3 Query Expansion and Refinement

A query or information need submitted by an end-user is ordinarily a short statement or an even shorter list of terms. This is only to be expected. The normal user is not necessarily an expert on all the term usages in a large collection of documents he wishes to query. Nor does he want to spend his time consulting thesauri and other reference works in an attempt to generate an ideal query. A sophisticated user may in fact do some of these things. But the approach taken in both some commercial IF engines and much IF research is to refine and expand the original query automatically based on the documents retrieved by the original query. [Salton et al., JASIS, 1990]

Query refinement and expansion may involve adding additional terms, removing “poor” terms, and refining the weights assigned to the query terms. It is possible to recompute the weights without expanding the query, or to expand the query without recomputing the weights, but experiment indicates that both expansion and re-weighting improve filtering performance. [Harman, SIGIR ‘92] The process of query expansion and re-weighting can be applied to either vector space queries or extended boolean queries. [Salton, ATP, 1989] [Salton, IP&M, 1988] The process may be wholly automatic or may involve a combination of automatic processes and user interaction.

3.1 Query Expansion (Addition of Terms)

A number of approaches to automatic query expansion have been tried. A common approach is to expand the query with terms drawn from the most relevant documents, i.e., the documents that the user judges relevant among those that were retrieved when her original query was applied to the collection. This process is called “relevance feedback.” [Salton et al., JASIS, 1990], [Harman, SIGIR ‘92] The process is interactive to the extent that the user must look at the documents most highly ranked by the filtering system and tell the system which ones are relevant. Note that since modern IF systems typically rank all the documents in the collection, the user must decide how far down the ranking she wants to go, e.g., she must decide to read the highest ranking $X$ documents where $X$ is a parameter of the filtering procedure. In effect, she makes the working assumption that the first $X$ documents are likely to be relevant and hence are worth examining initially to judge their relevance. (But see below for further refinements.) However, the system can “take it from there,” extracting terms from the relevant documents and adding them to the original query. The system can also reweight
terms in the original query, e.g., increasing the weights of terms that appear in the documents judged relevant, and reducing the weights of terms that do not appear in the relevant documents. If there is a training set of documents such that human judges have already identified relevant and non-relevant documents for this set (relative to the given query or topic), then terms occurring in the relevant documents of this training set can be added to the initial query. (Training sets are normal for routing or classification applications.) The process of relevance feedback is iterative. Each time the query is expanded and re-weighted, this “improved” query, also called the “feedback query” [Buckley et al., SIGIR ‘94], is executed. The user then makes relevance judgments of the top $N$ documents retrieved by the feedback query. “The process can continue to iterate until the user’s information need is satisfied.” Harman [SIGIR ‘92] recommends repeating the process as long as each iteration retrieves relevant documents not retrieved in the previous iteration. (Note: In the “residual collection” method of evaluation of relevance feedback [Salton and Buckley, JASIS, 1990] [Haines and Croft, SIGIR ‘93] [Chang et al., SMART, 1971], all documents previously seen and judged by the user are factored out of evaluation of the next iteration so that each iteration is only “credited” with additional relevant documents not previously retrieved.)

A refinement of the above procedure is to let the IF system assume that the top $X$ documents as ranked by the IF system are relevant. The system automatically expands the query using these $X$ documents, runs the expanded query and returns the ranking produced by this second-stage query as the first result actually seen by the user. Note that for the purposes of automatic query expansion (the first stage of this two-stage process), precision is more important than recall, i.e., it is essential that as many of the high-ranking $X$ documents as possible are relevant even if some relevant documents are overlooked. Hence, similarity measures may be employed in this automatic first stage that emphasize precision and sacrifice some recall. [Cornell, TREC 5]

