here so that the term *cat* will not differ from *cats*. The *tf-idf* measure can be

*Table 3.1.* Term-Document Matrix

| 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 |

*Table 3.2.* Term-Term Similarity 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 |

used in the term-term similarity matrix to give more weight to co-occurrences between relatively infrequent terms.

Early work done with the term-term similarity matrix was given in [Minker et al., 1972]. This paper summarizes much of the work done in the 1960's using term clustering, and provides some additional experiments [Salton, 1971c, Sparck Jones and Jackson, 1968, Sparck Jones and Barber, 1971]. 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

done 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.

Additional work with term-term similarity matrices is presented in [Chen and Ng, 1995]. A domain-specific thesaurus was constructed on information about the *Caenorhabditis elegans* worm in support of molecular biologists [Chen et al., 1995]. A term-term similarity measure was built with phrases and terms. A weight that used *tf-idf* but also included another factor $p_i$, was used where $p_i$ 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: $p_j$ and $W_j$. As we have said $p_j$ 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, $W_j$, gives additional preference to terms that occur infrequently without skewing the relationship between term $i$ and term $j$. The weight $W_j$ 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, $W_j$, 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 $SC(t_i, t_j)$. Since the coefficient is asymmetric we now give the definition of $SC(t_j, t_i)$:

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

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.

#### 3.6.1.2 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 [Gauch and Wang, 1996]. 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 [Gauch and Wang, 1996].

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."
$S_2$: "The dog barked at the cat."
$S_3$: "The bark fell from the tree to the ground."

In sentences $S_1$ and $S_2$, *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 *surface 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. In [Gauch and Wang, 1996], 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, $v_i$, 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 $v_3$ contains a component for each of the 200 context terms that occur three terms to the right of the target.

The $v_i$ vectors are all concatenated to form the entire $T_i$ vector for the term. For a simple example, we build the context vectors for the terms *bark* and *yelp* based on the document collection $S_1$, $S_2$, and $S_3$. 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 $S_1$ and $S_2$ is the driving force between the two vectors being very similar. The only differences occur because of the additional word sense that occurs in $S_3$.

This example uses the frequency of occurrence of a context term as the component of the context vectors. In [Gauch and Wang, 1996], 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 $v_i$, 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.

Gauch and Wang use the top 200 terms, with a seven term window size; so each term vector is of size 1200. The vectors are then compared with a standard cosine measure, and all terms with a similarity above a threshold are used. The choice of which target words to choose is difficult, and after some experimentation 4,000 target words were chosen from the frequency list.

Queries were then expanded using only the top $n$ terms that fell above a certain threshold. Unfortunately, average precision for the expanded query was not significantly higher than without the expansion.

Analysis of the repeated failure of automatically generated thesauri built from term-term similarity matrices is given in [Peat and Willett, 1991]. They noted a key problem with using term co-occurrence to generate a thesaurus is that relatively frequent terms co-occur with other frequent terms. The result is a thesaurus in which one relatively general term is found to be related to another general term (e.g., *hairy* might be found to be related to *furry*). Although these

terms are related, they do not improve precision and recall because, due to their relatively high frequency, they are not good discriminators.

Interestingly, an early paper showed that randomly selecting terms for expansion was sometimes more effective than using those generated by a term-term similarity matrix [Smeaton and Rijsbergen, 1983]. Given a Zipfian distribution [Zipf, 1949] most terms appear infrequently (over half occur only once), so there is a good chance that the randomly selected terms were low frequency, and hence, did not do as much damage as a high frequency non-discriminating term.

#### 3.6.1.3 Clustering with Singular Value Decomposition

Schutze and Pedersen use term clustering and a singular value decomposition (SVD) to generate a thesaurus [Schutze and Pedersen, 1997]. 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—see Section 3.2.2.3). For example, one A-class, $g_{A1}$, might have terms $(t_1, t_2, t_3)$ and another, $g_{A2}$, would have $(t_4, t_5)$.

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 $t_j$ co-occurs with term $t_1$ ten times, term $t_2$ five times, and term $t_4$ six times, B[1, 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 (see Section 3.2.3.4).

Finally, a matrix, C, is formed for all terms in the collection. An entry $C[i, j]$ indicates the number of times 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 (see Section 2.6). 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).

#### 3.6.1.4 Using only Document Clustering to Generate a Thesaurus

Another approach to automatically build a thesaurus is described in [Crouch, 1989, Crouch, 1990]. First, 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 see Section 3.2).

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 $\{t_1, t_2, t_3, t_4\}$, the intersection $\{t_2\}$, 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. (See Section 2.1 for more details 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.

## 3.6.2 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.

### 3.6.2.1 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 [Hines and Harris, 1971].

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 [Ghose and Dhawle, 1977].

A series of handbuilt thesauri (each one was constructed by students) was described in [Wang et al., 1985]. 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 1993 used a thesaurus containing three different relations: equivalence (synonym), hierarchical (is-a), and associative relationships [Kristensen, 1993]. 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.

#### 3.6.2.2 Extending Boolean Retrieval With a Hand Built Thesaurus

All work described attempts to improve relevance ranking using a thesaurus. Lee et al., describe the extensions to the extended Boolean retrieval model as a means of including thesaurus information in a Boolean request [Lee et al., 1994]. A description of the extended Boolean model is found in Section 2.5. Values for $p$ were attempted, and a value of six (value suggested for standard extended Boolean retrieval by Salton in [Salton, 1989]) was found to perform the best. Results of this approach showed slightly higher effectiveness.

## 3.7 Semantic Networks

Semantic networks are based on the idea that knowledge can be represented by *concepts* which are linked together by various relationships. A semantic network is simply a set of nodes and arcs. The arcs are labelled for the type of relationship they represent. Factual information about a given node, such as its individual characteristic (color, size, etc.), are often stored in a data structure called a *frame*. The individual entries in a frame are called *slots* [Minsky, 1975].

A frame for a rose can take the form:

(**rose**
(*has-color* red)
(*height* 2 feet)
(*is-a* flower)
)

Here the frame *rose* is a single node in a semantic network containing an *is-a* link to the node *flower*. The slots *has-color* and *height* store individual properties of the rose.

Natural language understanding systems have been developed to read human text and build semantic networks representing the knowledge stored in the text [Schank, 1975, Schank and Lehnert, 1977, Gomez and Segami, 1989, Gomez and Segami, 1991]. It turns out that there are many concepts that are not easily represented (the most difficult ones are usually those that involve temporal or spatial reasoning). Storing information in the sentence, "A rose is a flower.", is easy to do as well as to store, "A rose is red", but semantic nets have difficulty with storing this information: "The rose grew three feet last Wednesday and was taller than anything else in the garden." Storing information about the size of the rose on different dates, as well as, the relative location of the rose is often quite difficult in a semantic network. For a detailed discussion see the section on "Representational Thorns" about the large-scale knowledge representation project called *Cyc* (a project in which a large portion of common sense reasoning is being hand-crafted) [Lenat and Guha, 1989].

Despite some of the problems with storing complex knowledge in a semantic network, research was done in which semantic networks were used to improve information retrieval. This work yielded limited results and is highly language specific, however, the potential for improvement still exists.

Semantic networks attempt to resolve the *mismatch* problem in which the terms in a query do not match those found in a document, even though the document is relevant to the query. Instead of matching characters in the query terms with characters in the documents, the *semantic distance* between the terms is measured (by various measures) and incorporated into a semantic network. The premise behind this is that terms which share the same meaning appear relatively close together in a semantic network. *Spreading activation* is one means of identifying the distance between two terms in a semantic network.

There is a close relationship between a thesaurus and a semantic network. From the standpoint of an information retrieval system, a thesaurus attempts to solve the same mismatch problem by expanding a user query with related terms

and hoping that the related terms will match the document. A semantic network subsumes a thesaurus by incorporating links that indicate "is-a-synonym-of" or "is-related-to," but a semantic network can represent more complex information such as an is-a hierarchy which is not found in a thesaurus.

One semantic network used as a tool for information retrieval research is WordNet [Beckwith and Miller, 1990]. WordNet is publicly available and contains frames specifically designed for words (some semantic networks might contains frames for more detailed concepts such as *big-and-hairy-person*). WordNet can be found on the Web at: **www.cogsci.princeton.edu/~wn**.

WordNet contains different entries for the various semantic meanings of a term. Additionally, various term relationships are stored including: *synonyms*, *antonyms* (roughly the opposite of a word), *hyponyms* (lexical relations such as *is-a*), and *meronyms* (is a *part-of*). Most nouns in WordNet are placed in the *is-a* hierarchy while antonyms more commonly relate adjectives.

Interestingly, less commonly known relations of *entailment* and *troponyms* are used to relate verbs. Two verbs are related by entailment when the first verb entails the second verb. For example, to *buy* something entails that you will *pay* for it. Hence, *buy* and *pay* are related by entailment. A troponym relation occurs when the two activities related by entailment must occur at the same time (temporally co-extensive) such as the pair *(limp, walk)*. Software used to search WordNet is further described in [Beckwith and Miller, 1990].

It is reasonable to assume that WordNet would help effectiveness by expanding query terms with synsets found in WordNet. Initial work done by Voorhees [Voorhees, 1993], however, failed to demonstrate an improvement in effectiveness. Even with manual selection of synsets, effectiveness was not improved when queries were expanded. A key obstacle was that terms in queries were not often found in WordNet due to their specificity—terms such as *National Rifle Association* are not in WordNet. Also, the addition of terms that have multiple meanings or word senses significantly degrade effectiveness. More recent work, with improvements to WordNet over time has incorporated carefully selected phrases and showed a small (roughly five percent) improvement [Liu et al., 2004].

Semantic networks were used to augment Boolean retrieval and automatic relevance ranking. We describe these approaches in the remainder of this section.

### 3.7.1 Distance Measures

To compute the distance between a single node in a semantic network and another node, a spreading activation algorithm is used. A pointer starts at each of the two original nodes and links are followed until an intersection occurs between the two points. The shortest path between the two nodes is used to compute the distance. Note that the simple shortest path algorithm does not

apply here because there may be several links that exist between the same two nodes. The distance between nodes $a$ and $b$ is:

Distance($a$,$b$) = minimum number of edges separating $a$ and $b$

#### 3.7.1.1 R-distance

The problem of measuring the distance between two *sets* of nodes is more complex. Ideally the two sets line up, for example "large rose" and "tall flower" is one such example where "large" can be compared with "tall" and "rose" can be compared with "flower." The problem is that it is difficult to align the concepts such that related concepts will be compared. Hence, the R-distance defined in [Rada et al., 1987] takes all of the individual entries in each set and averages the distance between all the possible combinations of the two sets.

If a document is viewed as a set of terms that are "AND"ed together, and a query is represented as a Boolean expression in disjunctive normal form, then the R-distance identifies a measure of distance between the Boolean query and the document. Also, a NOT applied to a concept yields the distance that is furthest from the concept. Hence, for a query Q for terms ((a AND b AND c) OR (e AND f)) and Document D with terms ($t_1$ AND $t_2$), the similarity is computed below.

$$c_1 = \frac{d(a,t_1) + d(a,t_2) + d(b,t_1) + d(b,t_2) + d(c,t_1) + d(c,t_2)}{6}$$

$$c_2 = \frac{d(e,t_1) + d(e,t_2) + d(f,t_1) + d(f,t_2)}{4}$$

SC(Q,D) is computed now as the MIN($c_1, c_2$). Essentially, each concept represented in the query is compared to the whole document and the similarity measure is computed as the distance between the document and the closest query concept.

Formally, the R-distance of a disjunctive normal form query Q, and a document D with terms $(t_1, t_2, \ldots, t_n)$ and $c_{ij}$, indicates the $j^{th}$ term in concept $i$ is defined as:

$$\begin{aligned} SC(Q,D) &= min\,(SC_1(c_1,D), SC_1(c_2,D), \ldots, SC_1(c_m,D)) \\ SC_1(c_i,D) &= \frac{1}{mn}\sum_{i=1}^{n}\sum_{j=1}^{m} d(t_i,c_{ij}) \\ SC(Q,D) &= 0, \texttt{ if } Q = D \end{aligned}$$

### 3.7.1.2 K-distance

A subsequent distance measure referred to as the K-distance was developed in [Kim and Kim, 1990]. This measure incorporates weighted edges in the semantic network. The distance defined between two nodes is obtained by finding the shortest path between the two nodes (again by using spreading activation) and then summing the edges along the path. More formally the distance between terms $t_i$ and $t_j$ is obtained by:

$$d_{ij} = w_{t_i,x_1} + w_{x_1,x_2} + \ldots + w_{x_n,t_j}$$

where the shortest path from $t_i$ to $t_j$ is: $t_i, x_1, x_2, \ldots, t_j$.

The authors treat NOT as a special case. Details are given in [Kim and Kim, 1990] but the basic idea is to dramatically increase the weights of the arcs that connect the node that is being referenced with a NOT (referred to as separation edges). Once this is done, any paths that include this node are much longer than any other path that includes other terms not referenced by a NOT.

To obtain the distance between two sets, A and B, of nodes with weighted arcs, the K-distance measure computes the minimum of the distances between each node in set A and set B. These minimum distances are then averaged. Since the weights on the arcs may not be equivalent in both directions, the distance measure from A to B is averaged with the distance from B to A. For our same query Q:

((a AND b AND c) OR (e AND f))

Assume document D has only two terms: ($t_1$ AND $t_2$), the similarity is computed below.

$$c_1 = \frac{min(d(a,t_1), d(a,t_2)) + min(d(b,t_1), d(b,t_2)) + min(d(c,t_1), d(c,t_2))}{3}$$

$$c_2 = \frac{min(d(e,t_1), d(e,t_2)) + min(d(f,t_1), d(f,t_2))}{2}$$

SC(Q,D) is still the min($c_1, c_2$). The value of SC(D,Q) would then be obtained, and the two coefficients are then averaged to obtain the final similarity measure.

The K-distance of a disjunctive normal form query Q and a document D with terms $(t_1, t_2, \ldots, t_n)$ is defined as:

$$SC(Q,D) = \frac{SC_1(Q,D) + SC_1(D,Q)}{2}$$

$$SC_1(Q,D) = min\left(SC_2(c_1,D), SC_2(c_2,D), \ldots, SC_2(c_m,D)\right)$$

$$SC_2(c_i, D) = \frac{1}{n}\left(\sum_{j=1}^{n} min\left(d(c_{ij}, t_j)\right)\right)$$

$$SC(Q, D) = 0, \ \ if \ \ Q = D$$

The R-distance satisfies the triangular inequality such that r-dist(a,c) is less than or equal to r-dist(a,b) + r-dist(b,c). The K-distance does not satisfy this inequality but it does make use of weights along the edges of the semantic network.

#### 3.7.1.3 Incorporating Distance

Lee, et al., incorporated a distance measure using a semantic network into the Extended Boolean Retrieval model and called it—KB-EBM for Knowledge Base—Extended Boolean Model [Lee et al., 1993]. The idea was to take the existing Extended Boolean Retrieval model described in Section 2.5 and modify the weights used to include a distance between two nodes in a semantic network.

The Extended Boolean model uses a function F that indicates the weight of a term in a document. In our earlier description we simply called it $w_i$, but technically it could be represented as F($d, t_i$). Lee, et al., modified this weight by using a semantic network and then used the rest of the Extended Boolean model without any other changes. This cleanly handled the case of NOT.

The primitive distance function, $d(t_i, t_j)$, returns the length of the shortest path between two nodes. This indicates the conceptual closeness of the two terms. What is needed here is the conceptual distance, which is inversely proportional to the primitive distance function. Hence, the new $F$ function uses:

$$distance^{-1}(t_i, t_j) = \frac{\lambda}{\lambda + distance(t_i, t_j)}$$

First, the function F is given for a document with unweighted terms. The new function, F(d, $t_i$), computes the weight of term $t_i$ in the document as the average distance of $t_i$ to all other nodes in the document. The new function $F$ is then:

$$F(d, t) = \frac{\sum_{i=1}^{n} distance^{-1}(t_i, t)}{1 + \frac{\lambda}{\lambda+1}(n-1)}$$

For existing weights for a term in a document, F is modified to include weights $w_i$. This is the weight of the $i^{th}$ term in document $d$.

$$F(d, t) = \frac{\sum_{i=1}^{n} distance^{-1}(t_i, t) w_i}{1 + \frac{\lambda}{\lambda+1}(n-1)}$$

#### 3.7.1.4 Evaluation of Distance Measures

All three distance measures were evaluated on four collections with nine, six, seven, and seven documents, respectively. Precision and recall were not measured, so evaluations were done using comparisons of the rankings produced by each distance. In some cases MESH was used—a medical semantic network—in other cases, the Computing Reviews Classification Scheme (CRCS) was used. Overall, the small size of the test collections and the lack of precision and recall measurements made it difficult to evaluate these measures. They are presented here due to their ability to use semantic networks. Most work done today is not focused on Boolean requests. However, all of these distance measures are applicable if the natural language request is viewed as a Boolean OR of the terms in the query. It would be interesting to test them against a larger collection with a general semantic network such as WordNet.

### 3.7.2 Developing Query Term Based on "Concepts"

Instead of computing the distance between query terms and document terms in a semantic network and incorporating that distance into the metric, the semantic network can be used as a thesaurus to simply replace terms in the query with "nearby" terms in the semantic network. Vectors of "concepts" can then be generated to represent the query, instead of term-based vectors. This was described in the early 1970's. In 1988, an algorithm was given that described a means of using this approach to improve an existing Boolean retrieval system [Giger, 1988]. Terms in the original Boolean system were replaced with "concepts". These concepts were found in a semantic network that contained links to the original terms. The paper referred to the network as a thesaurus, but the different relationships existing between terms meet our definition of a semantic network.

The system described in [Chen and Lynch, 1992, Chen et al., 1993] used an automatically generated semantic network. The network was developed using two different clustering algorithms. The first was the standard cosine algorithm (see Section 3.2), while the second was developed by the authors and yields asymmetric links between nodes in the semantic net. Users were then able to manually traverse the semantic network to obtain good terms for the query, while the semantic nets were also used to find suitable terms to manually index new documents.

### 3.7.3 Ranking Based on Constrained Spreading Activation

Two interesting papers appeared in 1987 that are frequently referenced in discussions of knowledge-based information retrieval [Cohen and Kjeldsen, 1987, Kjeldsen and Cohen, 1987]. These papers describe the GRANT system in which potential funding agencies are identified based on areas of research

interest. A manually built semantic network with 4,500 nodes and 700 funding agencies was constructed with links that connect agencies and areas of interest based on the topics agencies are interested in.

Given a topic, the links emanating from the topic are activated and spreading activation begins. Activation stops when a funding agency is found. At each step, activation is constrained. After following the first link, three constraints are used. The first is *distance*. If the path exceeds a length of four, it is no longer followed. The second is *fan-out*, if a path reaches a node that has more than four links emanating from it, it is not followed. This is because the node that has been reached is too general to be of much use and it will cause the search to proceed in many directions that are of little use. The third type of constraint is a rule that results in a score for the link. The score is considered an *endorsement*. Ultimately, the results are ranked based on the accumulation of these scores. An example of one such endorsement occurs if a researcher's area of interest is a subtopic or specialization of a general topic funded by the agency it gets a positive endorsement. An agency that funds research on *database systems* will fund research in *temporal database systems*. More formally:

request-funds-for-topic(x) and IS-A(x,y) → request-funds-for-topic(y)

A negative endorsement rule exists when the area of research interest is a generalization of a funding agency's areas of research. An agency that funds *database systems* will probably not be interested in funding generic interest in *computer science*.

A best-first search is used such that high-scoring endorsements are followed first. The search ends when a certain *threshold* number of funding agencies are identified. The GRANT system was tested operationally, and found to be superior to a simple keyword matching system that was in use. Searches that previously took hours could be done in minutes. More formal testing was done with a small set of twenty-three queries. However, the semantic network and the document collection were both relatively small so it is difficult to generalize from these results. Overall, the GRANT system is very interesting in that it uses a semantic network, but the network was constrained based on domain-specific rules.

## 3.8 Parsing

The ability to identify a set of tokens to represent a body of text is an essential feature of every information retrieval system. Simply using every token encountered leaves a system vulnerable to fundamental semantic mismatches between a query and a document. For instance, a query that asks for information about *computer chips* matches documents that describe *potato chips*. Simple single-token approaches, both manual and automatic, are described in

Section 3.8.1. Although these approaches seem crude and ultimately treat text as a bag of words, they generally are easy to implement, efficient, and often result in as good or better effectiveness than many sophisticated approaches measured at the Text REtrieval Conference (TREC). More discussion of TREC is found in Chapter 9).

A step up from single-term approaches is the use of phrases in document retrieval. Phrases capture some of the *meaning* behind the bag of words and result in two-term pairs (or multi-term phrases, in the general case) so that a query that requires information about *New York* will not find information about the *new Duke of York*. Section 3.8.2 describes simple approaches to phrase identification.

More sophisticated approaches to phrase identification are given in Section 3.8.3. These are based on algorithms commonly used for natural language processing (NLP). These include part-of-speech taggers, syntax parsers, and information extraction heuristics. We provide a brief overview of the heuristics that are available and pay particular attention only to those that have been directly incorporated into information retrieval systems. An entire book could be written on this section as the entire field of natural language processing is relevant.

Overall, it should be noted that parsing is critical to the performance of a system. For complex NLP approaches, parsing is discussed in great detail, but to date, these approaches have typically performed with no significant difference in performance than simplistic approaches. A review of some initial work done to integrate NLP into information retrieval systems is given in [Lewis and Sparck Jones, 1996].

### 3.8.1 Single Terms

The simplest approach to search documents is to require manual intervention and to assign names of terms to each document. The problem is that it is not always easy to assign keywords that distinctly represent a document. Also, when categorizations are employed—such as the Library of Congress subject headings—it is difficult to stay current within a domain. Needless to say, the manual effort used to categorize documents is extremely high. Therefore, it was learned early in the process that manually assigned tokens did not perform significantly better than automatically assigned tokens [Salton, 1971d].

Once scanning was deemed to be a good idea in the early 1960's, the next step was to try to normalize text to avoid simple mismatches due to differing prefixes, suffixes, or capitalization. Today, most information retrieval systems convert all text to a single case so that terms that simply start a sentence do not result in a mismatch with a query simply because they are capitalized.

*Stemming* refers to the normalization of terms by removing suffixes or prefixes. The idea is that a user who includes the term "throw" in the query might

also wish to match on "throwing", "throws", etc. Stemming algorithms have been developed for more than twenty years. The Porter and Lovins algorithms are most commonly used [Porter, 1980, Lovins, 1968]. These algorithms simply remove common suffixes and prefixes. A problem is that two very different terms might have the same stem. A stemmer that removes *-ing* and *-ed* results in a stem of *r* for terms *red* and *ring*. KSTEM uses dictionaries to ensure that any generated stem will be a valid word [Krovetz and Croft, 1989, Krovetz, 1993]. Another approach uses corpus-based statistics (essentially based on term co-occurrence) to identify stems in a language-independent fashion [Croft and Xu, 1994]. These stemmers were shown to result in improved relevance ranking over more traditional stemmers.

Stop words are terms deemed relatively meaningless in terms of document relevance and are not stored in the index. These terms represent approximately forty percent of the document collection [Francis and Kucera, 1982]. Removing these terms reduces index construction, time and storage cost, but may also reduce the ability to respond to some queries. A counterexample to the use of stop word removal occurs when a query requests a phrase that only contains stop words (e.g., "to be or not to be"). Nevertheless, stop word lists are frequently used, and some research was directed solely at determining a good stop word list [Fox, 1990].

Finally, we find that other parsing rules are employed to handle special characters. Questions arise such as what to do with special characters like hyphens, apostrophes, commas, etc. Some initial rules for these questions are given in [Adams, 1991], but the effect on precision and recall is not discussed. Many TREC papers talk about *cleaning up their parser* and the authors confess to having seen their own precision and recall results improved by very simple parsing changes. However, we are unaware of a detailed study on single-term parsing and the treatment of special characters, and its related effect on precision and recall.

### 3.8.2 Simple Phrases

Many TREC systems identify phrases as any pair of terms that are not separated by a stop term, punctuation mark, or special character. Subsequently, infrequently occurring phrases are not stored. In many TREC systems, phrases occurring fewer than 25 times are removed. This dramatically reduces the number of phrases which decreases memory requirements. [Ballerini et al., 1996].

Once phrases are employed, the question as to how they should be incorporated into the relevance ranking arises. Some systems simply add them to the query, while others do not add them to the query but do not include them in the computation of the document length normalization [Buckley et al., 1995]. The reason for this is that the terms were already being considered. Tests using just

phrases or terms were performed on many systems. It was found that phrases should be used to augment, not replace the terms. Hence, a query for *New York* should be modified to search for *new, york*, and *New York*. Phrases used in this fashion are generally accepted to yield about a ten percent improvement in precision and recall over simple terms.

### 3.8.3 Complex Phrases

The quest to employ NLP to answer a user query was undertaken since the early 1960's. In fact, NLP systems were often seen as diametrically opposed to information retrieval systems because the NLP systems were trying to *understand* a document by building a canonical structure that represents the document. The goal behind the canonical structure is to reduce the inherent ambiguity found in language. A query that asks for information about *walking* should match documents that describe people who *are moving slowly by gradually placing one foot in front of the other.*

A NLP system stores information about *walking* and *moving slowly* with the exact same canonical structure—it does this by first parsing the document syntactically—identifying the key elements of the document (subject, verb, object, etc.) and then building a single structure for the document. Simple primitives that encompass large categories of verbs were proposed [Schank, 1975] such as PTRANS (physically transport), in which *John drove to work* and *John used his car to get to work* both result in the same simple structure *John PTRANS work.*

Progress in NLP has occurred, but the reality is that many problems in knowledge representation make it extremely difficult to actually build the necessary canonical structures. The CYC project has spent the last fifteen years hand-building a knowledge base and has encountered substantial difficulty in identifying the exact means of representing the knowledge found in text [Lenat and Guha, 1989].

A side effect of full-scale NLP systems is that many tools that do not work perfectly for full language understanding are becoming quite usable for information retrieval systems. We may not be able to build a perfect knowledge representation of a document, but by using the same part-of-speech tagger and syntactic parser that might be used by an NLP system, we can develop several algorithms to identify key phrases in documents.

#### 3.8.3.1 Use of POS and Word Sense Tagging

Part-of-speech taggers are based on either statistical or rule-based methods. ;oal is to take a section of text and identify the parts of speech for each

One approach incorporates a pretagged corpus to identify two measures: the frequency a given term is assigned a particular tag and the frequency with which different tag sequences occur [Church, 1988]. For example, *duck* might

appear as a noun (creature that swims in ponds) eighty percent of a time and a verb (to get out of the way of a ball thrown at your head) twenty percent of the time. Additionally, "noun noun verb" may occur ten percent of the time while "noun noun noun" may occur thirty percent of the time. Using these two lists (generated based on a pretagged training corpus) a dynamic programming algorithm can be obtained to optimize the assignment of a tag to a token for a given step. DeRose improved on Church's initial tagger in [DeRose, 1988]. Rule-based taggers in which tags are assigned based on the firing of sequences of rules are described in [Brill, 1992].

Part-of-speech taggers can be used to identify phrases. One use is to identify all sequences of nouns such as *Virginia Beach* or sequences of adjectives followed by nouns such as *big red truck* [Allan et al., 1995, Broglio et al., 1994]. Another use of a tagger is to modify processing such that a match of a term in the query only occurs if it matches the same part-of-speech found in the document. In this fashion, *duck* as a verb does not match a reference to *duck* as a noun. Although this seems sensible, it has not been shown to be particularly effective. One reason is that words such as *bark* have many different *senses* within a part of speech. In the sentences *A dog's bark is often stronger than its bite* and *Here is a nice piece of tree bark, bark* is a noun in both cases with very different word senses. Some initial development of *word sense taggers* exists [Krovetz, 1993]. This work identifies word senses by using a dictionary-based stemmer. Recent work on sense disambiguation for acronyms is found in [Zahariev, 2004].