When terms from relevant documents are to be added to the query, two questions are important: (1) How many documents should the user judge for relevance, e.g., the top 10 (a screenful), the top 30, the top 200? Or should the threshold be determined by similarity or probability value, e.g., all documents above a given similarity value? In any case, once a threshold is set, all documents above that threshold become the “filtering set.” The user judges each document in the filtering set as relevant or non-relevant. (2)
What should be the measure of whether a term that occurs in documents judged relevant is “important enough” to be added to the query? A common answer to the latter question is to add the term vectors of all documents judged relevant to the query. In other words, all terms in the relevant documents (after stop-word removal and stemming) are added to the query. However, if a relevant document is very large, adding all the terms in its term vector to the original query can produce a very large expanded query. This can degrade response time because “efficient large-scale filtering systems have response times that are heavily dependent on the number of query terms rather than the size of the collection.” [Harman, SIGIR ’92]

A refinement to the term selection procedure is to take all the terms from the term vectors of the relevant retrieved documents, sort them according to some criterion of importance, and then add the top $N$ terms from this sorted list to the query. An effective key for sorting in one experiment [Harman, SIGIR ’92] was found to be “noise*frequency” where noise is a global distribution term similar to $idf$, and frequency is the log of the total number of occurrences of the given term within the set of relevant retrieved documents. In other words, preference is given to terms that occur frequently in the documents judged relevant but that do not occur frequently in the collection as a whole. The same experiment found that adding the top 20 terms produced better filtering performance (measured as average precision) than adding a much larger number of terms, e.g., all of the non-stop words in the retrieved relevant documents. Adding all the terms dilutes “the effect of the ‘important’ terms … causing many non-relevant documents to move up in rank.” (Of course, if a given query goes through repeated iterations, 20 terms will be added at each iteration.) This experiment simulated an interactive, ad hoc query environment in which the user issues a query, judges the top $N$ ($N = 10$) documents for relevance, and the system then automatically expands the query using the “best” terms taken from the relevant retrieved documents. The user repeats this relevance feedback process with each iteration using the query expansion produced by the previous iteration.

Buckley et al. [TREC 2] [TREC 3] achieved substantial improvement in a routing environment by massive query expansion, e.g., each query was expanded by 200-300 terms before filtering improvement reached a point of diminishing returns. Subsequently [SIGIR ‘94], they reported improved performance for expansion up to
500 terms. However, Singhal et al. [TREC 4] found that with their improved term weighting scheme ($L_n u$ — see section 3.3.2), the maximum improvement occurs at 80-100 added terms. (Traditional term weighting favors shorter documents. Massive query expansion compensates for this bias by favoring longer documents. The new weighting scheme, by eliminating the document length bias, reduces the need for the compensating bias of massive expansion.)

The above approaches distinguish “important” terms, i.e., terms that are effective at discriminating documents about a given topic, from terms that are poor at such discrimination. However, as noted earlier, a large document may deal with a number of topics. Hence, it is quite possible that inappropriate terms will be added to the query, drawn from non-relevant sections of relevant documents. These terms may be good topic discriminators but they may discriminate the wrong topic. One approach to this problem, discussed earlier (see section on query-document similarity), is to break each large document into sections, commonly called “passages,” and treat each passage as a “document.” [Allan, SIGIR ‘95] In other words, the system computes the similarity between each passage and the user’s query. This enables the system to determine the “best” passage, i.e., the passage in a given document most similar (and hence one hopes most relevant) to the query. The query is then expanded only with terms taken from the “best” passage of each relevant document.

An alternative method of query expansion (mentioned earlier) is to expand each term in the original query with synonyms or related terms drawn from a generic on-line thesaurus, e.g., Princeton’s public domain WordNet [Miller, IJL, 1990] or a thesaurus developed for a particular application domain. [van Rijsbergen, 1979] A thesaurus may be an independently generated reference work, e.g., WordNet, or it may be generated from the target collection based on term co-occurrence or adjacency, e.g., INQUERY’s PhraseFinder. [Callan et al, IP&M, 1995] A term can be expanded with synonyms or replaced by a more general word representing the class to which the original term belongs. [van Rijsbergen, 1979] The use of a thesaurus to expand a set of terms automatically into a Boolean expression was discussed briefly in the section above on Boolean queries. Query expansion by thesaurus may be truly automatic since unlike expansion based on relevance feedback, no feedback from the user is required. On the other hand, a user may expand a query manually using a thesaurus. Of course,
expansion by thesaurus, and expansion with terms drawn from the relevant document set of the original query, are not mutually exclusive.