#### 3.8.3.2 Syntactic Parsing

As we move along the continuum of increasingly more complex NLP tools, we now discuss syntactic parsing. These tools attempt to identify the key syntactic components of a sentence, such as subject, verb, object, etc. For simple sentences the problem is not so hard. *Whales eat fish* has the simple subject of *Whales*, the verb of *eat*, and the object of *fish*. Typically, parsers work by first invoking a part-of-speech tagger.

Subsequently, a couple of different approaches are employed. One method is to apply a grammar. The first attempt at parsers used augmented transition networks (ATNs) that were essentially non-deterministic finite state automata in which: subject-verb-object would be a sequence of states. The problem is, that for complex sentences, many different paths occur through the automata. Also, some sentences recursively start the whole finite state automata (FSA), in that they contain structures that have all the individual components of a sentence. Relative clauses that occur in sentences such as *Mary, who is a nice girl that plays on the tennis team, likes seafood.* Here, the main structure of *Mary likes seafood* also has a substructure of *Mary plays tennis.* After

ATNs, rule-based approaches that attempt to parse based on firing rules, were attempted.

Other parsing algorithms, such as the Word Usage Parser (WUP) by Gomez, use a dictionary lookup for each word, and each word generates a specialized sequence of states [Gomez, 1988]. In other words, the ATN is dynamically generated based on individual word occurrences. Although this is much faster than an ATN, it requires substantial manual intervention to build the dictionary of word usages. Some parsers such as the Apple Pie Parser, are based on *light* parsing in which rules are followed to quickly scan for key elements of a sentence, but more complex sentences are not fully parsed.

Once the parse is obtained, an information retrieval system makes use of the component structures. A simple use of a parser is to use the various component phrases such as SUBJECT or OBJECT as the only components of a query and match them against the document. Phrases generated in this fashion match many variations found in English. A query with *American President* will match phrases that include *President of America*, *president who is in charge of America*, etc. One effort that identified head-modifier pairs (e.g., "America+president") was evaluated against a patent collection and demonstrated as much as a sixteen percent improvement in average precision [Osborn et al., 1997]. On the TREC-5 dataset, separate indexes based on stems, simple phrases (essentially adjective-noun pairs or noun-noun pairs), head-modifier pairs, and people name's were all separately indexed [Strzalkowski et al., 1997]. These streams were then combined and a twenty percent improvement in average precision was observed.

To date, this work has not resulted in substantial improvements in effectiveness, although it dramatically increases the run-time performance of the system.

#### 3.8.3.3 Information Extraction

The Message Understanding Conference (MUC) focuses on information extraction—the problem of finding various structured data within an unstructured document. Identification of people's names, places, amounts, etc. is the essential problem found in MUC, and numerous algorithms that attempt to solve this problem exist. Again, these are either rule-based or statistical algorithms. The first step in many of these algorithms is to generate a syntactic parse of the sentence, or at the very least, generate a part-of-speech tag. Details of these algorithms are found in the MUC Proceedings. More recently the Special Interest Group on Natural Language Learning of the Association for Computational Linguistics (CoNLL-2003) held a shared task on Language-Independent Named Entity Recognition. All of the proceedings may be found at **http://cnts.uia.ac.be/signll/conll.html**. In this task, language independent

algorithms were used to process standard test collections in English and German. We discuss these in more detail in Section 4.4.5.

Named entity taggers identify people names, organizations, and locations. We present a brief example that we created with a rule-based extractor from BBN Corporation to obtain this new document. This extractor works by using hundreds of hand-crafted rules that use surrounding terms to identify when a term should be extracted. First, we show the pre-extracted text—a paragraph about the guitarist Allen Collins.

<TEXT>
Collins began his rise to success as the lightning-fingered guitarist for the Jacksonville band formed in 1966 by a group of high school students. The band enjoyed national fame in the 1970's with such hits as "Free Bird," "Gimme Three Steps," "Saturday Night Special" and Ronnie Van Zant's feisty "Sweet Home Alabama."
</TEXT>

The following output is generated by the extractor. Tags such as PERSON and LOCATION are now marked.

<TEXT>
<ENAMEX TYPE="PERSON">Collins</ENAMEX> began his rise to success as the lightning-fingered guitarist for the <ENAMEX TYPE="LOCATION">Jacksonville</ENAMEX> band formed in <TIMEX TYPE="DATE">1966</TIMEX> by a group of high school students. The band enjoyed national fame in the <TIMEX TYPE="DATE">1970s </TIMEX> with such hits as "Free <ENAMEX TYPE="PERSON"> Bird </ENAMEX>," "Gimme Three Steps," "Saturday Night Special" and <ENAMEX TYPE="PERSON">Ronnie Van Zant</ENAMEX>'s feisty "Sweet Home <ENAMEX TYPE="LOCATION">Alabama</ENAMEX>."
</TEXT>

In this example, and in many we have hand-checked, the extractor performs well. Many extractors are now performing at much higher levels of precision and recall than those of the early 1990's [Sundheim, 1995]. However, they are not perfect. Notice the label of PERSON being assigned to the term "Bird" in the phrase "Free Bird."

Using extracted data makes it possible for a user to be shown a list of all person names, locations, and organizations that appear in the document collection. These could be used as suggested query terms for a user.

The simplest use of an extractor is to recognize key phrases in the documents. An information retrieval system could incorporate extraction by increasing term weights for extracted terms. Given that extractors are only recently running fast enough to even consider using for large volumes of text, research in the area of using extractors for information retrieval is in its infancy.

## 3.9 Summary

We described eight utilities, emphasizing that each of these utilities, both independently and in combination with each other can be integrated with any strategy. Most of these utilities address the term-mismatch problem, namely, a document can be highly relevant without having many terms that syntactically match those terms specified in the query. The relevance feedback, thesaurus, and semantic network strategies directly address this problem as they attempt to find related terms that do *match* the document and the query. Parsing and N-grams avoid mismatches by using fragments of terms instead of the actual terms. Fragmentation can avoid mismatches that can occur due to spelling errors or the loss or addition of a common prefix or suffix.

Passages attempt to focus on only the relevant part of a document. Thus, mismatching terms from spurious parts of the document are ignored and do not significantly reduce the similarity coefficient. Clustering algorithms also attempt to focus a user search onto only a relevant cluster of documents, thereby avoiding irrelevant documents.

Regression analysis estimates coefficients for a similarity measure based on a history of relevant documents. Although this does require prior relevance information, it offers an opportunity to fine tune different retrieval strategies.

Which utility is most important? Perhaps a more interesting question is: Which utility or combination of utilities work best with a given strategy? The answer to either of these questions is unclear. Relevance feedback is an accepted part of most systems participating in the TREC activities. Document clustering has exceeded most computational resources, and thus, is not used widely. Thesauri and semantic networks have yet to show dramatic improvements over a baseline comparison. Parsing plays a critical role in all information retrieval systems with much work done on various stemmers. N-grams are not as commonly used because they require substantial growth in an inverted index, but they do offer resilience to spelling errors.

The bottom line is that more testing is needed to identify which utility works best with a given strategy, and which measurements are needed to identify the extent to which a given utility improves effectiveness.

## 3.10 Exercises

1 Using the text from *Alice in Wonderland*, write some code that will use a trivial strategy for ranking documents. For a query with $i$ matching terms, assign a similarity measure of $i$ for a given document (for simplicity define a document as ten lines of the book). Implement automatic relevance feedback using this strategy to suggest ten new terms for a given query. Use *idf* as your new term sort order.

- Identify a query where five out of the ten terms are "good" in that they directly relate to the query.
- Identify a query where five of the terms are "bad".

2 Develop an example with a retrieval strategy of your choice and show how a modification to the parser will result in fundamentally different results (the document ranking will be different).

3 Implement an automatic thesaurus generation algorithm for the term *teacup* in the book. Give the top three terms most related to this term.

4 Give ten examples where stemming will do what a user would like it to do. Give ten terms where stemming will not do that a user would like.

5 One idea to improve effectiveness of an information retrieval system is to match on both the term and the *sense* of the term. The idea is that for a query of the term *duck* as noun, a document containing "She tried to duck to avoid the ball thrown at her." would not match. Implement five queries with your favorite Web search engine, and for each query identify a document that could have been avoided using this heuristic.

Chapter 4

# CROSS-LANGUAGE INFORMATION RETRIEVAL

Cross-Language Information Retrieval (CLIR) is quickly becoming a mature area in the information retrieval world. The goal is to allow a user to issue a query in language L and have that query retrieve documents in language $L'$ (see Figure 4.1). The idea is that the user wants to issue a single query against a document collection that contains documents in a myriad of languages. An implicit assumption is that the user understands results obtained in multiple languages. If this is not the case, it is necessary for the retrieval system to translate the selected foreign language documents into a language that the user can understand. Surveys of cross-language information retrieval techniques and multilingual processing include [Oard and Diekema, 1998, Haddouti, 1999].

## 4.1 Introduction

The key difference between CLIR and monolingual information retrieval is that the query and the documents cannot be matched directly. In addition to the inherent difficulty in matching the inherent style, tone, word usage, and other features of the query with that of the document, we must now cross the language barrier between the query and the document. Section 4.2 focuses on the core problems involved in crossing the language barrier. Section 4.3 describes cross-language retrieval strategies and Section 4.4 discusses cross-language utilities.

### 4.1.1 Resources

Numerous resources are needed to implement cross-language retrieval systems. 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 happen to have been written in different languages, but the documents are not literal translations of each other. A news article in language L by a newspaper in a country which speaks language L and an article in language $L'$ by a newspaper in a country which speaks language $L'$ is an example of comparable documents. The two newspapers wrote their own article; they did not translate an article in one language into another language. Another key with comparable corpora are that they must be *about the same topic*. A book in French on medicine and a book in Spanish on law are not comparable. If both books are about medicine or about law they are comparable. We will discuss CLIR techniques using a comparable corpus in Section 4.3.3.

A *parallel corpus* provides documents in one language L that are then direct translations of language $L'$ or vice versa. The key is that each document is in language L is a direct translation of a corresponding document in language $L'$. Hence, it is possible to align a parallel corpus at the document level, the paragraph level, the sentence level, the phrase level, or even the individual term level. Legislative documents in countries or organizations that are required to publish their proceedings in at least two languages are a common source of parallel corpora. In general, a parallel corpus will be most useful if it is used to implement cross-language retrieval of documents that are in a similar domain to the parallel corpus. Recent work shows that significant effectiveness can be obtained if the correct domain is selected [Rogati and Yang, 2004]. We discuss parallel corpus CLIR techniques in Section 4.3.2.1.

We also note that even within a single language such as Arabic, there are many different character sets (four are commonly used with Arabic). Language processing resources exist to not only detect a language but also to detect a character set. Cross-language systems often struggle with intricacies involved in working with different character sets within a single language. Unicode (**www.unicode.org**) was developed to map the character representation for numerous scripts into a single character set, but not all electronic documents are currently stored in Unicode.

### 4.1.2 Evaluation

Different measures are used to evaluate the performance of cross-language information retrieval systems. The most obvious is simply to compute the average precision of the cross-language query.

Another approach is to compute the percentage of monolingual performance. This can occasionally be misleading because the techniques used to achieve a given monolingual performance may be quite different than those used for cross-language performance. Straightforward techniques typically result in 50% of monolingual performance, but the CLIR literature contains results that

exceed 100% because of the inherent query expansion that occurs when doing a translation [Levow et al., 2004].

We note that queries with relevance judgements exist in Arabic, Chinese, Dutch, Finnish, French, German, Italian, Japanese, Korean, Swedish and Spanish. These have been used at various evaluations at TREC (Text REtrieval Conference, see **trec.nist.gov**), CLEF (Cross-Language Evaluation Forum see **clef.iei.pi.cnr.it**), and NTCIR (see **research.nii.ac.jp/ntcir**).

## 4.2 Crossing the Language Barrier

To cross the language barrier, we must answer four core questions [Oard, 2004]:

- What should be translated? Either the queries may be translated, the documents, or both queries and documents may be translated to some internal representation. Query translation is depicted in Figure 4.1). Section 4.2.1 describes query translation approaches. Document translation is shown in Figure 4.2). Section 4.2.2 describes document translation. Finally, we may obtain an internal representation of both the query and the document.

- Which tokens should be used to do a translation (e.g.; stems, words, phrases, etc.)? Phrase translation is described in Section 4.2.3.

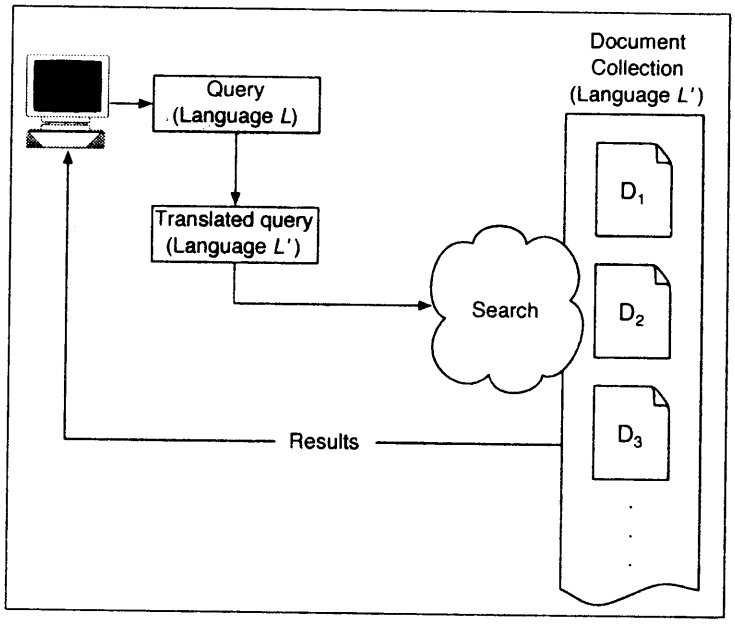
- How should we use a translation? In other words, a single term in language L may map to several terms in Language L/. We may use one of these terms, some of these terms, or all of these terms. Additionally, we might weight some terms higher than other terms if we have reason to believe that one translation is more likely than another. Various approaches are described in described in Section 4.2.4.

- How can we remove spurious translations? Typically, there are spurious translations that can lead to poor retrieval. Techniques exist to remove these translations. Pruning translations is described in Section 4.2.5.

We describe several query translation approaches in Section 4.2.1. Translating all of the documents into the language of the query is an arduous task if a complex machine translation system that does full natural language processing is used (lighter translation approaches are certainly viable, but intuitively appear to be less effective).

As with monolingual retrieval, various strategies and utilities exist for cross-language retrieval. As with the monolingual retrieval we organize the chapter according to strategies and utilities. Again, a strategy will take a query in language L and identify a measure of similarity between the query and documents in the target language $L'$. A utility enhances the work of any strategy. The core

*Figure 4.1.* Translate the Query




strategies: query translation, document translation, and use of internal representation are the focus of Sections 4.2.1, 4.2.2, and 4.3.2. Utilities such as $n$-grams, stemming, and entity tagging are described in Section 4.4.

### 4.2.1 Query Translation

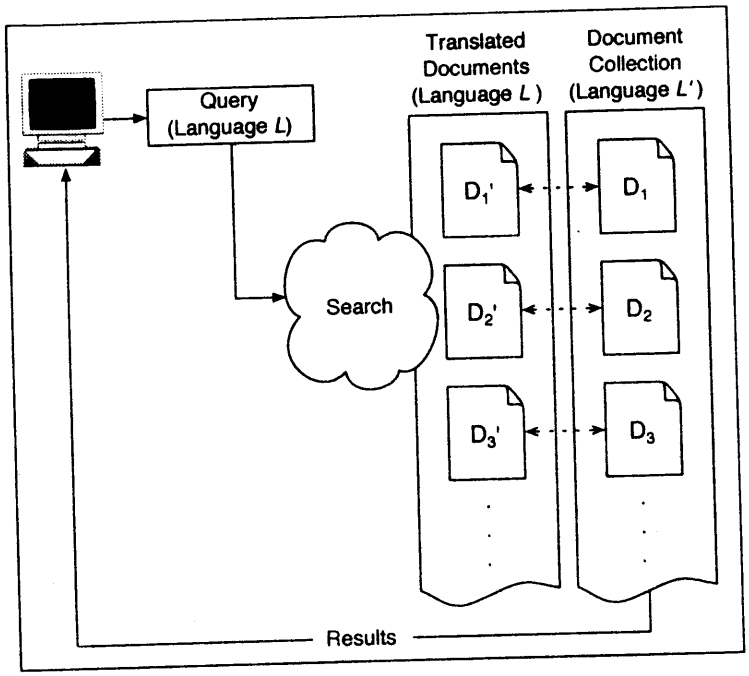
Initial work in query translation was done in the early 1970's where user specified keywords were used to represent documents and a dictionary was used to translate English keywords to German keywords [Salton, 1970a]. Query translation approaches use machine translation, language specific stemmers, dictionaries, thesauri, and automatically generated bilingual term lists to implement the necessary translation of the user query in language L to the target query language $L'$.

An excellent discussion of dictionary-based query translation techniques is given in [Levow et al., 2004]. In this work, a methodical study of various techniques is presented and effectiveness of specific techniques are measured.

### 4.2.2 Document Translation

A simple way to "translate" documents is to use any of the query translation approaches that we have already discussed. For example, one or more bilingual

*Figure 4.2.* Translate the Documents




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 $L'$. Although machine translation algorithms are not perfect, they have been used as a foundation for CLIR. An advantage of these algorithms is that the use of the full natural language text provides more evidence when selecting a potential translation.

### 4.2.3 Phrase Translation

Instead of simply using terms for translation, phrase-based approaches were shown to yield substantial improvement. Simply manually translating phrases instead of a term based translation was shown to yield improvement in [Hull and Grefenstette, 1996]. A bilingual phrase list extracted from a parallel text was used in [Davis and Ogden, 1997]. We discuss extraction of bilingual term lists from training texts in Section 4.3.2. Phrase translation using the Collins machine readable dictionary was described in [Ballesteros and Croft, 1997]. A core problem with any phrase translation approach is that there is far more

likelihood that a phrase will not be found in a bilingual list used for translation than a single term.

### 4.2.4 Choosing Translations

Once a bilingual term list or term lists are in place, the next challenge is to choose the appropriate translation. We have discussed dictionary coverage issues associated with query translation. We now focus on the choice of translations and the need to form these translations into a new query for the target language.

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

#### 4.2.4.1 Quality of Bilingual Term Lists

Like their monolingual counterparts, most cross-language systems use a variety of techniques to improve accuracy. At the core of these systems are bilingual lexicons. A survey of these approaches is given in [Pirkola et al., 2001]. 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, and the lack of even the existence of a machine-readable source.

Given that word sense disambiguation is very difficult with the short length and lack of context found in a typical query, multiple word senses for a single term result in numerous translations. 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 to *move out of the way*. The challenge of cross-language systems is to reduce the ambiguity created by a translated query in the target language that has numerous terms that have nothing to do with the query.

Experiments which focus on measuring the effect of coverage of a dictionary on effectiveness are found in [Demner-Fushman and Oard, 2003]. In this work, a set of 113,000 documents was used with thirty-three queries. Thirty-five different bilingual dictionaries were tested. In each case, the original English query was filtered based on the bilingual dictionary. Only terms found in the dictionary were used. For each dictionary, the average precision was computed based on the ability of that dictionary to filter a query. The idea is that a perfect dictionary would not filter at all. It was discovered that dictionary size linearly increases accuracy when a dictionary contains between 3,000 and 20,000 terms. After that, mean average precision does not increase.

Another key set of experiments on lexicon coverage is given in [McNamee and Mayfield, 2002a]. For a fixed translation resource, they gradually reduce the number of translations found in the resource by randomly eliminat-

ing some terms that are translated. This was done for five languages: Dutch, English, French, German, Italian, and Spanish. They find, what one would intuitively expect, performance degrades as lexical coverage degrades. (see Section 4.4.1).

#### 4.2.4.2 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 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 than the more general term – this is precisely the opposite of what would be preferred – increased weighting of more specific search terms.

#### 4.2.4.3 Balanced Queries

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

More formally, assume we have a weight $w_{ij}$ for a term $t_i$ in document $D_j$. As usual, the term frequency $tf_{ij}$ of the this term indicates the number of occurrences of term $i$ in document $j$. Let $|D_j|$ indicate the length of document $j$. Let $df_i$ equal the number of documents in the collection that contain term $i$. Our weight $w_{ij}$ will be computed with a function that combines the $tf$, the $df$ and the size of the document $|D|$. We have discussed both normalized cosine measures (see Section 2.1.2) and probabilistic measures (see Section 2.2). These measures may be thought of as term weighting functions that are increasing in $tf_{ij}$, decreasing in $df_i$ and decreasing in $|D_j|$.

To compute the weight of query term $q_i$ in language L. Assume we have $k$ translations of term $q_i$ in language $L'$. For each of these translations $t'_i$, we can compute the $tf'_{ij}$ and $|D'_i|$ for each document in language $L'$. Similarly, we can compute the collection frequency, $df'_i$ in language $L'$. From these we can compute the weight $w'_i$ for each of the translations in language $L'$. To compute the weight of our initial query term in language $L$ we simply average the corresponding translations as:

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

Balanced queries are described in more detail in [Leek et al., 2000, Levow and Oard, 2000].

#### 4.2.4.4 Structured Queries

For structured queries, the overall term frequency for a term in a document is computed as the sum of all of the term frequencies for all of the translations. Similarly, the corresponding document frequencies for all translations of a query term are combined as well. Assume we again have $k$ translations for query term $q_i$:

$$tf_i = \sum_{i=1}^{k} tf'_{ij} \qquad df_i = \sum_{i=1}^{k} df'i$$

These modified term and document weights are then used to compute $w_i$. Any translation for a give term contributes to the document frequency of that term. Structured queries are described in more detail in [Pirkola, 1998]. A recent study compared these approaches and found that structured queries significantly outperformed balanced and unbalanced queries [Levow et al., 2004].

#### 4.2.4.5 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 initial discussion with research results is found in [Ballesteros, 2001]. Here, the use of a pivot language was shown to degrade performance by 91% over a direct bilingual translation. This initial degradation was overcome with relevance feedback techniques (see Section 4.4.1).

An approach which uses two different pivot languages is given in [Gollins and Sanderson, 2004]. In this work, 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. The idea is that translations lost in converting from German to Spanish are improved with those from German to Dutch. Effectiveness was improved when *only* the terms in the target language that were obtained via all translation routes were used.

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. More recent results with the use of a pivot language are found in [Kishida and Kando, 2003].

### 4.2.5 Pruning Translations

Once translations are identified, it is often necessary to prune the list of translations for a given phrase. Instead of using all translations, only candidate translations that tend to co-occur are used in [Adriani, 2000].

Recently, work done with bidirectional English-Arabic dictionaries showed that performance improves *only* when translations, $t$, are used in which a term $s$ is translated to $t$ and then back to $s$ using a bidirectional English-Arabic dictionary [Aljlayl et al., 2002].

The use of HMM's to compute a most likely query translation was shown in [Federico and Bertoldi, 2002]. The most probable translation is used and the result is combined with a more traditional language model (as we just described in Section 4.3.1). The work also used an applied linear smoothing technique as opposed to smoothing approaches described in Section 2.3.2. The authors suggest that their approach is superior to the similarity measure given in Equation 4.1, but we are unaware of any direct comparisons. The authors experimented with using different numbers of query translations. They found, in many cases, a single high quality translation of the query outperformed a variety of translations.

## 4.3 Cross-Language Retrieval Strategies

Cross-language retrieval strategies develop a similarity measure for a query in language L against documents in Language L/. Section 4.3.1 describes the use of Language Models for CLIR.

### 4.3.1 Language Models for CLIR

A language modeling (see Section 2.3) approach may be used for cross-language retrieval. To briefly review, a simple monolingual language model will compute the likelihood of generating a query given a document.

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

Given a query $(t_1, t_2, \ldots, t_j, \ldots, t_{|Q|})$ where $t_j$ is a query term; $\alpha$ is estimated using an EM (Expectation Maximization) algorithm [Dempster et al., 1977], and $C$ is the document collection. While formulated as a Hidden Markov Model (HMM), the retrieval method is essentially similar to using the multinomial language model with linear interpolation smoothing (see Section 2.3.2). A good overview on Hidden Markov Models may be found in [Rabbiner, 1989].

The remaining probabilities are computed as:

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

where $f_j$ is the frequency or number of occurrences of term $j$ in the document collection.

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

where $tf_{ij}$ is the term frequency or number of occurrences of term $j$ in document $i$. For cross-language retrieval, the model is extended so a document written in language L generates a query in language $L'$ [Xu and Weischedel, 1999]. The notation $t_L$ is used to refer to a term in language $L$ and $t_{L'}$ is used to represent a term in language $L'$. Similarly, $D_{L_i}$ refers to the document $i$ in language L and $D_{L'_i}$ refers to document $i$ in language $L'$. Hence, we need to compute:

$$P(q_L|D_{L'_i} \in R) = \prod_{j=1}^{Q} \alpha P(t_{L_j}|C_L) + (1-\alpha)P(t_{L_j}|D_{L'_i}) \tag{4.1}$$

We now compute the probability that a query in language L will be generated with a document in language $L'$. All of the probabilities can be computed with no change except for $P(t_{L_j}|D_{L'})$. This probability can be computed with a term dictionary that will enable us to compute a translation probability: $P(t_{L_j}|t_{L'_k})$.

$$P(t_{L_j}|D_{L'_i}) = \sum_{k=1}^{|D_{L'}|} P(t_{L'_k}|D_{L'_i})\ P(t_{L_j}|t_{L'_k}) \tag{4.2}$$

where the first term is the ratio of our translated query terms to the size of the document retrieved.

$$P(t_{L'_k}|D_{L'_i}) = \frac{tf_{L'_k}}{|D_{L'_i}|}$$

The next term uses the actual translations in the lexicon. The probability $P(t_{L_j}|t_{L'_k})$ is the *translation probability* that given a term in language $L'$ there will be a term in language $L$. For simplicity, one might assign these probabilities as equally likely so that if a term in language L has five translations in language $L'$ we might assign them each a probability of 0.2. However, this framework allows for the development of other algorithms to estimate the translation probability [Brown et al., 1993]. If a parallel corpus exists (see Section 4.3.2.1) that corpus can be used to estimate the translation probability. A linear combination of three different sources for estimating this probability (e.g; two dictionaries and a bilingual term list generated from a parallel corpus) was used in [Xu et al., 2001].

*Table 4.1.* Raw Term Frequency for English Collection

| $tf_{t,d}$ | a | arrived | damaged | delivery | fire | gold | in | of | shipment | silver | truck |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $D_{e_1}$ | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| $D_{e_2}$ | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 2 | 1 |
| $D_{e_3}$ | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 |

*Table 4.2.* Raw Term Frequency for Spanish Collection

| $tf_{t,d}$ | camión | dañó | de | del | el | en | entrega | envío | fuego | la | llegó | oro | plata | un |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $D_{s_1}$ | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 |
| $D_{s_2}$ | 1 | 0 | 2 | 0 | 0 | 1 | 1 | 0 | 0 | 2 | 1 | 0 | 2 | 1 |
| $D_{s_3}$ | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 |

*Table 4.3.* $P(t|C_e)$ for All Terms in the English Collection

| a | arrived | damaged | delivery | fire | gold | in | of | shipment | silver | truck |
|---|---|---|---|---|---|---|---|---|---|---|
| $\frac{3}{22}$ | $\frac{2}{22}$ | $\frac{1}{22}$ | $\frac{1}{22}$ | $\frac{1}{22}$ | $\frac{2}{22}$ | $\frac{3}{22}$ | $\frac{3}{22}$ | $\frac{2}{22}$ | $\frac{2}{22}$ | $\frac{2}{22}$ |

#### 4.3.1.1 Example

Returning to our example from Chapter 2, we show how 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 use of two online bilingual English-Spanish dictionaries.