In a third approach to query expansion, interactive query expansion, the user is supplied with a list of candidate expansion terms derived from the relevant documents and ordered by some criterion of importance such as the “noise*frequency” discussed above. The user then chooses terms from this list to add to the query. [Efthimiadis, IP&M, 1995].

Whether terms for query expansion are selected interactively or automatically, a term ordering (also called “ranking” or “selection”) method is required. Efthimiadis, Harman [SIGIR ‘92], and Haines and Croft [SIGIR ‘93], have all studied term ordering methods. In general, the experiments conducted by these authors found that multiple ordering methods had comparable performance. They also found that the results depended on the collection to which it was applied, e.g., Haines and Croft found that relevance feedback worked much better on a collection of abstracts than on a collection of full-text documents. They found that a good term ordering method was \( rdf*idf \) where \( rdf \) is the number of documents judged relevant by the user that contain the given term. Harman and Efthimiadis both found that a probabilistic ordering method comparable to (in Harman, slightly better than) “noise*frequency” was the BI weighting formula (see section above on probability):

\[
w_k = \log \frac{p_k(1-u_k)}{u_k(1-u_k)}
\]

where \( p_k \) is the probability that the term \( t_k \) appears in a document given that the document is relevant, and \( u_k \) is the probability that \( t_k \) appears in a document given that the document is non-relevant. Harman and Efthimiadis both got good results with \( w_k(p_k - u_k) \). Efthimiadis got comparable results with his “r-lohi” method which consists of ranking terms by \( rtf \) (the number of occurrences of the given term in relevant documents) and breaking ties according to their term frequency (over all documents) from low frequency to high frequency. Buckley et al. [SIGIR ‘94] use \( rdf \), breaking ties by choosing the term with the highest average weight in the set of relevant documents.

Local Context Analysis (LCA) [Xu et al., SIGIR ‘96] employs a more elaborate scheme for automatic choosing, ranking, and weighting of query expansion terms. LCA “combines global analysis and local feedback.” It is based on fixed length passages
(300 words in the reported experiments). It is also based on “concepts” where a concept is a noun group (phrase), defined as “either a single noun, two adjacent nouns, or three adjacent nouns.” Given a query $Q$, a standard IF system \((\text{INQUERY} \text{ in the reported experiments})\) retrieves the top \(n\) passages in the collection being queried, i.e., the \(n\) passages in the entire collection that match \(Q\) most closely. Concepts in the top \(n\) passages are ranked according to the formula:

$$bel(Q,c) = \prod_{i \in Q} (\delta + \log(af(c,t_i)idf_c / \log(n)))$$

where \(c\) is the concept being ranked relative to query \(Q\), and \(bel\) (which stands for “belief”) is the ranking function. The heart of this ranking function is:

$$af(c,t_i) = \sum_{j} ft_{ij} fc_j$$

which multiplies the frequency of occurrence of query term \(t_i\) in passage \(j\) \((ft_{ij})\) by the frequency of occurrence of concept \(c\) in passage \(j\) \((fc_j)\), and then sums this product over all the \(n\) top passages. Hence, this factor measures the co-occurrence of a given query term \(t_i\) with the concept \(c\) being ranked, over all of the top \(n\) passages. The greater the amount of co-occurrence (co-occurrence in more passages, greater frequency of co-occuring query term and concept within a given passage), the greater the ranking of \(c\). This ranking factor is modified (multiplied) by \(idf_c\), a variation on the familiar global \(idf\) statistic which penalizes concepts occurring frequently in the collection. This product is normalized by the log of the number of top ranked passages. A small term, is added to prevent any concept from getting a score of zero. The resulting sum is raised to the power \(idf_c\) to emphasize infrequent query terms. The result is a score for \(c\) relative to query term \(t_i\). Finally, the scores for \(c\) relative to each of the query terms \(t_i\) are multiplied together to obtain the final ranking score for concept \(c\). Note that “[m]ultiplication is used to emphasize co-occurrence with all query terms.” In other words, if even one of the query terms has a very low co-occurrence score with a concept \(c\), that concept will receive a low ranking.