Document Collection:

$D_{e_1}$ : "Shipment of gold damaged in a fire."
$D_{e_2}$ : "Delivery of silver arrived in a silver truck."
$D_{e_3}$ : "Shipment of gold arrived in a truck."

$D_{s_1}$ : El envío del oro dañó en un fuego
$D_{s_2}$ : La entrega de la plata llegó en un camión de plata
$D_{s_3}$ : El envío del oro llegó en un camión

To find the document in the Spanish collection that is most relevant to the English query "gold silver truck", we need to find the translation ratios of "gold",

*Table 4.4.* $P(t|C_s)$ for All Terms in the Spanish Collection

| camión | dañó | de | del | el | en | entrega | envío | fuego | la | llegó | oro | plata | un |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\frac{2}{27}$ | $\frac{1}{27}$ | $\frac{2}{27}$ | $\frac{2}{27}$ | $\frac{2}{27}$ | $\frac{3}{27}$ | $\frac{1}{27}$ | $\frac{2}{27}$ | $\frac{1}{27}$ | $\frac{2}{27}$ | $\frac{2}{27}$ | $\frac{2}{27}$ | $\frac{2}{27}$ | $\frac{3}{27}$ |

*Table 4.5.* Term Frequency for Spanish Collection

| $tf_{t,d}$ | camión | dañó | de | del | el | en | entrega | envío | fuego | la | llegó | oro | plata | un |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\frac{tf}{D_{s1}}$ | 0 | $\frac{1}{8}$ | 0 | $\frac{1}{8}$ | $\frac{1}{8}$ | $\frac{1}{8}$ | 0 | $\frac{1}{8}$ | $\frac{1}{8}$ | 0 | 0 | $\frac{1}{8}$ | 0 | $\frac{1}{8}$ |
| $\frac{tf}{D_{s2}}$ | $\frac{1}{11}$ | 0 | $\frac{2}{11}$ | 0 | 0 | $\frac{1}{11}$ | $\frac{1}{11}$ | 0 | 0 | $\frac{2}{11}$ | $\frac{1}{11}$ | 0 | $\frac{2}{11}$ | $\frac{1}{11}$ |
| $\frac{tf}{D_{s3}}$ | $\frac{1}{8}$ | 0 | 0 | $\frac{1}{8}$ | $\frac{1}{8}$ | $\frac{1}{8}$ | 0 | $\frac{1}{8}$ | 0 | 0 | $\frac{1}{8}$ | $\frac{1}{8}$ | 0 | $\frac{1}{8}$ |

*Table 4.6.* Translation Probabilities: $P(t_{L_j} \mid t_{L'_k})$

| | camión | dañó | de | del | el | en | entrega | envío | fuego | la | llegó | oro | plata | un |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| gold | $\frac{0}{2}$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | $\frac{0}{1}$ | 0 |
| silver | $\frac{0}{2}$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | $\frac{0}{1}$ | 1 | 0 |
| truck | $\frac{1}{2}$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | $\frac{0}{1}$ | $\frac{0}{1}$ | 0 |

"silver", and "truck". Using Equation 4.2 we obtain the probability of a query term appearing in a document from the Spanish collection.

$$
\begin{aligned}
&P(gold \mid D_{s_1}) = \\
&\quad P(El \mid D_{s_1}) \cdot P(gold \mid El) + P(envío \mid D_{s_1}) \cdot P(gold \mid envío) + \\
&\quad P(del \mid D_{s_1}) \cdot P(gold \mid del) + P(oro \mid D_{s_1}) \cdot P(gold \mid oro) + \\
&\quad P(dañó \mid D_{s_1}) \cdot P(gold \mid dañó) + P(en \mid D_{s_1}) \cdot P(gold \mid en) + \\
&\quad P(un \mid D_{s_1}) \cdot P(gold \mid un) + P(fuego \mid D_{s_1}) \cdot P(gold \mid fuego) \\
&\quad = \tfrac{1}{8} \cdot \tfrac{1}{1} = \tfrac{1}{8}
\end{aligned}
$$

$$
\begin{aligned}
&P(gold \mid D_{s_2}) = \\
&\quad P(La \mid D_{s_2}) \cdot P(gold \mid La) + P(entrega \mid D_{s_2}) \cdot P(gold \mid entrega) + \\
&\quad P(de \mid D_{s_2}) \cdot P(gold \mid de) + P(la \mid D_{s_2}) \cdot P(gold \mid la) + \\
&\quad P(plata \mid D_{s_2}) \cdot P(gold \mid plata) + P(llegó \mid D_{s_2}) \cdot P(gold \mid llegó) + \\
&\quad P(en \mid D_{s_2}) \cdot P(gold \mid en) + P(un \mid D_{s_2}) \cdot P(gold \mid un) + \\
&\quad P(camión \mid D_{s_2}) \cdot P(gold \mid camión) \\
&\quad = 0
\end{aligned}
$$

$$
\begin{aligned}
&P(gold \mid D_{s_3}) = \\
&\quad P(El \mid D_{s_3}) \cdot P(gold \mid El) + P(envío \mid D_{s_3}) \cdot P(gold \mid envío) + \\
&\quad P(del \mid D_{s_3}) \cdot P(gold \mid del) + P(oro \mid D_{s_3}) \cdot P(gold \mid oro) + \\
&\quad P(llegó \mid D_{s_3}) \cdot P(gold \mid llegó) + P(en \mid D_{s_3}) \cdot P(gold \mid en) + \\
&\quad P(un \mid D_{s_3}) \cdot P(gold \mid un) + P(camión \mid D_{s_3}) \cdot P(gold \mid camión) \\
&\quad = \tfrac{1}{8} \cdot \tfrac{1}{1} \\
&\quad = \tfrac{1}{8}
\end{aligned}
$$

$$
\begin{aligned}
&P(silver \mid D_{s_1}) = \\
&\quad P(El \mid D_{s_1}) \cdot P(silver \mid El) + P(envío \mid D_{s_1}) \cdot P(silver \mid envío) + \\
&\quad P(del \mid D_{s_1}) \cdot P(silver \mid del) + P(oro \mid D_{s_1}) \cdot P(silver \mid oro) + \\
&\quad P(dañó \mid D_{s_1}) \cdot P(silver \mid dañó) + P(en \mid D_{s_1}) \cdot P(silver \mid en) + \\
&\quad P(un \mid D_{s_1}) \cdot P(silver \mid un) + P(fuego \mid D_{s_1}) \cdot P(silver \mid fuego) \\
&\quad = 0
\end{aligned}
$$

$$
\begin{aligned}
&P(silver \mid D_{s_2}) = \\
&\quad P(La \mid D_{s_2}) \cdot P(silver \mid La) + P(entrega \mid D_{s_2}) \cdot P(silver \mid entrega) + \\
&\quad P(de \mid D_{s_2}) \cdot P(silver \mid de) + P(la \mid D_{s_2}) \cdot P(silver \mid la) + \\
&\quad P(plata \mid D_{s_2}) \cdot P(silver \mid plata) + P(llegó \mid D_{s_2}) \cdot P(silver \mid llegó) + \\
&\quad P(en \mid D_{s_2}) \cdot P(silver \mid en) + P(un \mid D_{s_2}) \cdot P(silver \mid un) + \\
&\quad P(camión \mid D_{s_2}) \cdot P(silver \mid camión) \\
&\quad = \tfrac{2}{11} \cdot \tfrac{1}{1} \\
&\quad = \tfrac{2}{11}
\end{aligned}
$$

$$
\begin{aligned}
&P(silver \mid D_{s_3}) = \\
&\quad P(El \mid D_{s_3}) \cdot P(silver \mid El) + P(envío \mid D_{s_3}) \cdot P(silver \mid envío) + \\
&\quad P(del \mid D_{s_3}) \cdot P(silver \mid del) + P(oro \mid D_{s_3}) \cdot P(silver \mid oro) + \\
&\quad P(llegó \mid D_{s_3}) \cdot P(silver \mid llegó) + P(en \mid D_{s_3}) \cdot P(silver \mid en) + \\
&\quad P(un \mid D_{s_3}) \cdot P(silver \mid un) + P(camión \mid D_{s_3}) \cdot P(silver \mid camión) \\
&\quad = 0
\end{aligned}
$$

$$
\begin{aligned}
&P(truck \mid D_{s_1}) = \\
&\quad P(El \mid D_{s_1}) \cdot P(truck \mid El) + P(envío \mid D_{s_1}) \cdot P(truck \mid envío) + \\
&\quad P(del \mid D_{s_1}) \cdot P(truck \mid del) + P(oro \mid D_{s_1}) \cdot P(truck \mid oro) + \\
&\quad P(dañó \mid D_{s_1}) \cdot P(truck \mid dañó) + P(en \mid D_{s_1}) \cdot P(truck \mid en) + \\
&\quad P(un \mid D_{s_1}) \cdot P(truck \mid un) + P(fuego \mid D_{s_1}) \cdot P(truck \mid fuego) \\
&\quad = 0
\end{aligned}
$$

$P(truck \mid D_{s_2}) =$
$P(La \mid D_{s_2}) \cdot P(truck \mid La) + P(entrega \mid D_{s_2}) \cdot P(truck \mid entrega) +$
$P(de \mid D_{s_2}) \cdot P(truck \mid de) + P(la \mid D_{s_2}) \cdot P(truck \mid la) +$
$P(plata \mid D_{s_2}) \cdot P(truck \mid plata) + P(llegó \mid D_{s_2}) \cdot P(truck \mid llegó) +$
$P(en \mid D_{s_2}) \cdot P(truck \mid en) + P(un \mid D_{s_2}) \cdot P(truck \mid un) +$
$P(camión \mid D_{s_2}) \cdot P(truck \mid camión)$
$= \frac{1}{11} \cdot \frac{1}{2}$
$= \frac{1}{22}$

$P(truck \mid D_{s_3}) =$
$P(El \mid D_{s_3}) \cdot P(truck \mid El) + P(envío \mid D_{s_3}) \cdot P(truck \mid envío) +$
$P(del \mid D_{s_3}) \cdot P(truck \mid del) + P(oro \mid D_{s_3}) \cdot P(truck \mid oro) +$
$P(llegó \mid D_{s_3}) \cdot P(truck \mid llegó) + P(en \mid D_{s_3}) \cdot P(truck \mid en) +$
$P(un \mid D_{s_3}) \cdot P(truck \mid un) + P(camión \mid D_{s_3}) \cdot P(truck \mid camión)$
$= \frac{1}{8} \cdot \frac{1}{2}$
$= \frac{1}{16}$

Now we apply Equation 4.1 to find the probability of the query appearing in each Spanish document. Here $\alpha = 0.3$.

$P(gold, silver, truck \mid D_{s_1}) =$
$[(0.3)P(gold \mid C_e) + (1 - 0.3)P(gold \mid D_{s_1})] \times$
$[(0.3)P(silver \mid C_e) + (1 - 0.3)P(silver \mid D_{s_1})] \times$
$[(0.3)P(truck \mid C_e) + (1 - 0.3)P(truck \mid D_{s_1})]$
$= (0.3 \cdot \frac{2}{22} + 0.7 \cdot \frac{1}{8})(0.3 \cdot \frac{2}{22} + 0.7 \cdot 0)(0.3 \cdot \frac{2}{22} + 0.7 \cdot 0)$
$= 0.0000854$

$P(gold, silver, truck \mid D_{s_2}) =$
$[(0.3)P(gold \mid C_e) + (1 - 0.3)P(gold \mid D_{s_2})] \times$
$[(0.3)P(silver \mid C_e) + (1 - 0.3)P(silver \mid D_{s_2})] \times$
$[(0.3)P(truck \mid C_e) + (1 - 0.3)P(truck \mid D_{s_2})]$
$= (0.3 \cdot \frac{2}{22} + 0.7 \cdot 0)(0.3 \cdot \frac{2}{22} + 0.7 \cdot \frac{2}{11})(0.3 \cdot \frac{2}{22} + 0.7 \cdot \frac{1}{22})$
$= 0.0002491$

$P(gold, silver, truck \mid D_{s_3}) =$
$[(0.3)P(gold \mid C_e) + (1 - 0.3)P(gold \mid D_{s_3})] \times$
$[(0.3)P(silver|C_e) + (1 - 0.3)P(silver \mid D_{s_3})] \times$
$[(0.3)P(truck|C_e) + (1 - 0.3)P(truck \mid D_{s_3})]$
$= (0.3 \cdot \frac{2}{22} + 0.7 \cdot \frac{1}{8})(0.3 \cdot \frac{2}{22} + 0.7 \cdot 0)(0.3 \cdot \frac{2}{22} + 0.7 \cdot \frac{1}{16})$
$= 0.0002223$

$D_{s_2}$ gives the highest probability, therefore, it is most likely to be relevant to the query.

### 4.3.2 Bilingual Corpus Strategies

Logically, cross-language information retrieval systems should benefit from having *similar* document collections in both language L and $L'$. It is trivial to find documents in both L and $L'$, but for cross-language information retrieval techniques to work well, it is necessary to find documents that, at the very least, describe the *similar* content. A book translated into both language L and $L'$ is an example of a bilingual corpus of documents. In other cases, we might only have a book in language L and another book on an entirely different topic in $L'$. Numerous cross-language information retrieval techniques were developed assuming that, at some level, a bilingual corpus exists. Even for obscure languages there are often parallel translations of some texts (e.g.; religious texts) which might be sufficient to build a useful bilingual term list. Additionally, the European Union is legally required to publish decisions regarding patents in multiple languages.

#### 4.3.2.1 Parallel

In the world of machine translation, parallel corpora have been used since the early 1990's to identify potential translations for a given word. The first step is to match up sentences in the two corpora. This is known as *sentence alignment.* One of the first papers that described sentence alignment for parallel corpora to generate translation probabilities is [Brown et al., 1990]. The general idea is to take a document in language L and another document which is a translation of L in language $L'$ and *align* the two texts. The problem is non-trivial as one sentence in L can map to many sentences in $L'$. The alignment can be at the sentence, paragraph, or document level. Initial approaches for aligning corpora used dynamic programming algorithms based on either the number of words [Brown et al., 1991] in a sentence or the number of characters in the sentence [Gale and Church, 1991].

A bilingual term lexicon can be used to assist in the alignment process (e.g.; if we see a sentence with the word *peace* in English we might align it with the word *shalom* in Hebrew). We note that one use of aligned corpora is to generate a bilingual term lexicon, so in cases of more obscure languages, such a lexicon might not exist.

The problem can be expressed as a means of estimating the translation probability $P(f|e)$ where $e$ is a word in English and $f$ is a word in French. Five models for estimating this probability are given in [Brown et al., 1993]. These are referred to frequently as IBM Model 1, 2, 3, 4, and 5. In model one, word order is not included. Model two uses the word order. Model three includes the length of terms being matched. Models four and five include the probability of a given connection between the English word and the French word.

Later, an approach using HMM's was given in [Vogel et al., 1996]. Software to easily implement these algorithms was not widely available until a workshop was held with the stated purpose of building and disseminating such tools [Al-Onaizan et al., 1999]. At this workshop a program named GIZA was developed and it included IBM Models 1, 2, and 3. Later, a new version GIZA++ was implemented which contained all five IBM models, Vogel's HMM-based model, as well as a variety of other improvements. Today GIZA++ is widely used by CLIR applications that wish to generate a bilingual term list from a parallel corpus of two different languages. Details of GIZA++ as well as comparative results using the different models are given in [Och and Ney, 2000b, Och and Ney, 2000a, Och and Ney, 2003].

An algorithm for using this technique to align English and Norwegian sentences is described in [Hofland, 1996]. A survey of alignment techniques is given in [Somers, 2001, Veronis, 2000]. An approach to building a parallel corpus by starting with a small collection and gradually adding to it with translation models is described in [Callison-Burch and Osborne, 2003].

#### 4.3.2.2 Comparable

Comparable corpora are two collections that are about the same basic content but the documents are not translations of one another. The cross-language information retrieval techniques we discuss in our comparable corpora section assume that document A in language L corresponds to document B in language $L'$.

### 4.3.3 Comparable Corpora Strategies

Comparable corpora exist when documents in one language are about the same topic, but are not translations. Many cross-language information retrieval techniques focus on this type of a collection since it is more likely to have a comparable corpus between language pairs than to obtain a parallel corpus.

We note that when the entire document collection exists in language L and a comparable collection exists in $L'$ a simple approach for cross-language information retrieval can be used. The query can be executed in language L and results are obtained. The resulting documents are mapped to their comparable twins in language $L'$. A document list in language $L'$ is then returned to the user. In most cases, the comparable corpus is simply a subset of training documents which can be used to facilitate cross-language retrieval of larger document collections.

#### 4.3.3.1 Extraction of Bilingual Term Lists

Early work with comparable corpora appeared in [Sheridan and Ballerini, 1996]. In this work, documents are aligned by date and by a language inde-

pendent descriptor (e.g.; ".mil" indicated that the document was about military issues). In subsequent work, proper nouns in documents were also used to facilitate alignment [Sheridan et al., 1997]. This was done as it was hoped that proper nouns would be somewhat language independent. From this alignment, a bilingual term list was constructed and used as a basis for query translation. A term-term similarity function is then defined, which yields, for a given term, other terms that are similar. This function is used for a query in language L to find terms in language $L'$.

Ballesteros and Croft also used co-occurrence to reduce ambiguity by using the assumption that good translations should co-occur in target language documents while incorrect translations should not co-occur [Ballesteros and Croft, 1997, Ballesteros and Croft, 1998]. An iterative proportional fitting procedure is used to gradually compute a probability that a given term is a translation of another term. This probability is used to choose among the candidate translations.

Another approach uses a vector to represent a target word [Fung, 1998]. This context vector, $T$ is populated by using the words that surround it. The approach resembles those used to automatically generate a monolingual term list (see Section 3.6). The context words are then translated to source terms using a manually constructed bilingual term list. Let us call the new vector $S'$. The idea is that at least some of these words will exist in the bilingual term list. Once this is done, context vectors for the source terms are computed based on surrounding terms; we refer to this vector as $S$. Now, one or more translations are identified simply by using a cosine similarity measure between S and $T$. This technique was applied in a recent Cross-Language Evaluation Forum (CLEF) submission [Cancedda et al., 2003].

#### 4.3.3.2 Hidden Markov Models

In Section 4.2.5, we described how Hidden Markov Models can be used to incorporate a bilingual term dictionary into cross-language retrieval. With a comparable corpus, this approach can be modified to use a more accurate *translation probability*. Instead of a uniform translation probability, the frequency of occurrence in the comparable corpus can be used. Accuracy was improved in [Federico and Bertoldi, 2002, Bertoldi and Federico, 2003] when probabilities were used in conjunction with the assignment of uniform probabilities based on the bilingual dictionary. The combination protects the system from query words that appear only in the dictionary or only in the parallel corpus.

#### 4.3.3.3 Cross-Language Relevance Models

We described monolingual language models in Section 2.3. Typically, these models differ in how they estimate the likelihood that a given term will appear

in a relevant document. To briefly review, this estimate is trivial if there are relevant documents. However, if there are relevant documents from a prior query, there is no need to run the query. Given this inherent *Catch-22*, Section 2.2.1 contained a variety of different estimates for this probability. Estimates can be made by equating the probability of relevance for a given term with the probability of co-occurrence with *every* term in the query. The idea is that if a term appears in a document that contains every query term, it is highly likely to be relevant. More formally:

$$P(w|R_q) \approx P(w|Q) = \frac{P(w, q_1, q_2, \ldots, q_k)}{P(q_1, q_2, \ldots, q_k)}$$

The next step is to compute the probability of an arbitrary term co-occurring with every term in the query. One estimate is:

$$P(w|q_1, q_2, \ldots, q_k) = \sum_{M \in \mu} P(M) \left( P(w|M) \prod_{i=1}^{k} P(q_i|M) \right)$$

where $P(w|M)$ is the probability of finding term $w$ in a model $M$. The universe of models $\mu$ can be all of the documents in the collection. It follows that $P(M) = \frac{1}{\mu}$ if P(M) uses a uniform distribution. Different estimates based on the *tf* and essentially the *idf* can then be used to estimate the $P(w|M)$.

A cross-language version of this model that uses a comparable corpus simply changes the estimate to compute the probability of the term in language $L'$ that co-occurs in a *comparable* document with *all* of the terms in the query language L. Instead of simply computing the probability of co-occurrence, the comparable corpus is used to map the co-occurring document in language L to one in language $L'$.

More formally:

$$P(w, q_1, \ldots, q_k) = \sum_{M_L, M_{L'} \in \mu} P(M_L, M_{L'}) \left( P(w|M_{L'}) \prod_{i=1}^{k} P(q_i|M_L) \right)$$

To normalize $P(w, q_1, \ldots, q_k)$ it is divided by the summation of $P(w, q_1, \ldots, q_k)$ for every term. Now we use (and this is the big leap that is made when using relevance models) $P(w|Q)$ to estimate $P(w|R_q)$.

For document relevance estimation, the KL divergence relative entropy metrics are used. These metrics compare the $P(w|D)$ distribution from a document to the $P(w|R)$ distribution that was estimated.
Formally:

$$KL(R|D) = \sum_{w} P(w|R) \log \frac{P(w|R)}{P(w|D)}$$

where $P(w|D)$ is estimated by:

$$p(w|D) = \lambda \left( \frac{tf_{w,D}}{\sum_v tf_{v,D}} \right) + (1 - \lambda)P(w)$$

where $v$ represents a term in the entire vocabulary of the document collection. $P(w|D)$ has two components. The first denotes the occurrence of the term $w$ in document D. The second includes the occurrence of term $w$ in the document collection. $\lambda$ may be used to tune the weight of each component.

This model is clearly not scalable because it requires computing probabilities for every term in a language at query time. To speed up computation, only the top documents are retrieved for a query using a more straightforward similarity measure and those documents are then re-ranked with this model. Mean average precision is reported in [Lavrenko et al., 2002].

#### 4.3.3.4 Generalized Vector Space Model

The traditional vector-space model represents documents and queries as vectors in a multi-dimensional term space. Once this is done, a similarity measure can be computed by measuring the distance from a query to a document vector. Another approach is to use a term-document matrix where each row contains the number of occurrences of the term in each document [Wong et al., 1985]. Each column represents a traditional document vector. If we compute $Q' = (A^T)(Q)$, we obtain a new vector representation of the query $Q'$ that has a component for each document in the collection. Each component of $Q'$ is simply an inner product of the query terms with the document that corresponds to this component. A component that represents a document that contains none of the query terms will have a value of zero. A component that represents a document that contains all of the query terms will have a much higher value. For monolingual retrieval, we can simply compute

$$SC(Q, D_i) = \cos\left(Q', A^T D_i\right)$$

where $D_i$ is a document vector.

For cross-language retrieval, using a comparable corpus it is possible to construct the $A^T$ matrix for language L and another matrix $B^T$ for language $L'$ [Carbonell et al., 1997]. To rank document $D_i$ in language $L'$, it is necessary to treat it as we did the query in language L and compute $B^T D_i$ . At this point we have a vector with a component for each document in language L and a similar vector for each document in language $L'$. To compute a cross-language similarity measure, the cosine of the angle between the two vectors is computed. The cross-language similarity measure is given below:

$$SC(Q, D_i) = \cos\left(A^T Q, B^T D_i\right)$$

#### 4.3.3.5 Latent Semantic Indexing

We discussed monolingual Latent Semantic Indexing in Section 2.6. To review, LSI starts with a term document matrix A and uses a singular value decomposition to compute [Dumais, 1994]:

$$A = U\sigma V^T$$

One similarity measure is defined as:

$$SC(Q, D_i) = \cos\left(U^T Q, U^T D_i\right)$$

For cross-language retrieval, the singular value decomposition is applied to the a new term-document matrix that combines the term-document matrix A in language L with the term-document matrix B in language $L'$ [Dumais et al., 1997].

$$\begin{bmatrix} A \\ B \end{bmatrix} = U_2 \sigma_2 V_2^T$$

Once this is done the similarity measure can be computed as:

$$SC(Q, D_i) = \cos\left(U_2^T Q, U_2^T D_i\right)$$

We note that different head to head comparisons of LSI and GVSM for cross-language retrieval were reported in [Carbonell et al., 1997, Littman and Jiang, 1998]. Furthermore, VSM, LSI, and GVSM were shown to be variations of a common algorithm that projects documents and queries into a vector space using singular vectors. The algorithms then differ on how they weight the components of the vectors.

A monolingual comparison of VSM, LSI, and GVSM on the small Cranfield collection and the substantially larger TREC collection is given in [Littman and Jiang, 1998]. For monolingual retrieval, LSI had the best performance on the Cranfield collection, but VSM had the highest effectiveness on the TREC collection.

For cross-language retrieval using TREC French and German documents, LSI outperformed GVSM. An improvement over LSI is obtained by using a new algorithm called ADE (Approximate Dimension Equalization) which is roughly a weighted combination of LSI and GVSM.

**GVSM Example:**

We now give a brief GVSM example using the same English and Spanish documents used in our example in Section 4.3.1. First we compute a vector for the original query. This is computed by multiplying the query vector by the document matrix.

$$Q\prime = A^T Q = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 2 & 0 \\ 0 & 1 & 1 \end{bmatrix}^T \times \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 3 & 2 \end{bmatrix} \qquad (4.3)$$

Next, we obtain vectors for the Spanish Collection. The vectors will represent $Q'$ but will be computed using the Spanish document matrix and the Spanish document collection vectors. We then compute one vector for each document in the collection.

$$Q' = B^T D_{s_1} = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 2 & 0 \\ 1 & 0 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 2 & 0 \\ 1 & 1 & 1 \end{bmatrix}^T \times \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 1 \\ 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 8 & 2 & 6 \end{bmatrix} \qquad (4.4)$$

$$Q' = B^T D_{s_2} = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 2 & 0 \\ 1 & 0 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 2 & 0 \\ 1 & 1 & 1 \end{bmatrix}^T \times \begin{bmatrix} 1 \\ 0 \\ 2 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 2 \\ 1 \\ 0 \\ 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 2 & 17 & 4 \end{bmatrix} \quad (4.5)$$

$$Q' = B^T D_{s_3} = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 2 & 0 \\ 1 & 0 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 2 & 0 \\ 1 & 1 & 1 \end{bmatrix}^T \times \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 6 & 4 & 8 \end{bmatrix} \quad (4.6)$$

Finally, we compute the similarity between the English query vector and each Spanish document vector.

$SC(Gold, Silver, Truck, D_{s_1}) = cos(< 1,3,2 >, < 8,2,6 >) = 0.6814$
$SC(Gold, Silver, Truck, D_{s_2}) = cos(< 1,3,2 >, < 2,17,14 >) = 0.9274$
$SC(Gold, Silver, Truck, D_{s_3}) = cos(< 1,3,2 >, < 6,4,8 >) = 0.8437$

## 4.4 Cross Language Utilities

As we described monolingual utilities in Chapter 3, we now describe cross-language utilities. The idea is that a utility should be impervious to a given strategy.

### 4.4.1 Cross Language Relevance Feedback

Given that we are taking a query in language L, translating it to language $L'$ and then submitting the query, there are two different opportunities to apply relevance feedback (see Section 3.1). Expansion is possible both before the query is translated and after the query is translated. Ballesteros and Croft were the first to describe this in [Ballesteros and Croft, 1997]; they showed improvements in English-Spanish accuracy from both pre and post translation query expansion.