Once the concepts in the top \(n\) passages have been ranked, the \(m\) highest ranking concepts are used to form an auxiliary query, which combined with the original query \(Q\) using a weighted sum operator. In the reported experiments, \(m\) was set to 70. These \(m\) expansion concepts were weighted (within the auxiliary query) according to a linear
weighting function such that the highest ranking concept, \( c_1 \) received a weight close to 1, and the lowest ranking (\( m \)-th) concept received a weight of 0.1. The expanded query was then applied to TREC3 and TREC4 data. All runs produced improvement (measured in average precision), but the amount of improvement depended on the number of passages. The best run on TREC3 (200 passages) produced an improvement of 24.4\%. The best run on TREC4 (100 passages) produced an improvement of 23.5\%. Currently, no method of choosing the optimum number of passages is known, but fortunately, performance is relatively flat for a wide range (30 to 300 passages), at least for the TREC data. If the number of passages is below or above this optimum range, the level of improvement declines.

Note the mixture of global and local analysis in LCA. The \( n \) passages that best match \( Q \) are selected globally from the entire collection. Likewise, the two \( \text{idf} \) statistics, \( \text{idf}_c \) and \( \text{idf}_i \), are calculated globally over all the passages in the collection. On the other hand, query term/concept co-occurrence is computed “locally,” i.e., it is computed only for those \( n \) best passages retrieved for \( Q \). Note that term occurrence statistics and \( \text{idf} \) statistics can be computed ahead of time for all the passages in a given collection. At run time, i.e., when a query \( Q \) is being executed, the only global activity required is filtering of the \( n \) best passages. Co-occurrence statistics and the concept ranking can then be computed locally. Hence, filtering time is very fast.

LSI has been used as a form of query “expansion” in conjunction with (or in place of) relevance feedback. In “[m]ost of the tests … the initial query is replaced with the vector sum of the documents the users have selected as relevant.” [Berry et al., SIAM, 1995] The effect is equivalent to incorporating terms from the relevant documents into the query, yet the resulting query vector using LSI factors is much lower-dimensional than if the terms themselves were actually added (which means that the query will execute much faster). [Harman, SIGIR ‘92] [Hull, SIGIR ‘94] Dramatic improvements were achieved when even the first relevant document or the average of the first three relevant documents were used.

### 3.2 Query Refinement (Term Re-Weighting)

Once the query has been expanded, the weights of query terms must be recalculated. A widely used method of re-calculating the term weights given relevance feedback is the
The Rocchio formula calculates new term weights from the old term weights and the relevance judgments as follows. Let $Q_i^{old}$ be the existing weight of the $i$-th term, e.g., computed using a scheme like the popular “ltc” (see section above on term weights) for a term in the original (non-expanded) query. Of course, if term $t_i$ is an expansion term, then there is no $Q_i^{old}$, i.e., $Q_i^{old} = 0$ in the equation below. Let $Q_i^{new}$ be the new weight of query term $i$ after re-evaluation. Let $|rel\ docs|$ be the number of retrieved documents judged relevant. Let $|nonrel-docs|$ be the number of retrieved documents judged non-relevant. Let $wt_i$ the weight of term $t_i$ in any given relevant or non-relevant document. Let $A$, $B$, and $C$ be three constants to be adjusted experimentally. Then the Rocchio formula [Buckley et al., SIGIR ‘94] [Salton and Buckley, JASIS, 1990] is:

$$Q_i^{new} = A Q_i^{old} + B \frac{1}{|rel\ docs|} \sum_{rel\ docs} wt_i - C \frac{1}{|nonrel\ docs|} \sum_{nonrel\ docs} wt_i$$

Note that the $B$ term in the Rocchio formula averages the weights of query term $t_i$ over the relevant documents. (Remember that a given term can have a different weight in each relevant document in which it occurs.) The $C$ term averages the weights of query term $t_i$ over the non-relevant documents. Hence, the ratios of $A$, $B$, and $C$ determine the relative importance of the old query (i.e., the previous version of the query), the relevant documents, and the non-relevant documents in modifying term weights in the query. It should be stressed that the query expansion terms are drawn entirely from the documents judged relevant. However, some of these expansion terms may also occur in non-relevant documents. Hence, the $C$ portion of the Rocchio formula does not add any terms to the query; its only effect is to reduce the weights of some expansion query terms (or on a first iteration, original query terms) because of their occurrence in non-relevant documents. The effect of the $C$ portion of Rocchio may be to make the weight of some terms negative. Terms with negative weights may be dropped. [Buckley et al., SIGIR ‘94] This is the “original” Rocchio formula. A modified version of the formula [Buckley, SIGIR ‘94] re-interprets “nonrel docs” to include not just the retrieved documents that the user has explicitly judged non-relevant but all documents in the collection that have not been explicitly judged relevant by the user. The assumption of the “modified” formula
is that most of the documents the user never sees, the non-retrieved or lower-ranking documents, will in fact be non-relevant.

Buckley et al. point out a significant virtue of the modified Rocchio formula. By pure chance, a low frequency term may easily happen to occur more frequently in relevant than in non-relevant documents within a given collection. Rocchio averages such a term over all the relevant documents, and all the non-relevant documents, in the collection, not merely the relevant and non-relevant documents in which the term happens to occur. Hence, both relevant and non-relevant documents will make a small contribution to the weight of a low-frequency term. Rocchio weights an expansion term by the difference between these contributions, which will therefore also be small. This is contrasted with the probabilistic BI term weighting formula discussed in section 7.3.1. This formula computes the term weight as the log of a ratio involving the probabilities of occurrence of the given term in relevant and non-relevant documents. For example, if 1/10 of the relevant documents contain the given term $t_k (p_k = 0.1)$, and 1/100 of the non-relevant documents contain $t_k (u_k = 0.01)$, then $t_k$’s weight, $w_k$ (by the BI formula) is log $(11)$. If $t_k$ is much lower frequency, e.g., $p_k = 0.01$ and $u_k = 0.001$), then $w_k$ is approximately equal to log $(10.1)$. The weight of a low frequency term can be approximately the same as the weight of a high frequency term, as long as the ratio of their occurrence in relevant and non-relevant terms, is the same. Harman [SIGIR ‘92] notes the same problem with probabilistic term weighting.

In general, the SMART system weighting scheme described above allows one weighting scheme to be applied to query terms and another weighting scheme to be applied to document terms. However, as Buckley et al. [SIGIR ‘94] point out, the Rocchio formula involves adding query weights to document weights so the same weighting scheme, e.g., ltc, must be applied to both query and document terms if Rocchio is to be used for relevance feedback. (This is in contrast to the usual scheme, e.g., lnc-ltc as described in section 6.3, in which the idf is a factor in the query term weight, but not in the document term weights.)