The process was repeated for English-Arabic in [Larkey et al., 2002b]. For pre-translation, using English queries, the top ten documents for each query were obtained. Terms from these documents were ranked based on their weight across the ten documents. The top five terms were added to the query but were only weighted half as important as the original query terms. For post-translation Arabic terms, the top ten documents were retrieved, and the top fifty terms were added to the query.

Another form of query expansion was described in [Adriani, 2000]. Here, it was assumed that the real problem was with the query translation so only post-translation query expansion was used. The top 20 passages from the initial result set were obtained, and a term-term co-occurrence matrix was built for these results. Next, the sum of the similarity of these terms and each post-translation query term was computed. The top ten terms from these passages were then added to the query. A direct application of Rocchio relevance feedback [Rocchio, 1971] (see Section 3.1.1) for post-translation processing was described in [Ruiz, 2003].

Work on query expansion is given in [McNamee and Mayfield, 2002a]. In this work, the authors cite different studies that showed very different results (e.g.; expansion works, expansion does not work). The authors implemented new studies on a sufficiently large dataset. They found that a combination of pre and post translation query expansion provided a fifteen percent effectiveness improvement. They further showed that pre and post translation expansion improved effectiveness even when translation resources were degraded. In these experiments they were testing the impact of limited translation resources. Their key conclusion is that pre-translation expansion does not result in much improvement if translation resources have substantial lexical coverage. This expansion provides significant improvement if lexical coverage is poor.

Relevance feedback in conjunction with logistic regression was recently used for a multilingual retrieval over Chinese, Japanese, and Korean. It was shown to result in as much as a 66 percent improvement in effectiveness [Chen and Gey, 2003].

Interestingly, a more recent result has shown that pre-translation query expansion does improve effectiveness when querying a French document collec-

tion with an English query, but it actually degrades performance when querying a Mandarin Chinese collection [Levow et al., 2004].

### 4.4.2 Stemming

We discussed stemming for English in Section 3.8. For cross-language retrieval, stemming can have a significant impact in a variety of architectural components. First, the source query processor must identify terms. Next, the precise bilingual term list used for translation might dictate the type of stemming needed. A bilingual term list that was built with the Porter stemmer [Porter, 1980] might not be very effective when used with query terms obtained with the Lovins stemmer [Lovins, 1968].

Stemmers for a variety of languages are available and many of them are freely available on the Internet. Broadly speaking, European language stemmers tend to follow the general patterns of the Porter and Lovins stemmers, removing common prefixes and suffixes. We note that with some languages like German, extremely long terms can be difficult to stem as they correspond to several terms in English. A method of *decompounding* long German terms is given in [Kamps et al., 2003].

#### 4.4.2.1 Backoff Translation

When using blingual term lists or lexicons for document or query translation with stemmers, it can be useful to gradually try to match the query term with the term the translation lexicon. A term "jumping" might not match anything in the lexicon, but the root form "jump" might match. When matching an unstemmed or *surface* form of the term with a stem, four combinations exist. In our examples below we use stem($t$) to indicate the stem of term $t$.

- Surface form of query or document term may match with surface form of term in lexicon (e.g.; *jumping* matches with *jumping*).
- Stem form of query or document term may match with surface form of term in lexicon (e.g.; stem(*jumping*) matches with *jump*).
- Surface form of query or document term may match with stem form of term in lexicon (e.g.; *jump* matches with stem(*jumping*)).
- Stem form of query or document term may match with stem form of term in lexicon (e.g.; stem(*jumping*) with stem(*jumped*)).

These matches can be done in succession. Using this progression in which we gradually relax the constraints needed to match a query or document term with a term in a translation lexicon is referred to as *backoff translation*. This technique has been shown to improve effectiveness and was used on the CLEF data in [Oard et al., 2000].

The concern is that there are hundreds of languages and some of them are not popular enough to justify allocating resources to build a language dependent stemmer. Additionally work was done on automatic generation of stemmers (often referred to as *statistical stemming*).

#### 4.4.2.2 Automatically Building a Stemmer

Initial work in Spanish showed that a stemmer could be produced by identifying lexicographically similar words to find common suffixes [Buckley et al., 1994]. Automatically building a stemmer is related to work done in the *Linguistica* software package which is a program that automatically learns the structure of words in any human language solely by examining raw text. The idea is that the software can take an English text and identify that there is a category of terms with suffixes *ing, ed, s*. With only 500,000 terms candidate suffixes can be reasonably identified. A full description of Linguistica may be found in [Goldsmith, 2001]. First basic candidate suffixes are found using weighted mutual information. Regular signatures for each class of words are identified and then a minimum description length (MDL) is used to correct errors. The MDL ensures that breakpoints should find a common stem with a common suffix. The only problem with this for CLIR is that some unusual terms may not be found. Hence, additional work was done to augment the Linguistica set of stems with rule induction. This work grouped stems by frequency for all terms with more than three characters. All two, three, and four character stems were sorted by frequency (after removing stems that were overlapped by a longer stem). Essentially, the most frequent suffixes are used to stem input terms. More details are found in [Oard et al., 2000].

The core idea is that a stemmer is simply a tool that identifies a good *split point* for a term in that the start of the term is the stem, and the rest of the term is the portion that does not need to be stemmed. Results that combined statistical stemming and backoff translation were significantly more effective than an unstemmed collection [Oard et al., 2000]. Interestingly, the stemming based on rule-induction was found to slightly outperform stemming based on Linguistica.

Recently, hidden Markov models were designed to build stemmers [Nunzio et al., 2003]. This approach did not remove prefixes, but started with initial states that ultimately transition to suffix states of size one, two, three, or four characters (at present, this approach is focused on European languages). By simply submitting a list of terms from a new language to the Markov model, it is possible to identify which terms are stems of one another because of their frequency. Once, the model is trained, a term can be processed and the model will output the most likely stem for a given term.

The motivation for this approach is that a training language can have many more instances of *jump* than *jumping*, and the model will learn that there is

a high probability of generating the term *jump* when *jumping* is input. A set of experiments comparing the Markov model-based automatically generated stemmer to the Porter stemmer in Dutch, French, German, Italian, and Spanish found that the automatically derived Markov model stemmer achieved only slightly lower (less than five percent) effectiveness than the manually derived Porter stemmer.

#### 4.4.2.3 Arabic Stemmers

Middle Eastern languages have also been studied at length. The first work on Arabic information retrieval is given in [Al-Kharashi and Evens, 1994]. This work used a hand-crafted stemmer in which a lexicon was used to map each vocabulary term to its corresponding stem or root. Khoja removed common prefixes and suffixes and tried to identify the *root* of a given term [Khoja and Garside, 1999]. Enhancements to this stemmer are given in [Larkey and Connell, 2000]. The problem with this approach is that the root of a term is often very broad. For example, the Arabic word for *office* and *book* both have the same root. Subsequently, work that simply removed plurals and other less harmful stems demonstrated substantial improvement. These Arabic stemmers are referred to as *light* stemmers. Light stemmers for Arabic are described in [Larkey et al., 2002b, Darwish and Oard, 2003, Aljlayl et al., 2002, Aljlayl and Frieder, 2002]. A head-to-head comparison of an early light stemmer named *Al-Stem*, the light stemmer described in [Larkey et al., 2002b] and a modified *Al-Stem* is given in [Darwish and Oard, 2003]. Additional, slight improvements are given in [Chen and Gey, 2002]. We note that these stemmers would be useful for monolingual Arabic retrieval, but they are clearly useful for CLIR as well as they support queries from a query in language L to then be translated into Arabic, the newly translated query may then be run against the Arabic collection, and Arabic documents will be retrieved.

#### 4.4.2.4 Asian Languages

Chinese and other Asian languages pose the interesting problem that white space is not used to denote a word boundary. A statistical approach using staged logistic regression to identify Chinese word boundaries from a trained set of 300 sentences is given in [Dai et al., 1999]. In this work, the relative frequency of individual characters and two character pairs called *bigrams* was used with the document frequency, the weighted document frequency, the local frequency, contextual information (frequency of surrounding characters), and positional information. A combined approach is given in [Xue and Converse, 2002] – this combines frequency based algorithms and dictionary-based algorithms which simply look for the longest matching set of characters in a machine-readable dictionary. Interestingly, it has been shown that the segmentation problem can be ignored for Chinese and Japanese CLIR and reason-

able results can be obtained simply by using overlapping n-grams (see Section 4.4.3).

### 4.4.3 N-grams

A language-independent approach uses sequences of characters called *n-grams* [Damashek, 1995]. We described n-grams for use in English in Section 3.4. For European languages, n-grams of size $n = 4$ or $n = 5$ showed good effectiveness. Interestingly, n-grams were compared to stemming. An approach that used the least common n-gram to represent a term, resulted in a slight improvement over a Porter stemmer [Mayfield and McNamee, 2003]. The idea is that a single carefully chosen n-gram for each word might serve as an adequate stem substitute.

The problem of segmentation for Asian languages was avoided when n-grams were used by in [McNamee, 2001]. Interestingly, results were comparable to more sophisticated segmentation algorithms. Additional cross-language experiments with 6-grams are described in [Qu et al., 2003b].

### 4.4.4 Transliteration

Recent work focused on using forms of transliteration to assist with the cross-language name matching problem. Pirkola et al use transformation rules to identify how a name in language L should be transliterated into a name in language $L'$ [Pirkola et al., 2003]. The core motivation is that these technical terms and names are not found in machine readable bilingual term lists typically used to translate query or document terms. In this effort, transformation rules are learned by using paired lists of terms in a source and target language. First, terms are identified that are within some reasonable threshold of closeness as defined by their edit distance. The Levenshtein edit distance is computed as the minimum number of substitutions, insertions, and deletions needed to modify the source term into the target term [Levenshtein, 1966]. For example the edit distance between *fuzzy* and *fuzy* is one.

Once the candidate sets of terms are found, all transformations that resulted in the minimum edit distance were considered. An error value was assigned for each character by character transformation. The assignments included error values of zero for no change, and one for less substantial changes such as consonant replaces consonant, vowel replaces vowel, or insert or delete of one character. A value of two is assigned for more substantial changes (e.g.; a consonant replaces vowel or vowel replaces consonant). The transformation with the least error values are then identified. Once this is done, a threshold is used to ensure that a given transformation occurs frequently. These transformations are then used for source to target translation. The confidence of a given rule can be varied such that more rules with higher recall can be gener-

ated or fewer rules with higher precision. Test results showed that for several languages, there was significant improvement when using these transformation in conjunction with n-grams [Pirkola et al., 2003].

#### 4.4.4.1 Phonetic Approach

For English-Japanese retrieval, transliteration using a phonetic approach was studied in [Qu et al., 2003a]. For English-Japanese the general approach is to use the phonetic sounds of the term. Transliteration is done:

English word → English phonetic → Japanese phonetic → Japanese word

English phonetic representations are found for a given English term. The first phonetic representation for a term in the CMU Speech Pronunciation Dictionary is used. The dictionary contains 39 different phonemes. Once the Japanese phonemes are identified a standard mapping converts from Japanese phoneme to the target *katakana* character set.

For Japanese-English the problem is harder because of the lack of word boundaries in Japanese. First, a word segmentation algorithm which essentially matches the longest match word in a dictionary is used. The EDICT Japanese-English dictionary is used. Using the same phonetic process as used for English-Japanese, Japanese-English phonetic transliterations are obtained. These transliterations can be thought of as simply additions to the EDICT dictionary. Checking the transliteration is required because it is possible to have dramatic word segmentation failures. An example of the term *Venice* which has no entry in EDICT is transliterated into *bunny* and *cheer* because of the subparts of the Japanese word that did match EDICT. Hence, transliterations are checked to see if they co-occur with a reasonable mutual information in the target corpus as they occur in the source corpus. Dramatic improvements in query effectiveness were found for queries that contained terms that did not occur in EDICT.

### 4.4.5 Entity Tagging

Identifying entities in English was described in Section 3.8. An obvious extension to this is to identify people, places, and locations in documents written in other languages. If we are able to identify these entities it may well open the door to improved transliteration and phonetic techniques specific to these entities for CLIR (see Section 4.4.4 and 4.4.4.1). Also, if we know an item is a person name, perhaps we could suppress incorrect translations by avoiding a lookup in a common translation dictionary.

Machine learning techniques were shown to work well for this problem as a training set of named entities can be identified and learning algorithms can be used to identify them [Carreras et al., 2002]. Earlier work used a lan-

guage independent bootstrapping algorithm based on iterative learning and re-estimation of contextual and morphological patterns [Cucerzan and Yarowsky, 1999]. This algorithm learns from un-annotated text and achieves good performance when trained on a very short labelled name list with no other required language specific information.

For each entity class to be recognized and tagged, it is assumed that the user can provide a short list (around one hundred) unambiguous examples (seeds). Additionally, some specifics of the language should be known (e.g.; presence of capitalization and the use of word separators). This algorithm relies on both internal and contextual clues as independent evidence. Internal clues refer to the morphological structure of the word. These clues use the presence of prefixes and suffixes as indicators for certain entities. For example, knowing that *Maria, Marinela, and Maricica* are feminine first names in Romanian, the same classification can be a good guess for *Mariana* because of the common prefix: *Mari*. Suffixes are typically even more informative, for example: *-escu* is an almost perfect indicator of a last name in Romanian – the same applies to *-wski* in Polish *-ovic* and *-ivic* in Serbo-Croatian and *-son* in English. Such morphological information is learned during bootstrapping. The algorithm was used to mark named entities in named entities in English, Greek, Hindi, Rumanian and Turkish.

A language dependent Arabic entity tagger that relies on specific patterns to do morphological analysis is described in [Maloney and Niv, 1998]. An Arabic entity tagger without language dependent resources is described in [McNamee and Mayfield, 2002b]. These use algorithms similar to those used in English entity taggers.