The effect of Rocchio re-weighting is to increase the weights of terms that occur in relevant documents and to reduce the weights of terms that occur in non-relevant documents. This works well if all the relevant documents are clustered near each other in document space,[Salton, ATP 1979] In that case, relevance re-weighting using the
Rocchio formula will move the query vector toward the centroid of the cluster of relevant documents and away from the centroid of the non-relevant documents. However, if the relevant documents are not tightly clustered, the “optimum” query may not be effective at filtering. Even worse, repeated re-weighting may cause the query to wander around the document space, never settling down, because one set of relevant documents may pull the query this way, another set may pull it that way. One possible solution is to use a technique like LSI that captures term dependencies; relevant documents may be far more tightly clustered in a document space whose dimensions are LSI factors than in a conventional document space whose dimensions are actual terms occurring in the documents. Another possibility is to modify the document vectors by adding terms drawn from the user’s query or application domain to the indexes of documents judged relevant. The effect is to move relevant documents closer together in document space, and move non-relevant documents farther away. [Salton, ATP, 1979] Of course, this approach won’t improve performance for the original query, but it will help if the user submits similar queries in the future. A third approach is to cluster the retrieved relevant documents (see section on clustering). If two or more well-defined clusters are detected, one can split the original query into multiple queries, one for each identified cluster, and then proceed with normal relevance feedback [Salton, ATP, 1989].

Other re-weighting formulas can be effective. In general, term ordering formulas are potentially term re-weighting formulas. The BI probabilistic formula can be applied to relevance feedback data as described above in the section on probabilistic approaches. This formula can be used as both a term ordering formula to select good terms for query expansion, and as a formula for re-weighting query terms after each query iteration. [Sparck Jones, JDoc, 1979] According to Harman [SIGIR ‘92], the effectiveness of “query expansion using the probabilistic model seems to be heavily dependent on the test collection being used … Collections with short documents … generally perform poorly, probably because there are not enough terms to make expansion effective.” Salton and Buckley [JASIS, 1990] note that probabilistic re-weighting does not make use of such useful information as the weights of terms assigned to retrieved documents. “Furthermore, the set of relevant retrieved items [documents] is not used directly for query adjustment … Instead the term distribution in the relevant retrieved items [documents] is used indirectly to determine a probabilistic term
weight.” They claim that probabilistic relevance feedback is less effective than vector, e.g., Rocchio, relevance feedback, perhaps for these reasons.

Haines and Croft [SIGIR ‘93] used $rtf^*idf$ to weight query expansion terms, where $rtf$ is the frequency, i.e., total number of occurrences, of the given term in relevant documents. The $rtf^*idf$ is similar to Harman’s “noise*frequency”. Note that they could not have used either BI or Rocchio for re-weighting because they were running their queries against an inference network filtering engine. For an inference network, “the weight associated with a query term is used to estimate the probability that an information need is satisfied given that a document is represented by that term.” Hence, relevance feedback involves re-estimation of that probability rather than estimation of the probability that a document described by a given term is relevant to the given query (as in the BI formula). Moreover, non-relevant documents are not considered at all in their inference network. [Haines and Croft, SIGIR ‘93] Instead, “relevance feedback using the inference network model adds new terms as parents of the query node … and re-estimates the relative weights of the parents’ contributions to [a] weighted sum” representing the belief that the information need expressed by that query is satisfied by a given document.

Buckley et al. [TREC 5] have tried to improve re-weighting by applying relevance feedback not to the entire collection, but to documents in a vector sub-space around the query, called a query zone. This is a loose region around the query, such that documents in the region are not necessarily relevant to the query, but are (it is hoped) related to the query, e.g., they may be about the application domain to which the query belongs. For example, a document about computer monitors is not relevant to a query such as, “Which disk drive should I get for my Mac?” but query and document both belong to the domain of computer hardware, and hence may be in the same query zone. Use of query zone may improve re-weighting in two ways: First, a term like “computer” that is very common in the query zone, but much less common in the larger collection will be substantially downweighted, reflecting the fact that the term is good for distinguishing the domain, but not good for distinguishing relevant from non-relevant documents within the domain. Second, a term may be common in the collection as a whole (hence, not good for distinguishing the domain), but very good for distinguishing the relevant from non-relevant documents within the domain. They offer the example of tire for the query
recycling of tires. Such a term will have its weight substantially increased by the query zone approach. Clearly, the choice of a good similarity measure for inclusion in the zone is the key to success with this approach.