It was shown that bilingual term lists frequently lack coverage of entities. For some cross-language tasks (typically using English and European languages), it is possible to recognize an entity and hope that it simply will be spelled the same in both languages. Recent experiments showed that when proper names were removed from a bilingual lexicon, performance was reduced by over fifty percent.

The Conference on Computational Natural Language Learning (CoNLL-2003) focused on language independent named entity recognition. Results for both English and German were obtained and the effectiveness of the top ranked systems were well above eighty-five percent for both precision and recall. In German, the accuracy was between eighty and eighty-five percent [Sang and Meulder, 2003]. The top three results in English and the top two in German used Maximum Entropy Models [Guiasu and Shenitzer, 1985].

Overall, entity tagging in foreign languages and exploiting these entities for cross-language retrieval is a topic of future research.

### 4.4.6 Fusion

Fusion of different retrieval strategies for monolingual retrieval is discussed at length in Section 8.3. Simply merging results from different strategies (e.g., vector space model, language models, LSI, etc.) has not been shown to improve accuracy, but merging results from different representations (e.g.; title only queries, title+description, etc.) was shown to help. Fusing different query representations for Arabic cross-language retrieval is described in [Larkey et al., 2002a]. The reality is that a user can issue a query to multiple document collections, and each collection can be written in a different language. This can be viewed as multilingual information retrieval (MLIR). For multilingual retrieval, two architectures exists: centralized and distributed. A distributed architecture indexes documents in each language separately. A centralized collection combines all of the documents in a single index.

To rank documents for an MLIR, three approaches were tested recently:

- Translate the query into each language and build a separate index with each language. Normalize all relevance scores and present a single ranked list of documents. This is a distributed approach.
- Translate the query into each language and build a separate index with each language, but instead of merging by score, merge by the relative position in the result list. This is a distributed approach.
- Translate the query into each language and concatenate all terms into one big query. Index all of the documents in each language into one large index. Run the single query against the single index. This is a centralized approach.

The centralized approach performed better than the previous two. It can be that the presence of proper nouns and the ability to track their distribution over the entire document collection is the reason for this success [Martinez et al., 2003]. Fusion can also be done by weighting a document collection based on its predicted retrieval effectiveness. With this approach, a source collection might receive a low weight if a poor-quality translation dictionary exists.

Additionally, the number of ambiguous words in a query for a given collection can be used to weight a collection. It was shown that this approach results in a slight improvement in accuracy [Lin and Chen, 2003].

## 4.5 Summary

In some cases, cross-language retrieval systems actually exceed monolingual retrieval. Hence, some have deemed cross-language retrieval a *solved* problem. However, much work remains to be done in terms of user interaction of these systems and more study as to why they are not in widespread use.

Directions for the future of cross-language retrieval are described in [Oard, 2003].

Numerous commercial search engines focus on monolingual search. However, CLIR systems have not been widely adopted. A key question is: How many users truly wish to query documents using a language they know and return results in a language that they do not know? More studies need to be done on how many projects exist with this requirement and such work should gather precise reasons as to why these systems have not deployed existing CLIR algorithms.

## 4.6 Exercises

1 Consider a system in which documents are obtained in multiple languages. Describe what factors would you use to determine if the documents should be translated prior to indexing or if you should simply translate incoming queries.

2 Give a detailed example of how a CLIR result might exceed 100 percent of its monolingual counterpart.

3 Consider two documents collections in languages A and B. Documents in language A all have a corresponding document in Language B. Describe CLIR algorithms that could be used to take advantage of this comparable corpus.

# Chapter 5

# EFFICIENCY

Thus far, we have discussed algorithms used to improve the effectiveness of query processing in terms of precision and recall. Retrieval strategies and utilities all focus on finding the relevant documents for a query. They are not concerned with how long it takes to find them.

However, users of production systems clearly are concerned with run-time performance. A system that takes too long to find relevant documents is not as useful as one that finds relevant documents quickly. The bulk of information retrieval research has focused on improvements to precision and recall since the hope has been that machines would continue to speed up. Also, there is valid concern that there is little merit in speeding up a heuristic if it is not retrieving relevant documents.

Sequential information retrieval algorithms are difficult to analyze in detail as their performance is often based on the selectivity of an information retrieval query. Most algorithms are on the order of $O(q(tf_{max}))$ where $q$ is the number of terms in the query and $tf_{max}$ is the maximum selectivity of any of the query terms. This is, in fact, a high estimate for query response time as many terms appear infrequently (about half are *hapax legomena*, or those that occur once according to Zipf's law [Zipf, 1949]).

We are not aware of a standard analytical model that effectively can be used to estimate query performance. Given this, sequential information retrieval algorithms are all measured empirically with experiments that require large volumes of data and are somewhat time consuming.

The good news is that given larger and larger document collections, more work is appearing on improvements to run-time performance. We describe that work in this chapter. Additional research was done to speed up information retrieval algorithms by employing multiple processors. That work is covered in Chapter 7. However, in any study of parallel algorithms, it is important

that work be compared with the best sequential algorithm. Hence, this chapter describes the best sequential algorithms that we are aware of for information retrieval.

We also note that most algorithms described in this chapter are directed at the vector space model. These algorithms are also directly applicable to the probabilistic model. Clearly, similar work is needed to improve performance for other retrieval strategies and utilities.

Early information retrieval systems simply scanned very small document collections. Subsequently, inverted indexes were used to speed query processing at the expense of storage and time to build the index. Signature files were also proposed. These are typically smaller and faster, but support less retrieval functionality than an inverted index.

Compression of inverted indexes is often used to speed up query processing. Additionally, partial document rankings that are much faster than full rankings can be done at relatively low cost. In some cases precision and recall are comparable to doing a full ranking [Lee and Ren, 1996].

In Section 5.1, we first survey inverted indexing and then describe methods used to compress an inverted index. In Section 5.2, we describe algorithms that improve run-time of query processing, and in Section 5.3, we review signature files.

## 5.1 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. Inverted indexes were used in both early information retrieval and database management systems in the 1960's [Bleir, 1967]. 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*. Figure 5.1 illustrates an inverted index.

An entry in the list of documents can also contain the location of the term in the document (e.g., word, sentence, paragraph) to facilitate proximity searching. Additionally, an entry can contain 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 coefficient and presenting a ranked list to the user.

Indexing requires additional overhead since the entire collection is scanned and substantial I/O is required to generate an efficiently represented inverted index for use in secondary storage. Indexing was shown to dramatically reduce the amount of I/O required to satisfy an ad hoc query [Stone, 1987]. Upon

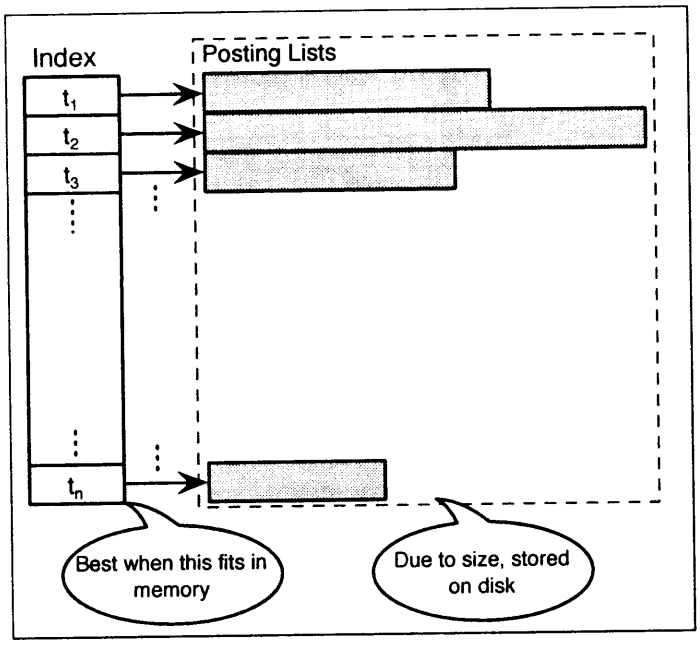



*Figure 5.1.* Inverted Index

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

The size of the index is another concern. Many indexes can be equal to the size of the original text. This means that storage requirements are doubled due to the index. However, compression of the index typically results in a space requirement of less than ten percent of the original text [Witten et al., 1999]. The terms or phrases stored in the index depend on the parsing algorithms that are employed (see Section 3.8).

The size of posting lists in the inverted index can be approximated by the Zipfian distribution—Zipf proposed that the term frequency distribution in a natural language is such that if all terms were ordered and assigned a rank, the product of their frequency and their rank would be constant [Zipf, 1949]. Table 5.1 illustrates the Zipfian distribution when this constant is equal to one.

Using $\frac{C}{r}$, where $r$ is the rank and C is the value of the constant, an estimate can be made for the number of occurrences of a given term. The constant, C, is domain-specific and equals the number of occurrences of the most frequent term.

*Table 5.1.* Top Five Terms in Zipfian Distribution

| Rank | Frequency | Constant |
|---|---|---|
| 1 | 1.00 | 1 |
| 2 | 0.50 | 1 |
| 3 | 0.33 | 1 |
| 4 | 0.25 | 1 |
| 5 | 0.20 | 1 |

### 5.1.1 Building an Inverted Index

An inverted index consists of two components, a list of each distinct term referred to as the *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, tf*> for each distinct term in the collection. A typical uncompressed index spends four bytes on the document identifier and two bytes on the term frequency since a long document can have a term that appears more than 255 times.

Consider a document collection in which document one contains two occurrences of *sales* and one occurrence of *vehicle*. Document two contains one occurrence of *vehicle*. The index would contain the entries *vehicle* and *sales*. The posting list is simply a linked list that is associated with each of these terms. For this example, we would have:

$$sales \rightarrow (1, 2)$$
$$vehicle \rightarrow (1, 1)\,(2, 1)$$

The entries in the posting lists are stored in ascending order by document number. Clearly, the construction of this inverted index is expensive, but once built, queries can be efficiently implemented. The algorithms underlying the implementation of the query processing and the construction of the inverted index are now described.

A possible approach to index creation is as follows: 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**. Contains 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:

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

where $d_i$ indicates the document identifier of document $i$ and $tf_{ij}$ indicates the number of times term $j$ occurs in document $i$.

- **Document file**. Contains information about each distinct document—document identifier, long document name, date published, etc.
- **Weight file**. Contains the weight for each document. This is the denominator for the cosine coefficient—defined as the cosine of the angle between the query and document vector (see Section 2.1).

The construction of the inverted index is implemented by scanning the entire collection, one term at a time. When a term is encountered, a check is made to see if this term is a stop word (if stop word removal is used) or if it is a previously identified term. A hash function is used to quickly locate the term in an array. Collisions caused by the hash function are resolved via a linear linked list. Different hashing functions and their relative performance are given in [McKenzie et al., 1990]. 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. If it does, the term frequency is merely incremented. Otherwise, this is the first occurrence of this term in the document, so a new posting list entry is added to the start of the list.

The posting list is stored entirely in memory. Memory is allocated dynamically for each new posting list entry. With each memory allocation, a check is made to determine if the memory reserved for indexing has been exceeded. If it has, processing halts while all posting lists resident in memory are written to disk. Once processing continues, new posting lists are written. With each output to disk, posting list entries for the same term are chained together.

Processing is completed when all of the terms are processed. At this point, the inverse document frequency for each term is computed by scanning the entire list of unique terms. Once the inverse document frequency is computed, it is possible to compute the document weight (the denominator for the cosine coefficient). This is done by scanning the entire posting list for each term.

### 5.1.2 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. The size of the index file determines the storage overhead imposed. Furthermore, since large index files demand greater I/O bandwidth to read them, the size also directly affects the processing times.

Although compression of text was extensively studied [Bell et al., 1990, Gutmann and Bell, 1994, Gupte and Frieder, 1995, Trotman, 2003], relatively

little work was done in the area of inverted index compression. However, in work by Moffat and Zobel, an index was generated that was relatively easy to decompress. It comprised less than ten percent of the original document collection, and, more impressively, *included stop terms.*

Two primary areas in which an inverted index might be compressed are the term dictionary and the posting lists. Given relatively inexpensive memory costs, we do not focus on compression of indexes, although some work is described in [Witten et al., 1999]. The King James Bible (about five megabytes) contains 9,020 distinct terms and the traditional TREC collection (slightly over two gigabytes) contains 538,244 distinct terms [Witten et al., 1999]. The number of new terms always slightly increases as new domains are encountered, but it is reasonable to expect that it will stabilize at around one or two million terms. With an average term length of six, a four byte document frequency counter, and a four byte pointer to the first entry in the posting list, fourteen bytes are required for each term. For the conservative estimate of two million terms, the uncompressed index is likely to fit comfortably within 32 MB. Even if we are off by an order of magnitude, the amount of memory needed to store the index is conservatively under a gigabyte.

Given the relatively small size of an index and the ease with which it should fit in memory, we do not describe a detailed discussion of techniques used to compress the index. We note that stemming reduces this requirement and Huffman encoding can be used in a relatively straightforward fashion [Witten et al., 1999]. Also, the use of phrases improves precision and recall (see Section 3.8.2). Storage of phrases in the index may well require compression. This depends upon how phrases are identified and restricted. Most systems eliminate phrases that occur infrequently.

To introduce index compression algorithms, we first describe a relatively straightforward one that is referred to as the Byte Aligned (BA) index compression [Grossman, 1995]. BA compression is done within byte boundaries to improve runtime at a slight cost to the compression ratio. This algorithm is easy to implement and provides good compression (about fifteen percent of the size of an uncompressed inverted index when stop words are used). Variable length encoding is described in [Witten et al., 1999]. Although such efforts yield better compression, they do so at the expense of increased implementation complexity.

#### 5.1.2.1 Fixed Length Index Compression

As discussed in the previous section, the entries in a posting list are in ascending order by document identifier. An exception to this document ordering occurs when a pruned inverted index approach is used (see Section 5.1.5). Hence, run-length encoding is applicable for document identifiers. For any document identifier, only the offset between the current identifier and the iden-