Buckley and Salton [SIGIR ‘94] point out that:

[1]he same relevance feedback techniques can be used in the routing environment, in which a user may have a long-lived information need and is interested in any new documents that match the need. In this case, the user’s query [initially weighted by the training set] can be constantly updated by the system as it receives relevance information about new documents seen by the user. Over the life of the query, thousands of documents could conceivably be returned to the user for relevance judgments.

In a document routing or classification environment, relevance feedback is often provided by a (relatively large) “training set” or “learning set” of documents whose relevance or non-relevance has been judged before the actual routing begins. In a test and evaluation environment such as the TREC conferences, it is common to break a document collection in half, with one half serving as the “training set” and the other half serving as the “test set” on which the routing abilities of a trained system (or an enhanced, refined query) are to be tested. The crucial point is that the system is trained (at least initially) on a different set of documents than the ones on which it will be required to perform. Hence, there is a danger that the system will be so perfectly “overfitted” to the training set that it will perform very poorly against the “real” documents. Buckley and Salton [SIGIR ‘95] point to an extreme example of overfitting: an information need expressed as a boolean expression in which each known relevant document in the training set is represented by the AND of all its index terms, and these ANDs are then ORed together. Such a query is guaranteed to retrieve all of the relevant documents in the training set, but it has been so specialized that it will do very poorly against new, incoming documents.

Buckley and Salton [SIGIR ‘95] try to avoid the danger of overfitting by what they call Dynamic Feedback Optimization (DFO). They generate an initial feedback query by expanding the initial query using the $N$ best terms from the known relevant documents in the training set. (Their ordering criterion for “best” terms is number of occurrences of the given term in the relevant documents, i.e., $rtf$.) They weight the terms using the
Rocchio approach. Then (this is where DFO comes in) they refine the query weights by a process that does not involve adding any additional terms. Instead, they run a series of passes on the existing set of query terms and weights. On each pass they try the effect of increasing each term weight by a factor which is fixed for each pass but may change (be reduced) from one pass to the next. They test each term weight increase by running the query with just that change against the training set. At the end of a pass, they preserve the term weight increases that improved performance against the training set as measured by average recall-precision on the top $X$ documents (200 in their experiments). Number of passes, percentage weight increase per pass, Rocchio coefficients ($A$, $B$, $C$) etc., are parameters that can be varied. Interestingly, Buckley and Salton note that the run in which the weights were increased most on each pass produced the best “retrospective” performance on the training set, but the worst performance on the test set, a classic example of overfitting! They conclude that limiting the weight increase on each pass is an essential element of DFO.

### 3.3 Expansion/Refinement of Boolean and Other Structured Queries

The previous section discussed the expansion and re-weighting of vector space, e.g., term-based queries. Similar techniques, e.g., relevance feedback, can be used to expand and re-weight structured queries, e.g. boolean [Salton, ATP, 1989] [Salton, IP&M, 1988], and phrase-structured queries. [Haines and Croft, SIGIR ‘93]

Salton expands a boolean query by extracting terms from retrieved documents judged relevant as in the vector case. However, instead of merely ranking the terms as candidates for expansion as in the vector case, he uses the term “postings”, i.e., the number of relevant documents indexed by the given term ($rdf$), to estimate the number of relevant documents likely to be retrieved by various boolean combinations, e.g., for three terms $t_i$, $t_j$, and $t_k$, the combinations ($t_i$ AND $t_j$), ($t_i$ AND $t_k$), ($t_j$ AND $t_k$), and ($t_i$ AND $t_j$ AND $t_k$) are applicable. The combinations that are estimated to retrieve the most additional relevant documents are then ORed to the original query. Obviously, the number of additional terms must be limited to prevent a combinatorial explosion of boolean AND terms. Also, it is clear that boolean expansions generated in this way do not reflect the full semantics that might be derived from a natural language statement of the information need using the same terms.
Strict boolean expressions do not have weights. However, extended booleans do, e.g., the $p$-norm model discussed earlier. The components of a $p$-norm extended boolean query can be weighted by assigning different values of the parameter $p$ to different clauses. A higher value (Salton suggests $p = 2.5$) can be assigned to more important clauses, “implying a stricter interpretation of the boolean operators;” less important clauses can be assigned a lower $p$ value, e.g., $p = 1.5$, meaning that their boolean operators will be interpreted more loosely (closer to a vector space interpretation).

Haines and Croft [SIGIR ‘93] ran some relevance feedback tests on structured queries in which the structure consisted of phrase or proximity operators rather than the standard boolean operators. In other words, the query was a term vector in which some of the terms were phrases or single terms in proximity. The tests were run on an inference network search engine (see section on inference networks). They used the classic query expansion and re-weighting technique except that the “terms” added were not single words but phrases or groups of words in proximity. A phrase is a statistical or syntactic co-occurrence of words. The words in a phrase may but need not satisfy a proximity relationship. [Croft et al., SIGIR ‘91] Croft et al. found that relevance feedback improved the performance of such structured queries but not as much as for queries composed of single words.

3.4 Re-Use of Queries

A lot of effort goes into refining, expanding, and optimizing a query using the methods described above, particularly the methods that require user relevance judgments. Yet once the optimized query is executed to the user’s satisfaction it is typically thrown away. (The one notable exception to this is the routing or classification case where a query or topic specification is used on an ongoing basis to identify relevant incoming documents and direct them to the appropriate user.)

Raghavan and Sever [SIGIR ‘95] suggest that these “past optimal queries” should be saved for Information Filtering (IF) applications too. Their reasonable assumption is that user information needs will recur. They call queries that satisfy such recurrent needs “persistent” queries. They propose storing optimal persistent queries in a query data base which they call a “query base;” each persistent query in the query base would be associated with the documents that it had previously retrieved. (Obviously, it would not be necessary to store the actual retrieved documents for a given persistent query in the query base, only logical pointers to these documents.)
Raghavan and Sever identify two ways in which this query base could be used in conjunction with a new user query: (1) “If there exists a persistent query with which [the] user query has sufficient similarity, then the filtering output that is associated with the persistent query is presented to the user.” (2) If there is no persistent query in the query base close enough to the user query to provide candidate output as in case 1, the system can find the persistent query that is closest to the user query, and use this persistent query as the starting point of a search through query space for an optimal query via a “steepest descent” search method. Note: The search through query space is not a search through the query base; if the optimal query for which we were searching was already in the query base, this would be case 1. The search method is described briefly below.

What does it mean for two queries to exhibit “sufficient similarity?” Raghavan and Sever don’t specify a similarity threshold; indeed, the threshold might depend on the application domain. However, they do make a valuable point: Most of the IF literature, especially the literature devoted to vector space methods, assumes that a query can be viewed as “just another document;” in other words, queries and documents are viewed as occupying the same space, and the same similarity measures, e.g., cosine similarity, are applied whether two documents are being compared or whether a document is being compared to a query. However, the IF literature seldom addresses the issue of computing similarity between two queries. Raghavan and Sever argue that a different kind of similarity measure is required for comparing queries, and they propose such a measure. Then, the threshold for “sufficient similarity” in a given application domain can be expressed in terms of their proposed similarity measure.

Their argument is based on another valuable idea: the concept of a “solution region”. Briefly, the search for an optimal query to satisfy the user’s need will yield not a single optimal query but a region of query space containing optimal queries. This region is called the “solution region”. An important property of a solution region $S$ is that if two query vectors $Q$ and $Q'$ are both members of $S$, then $kQ$ ($k > 0$) and $Q+Q'$ are members of $S$ too. On the other hand, cosine similarity is not preserved by such a transformations. Hence, cosine similarity can lead to various anomalies when queries are compared with reference to a solution region. For example, two different “optimal” queries will not be computed as exactly similar (identical) according to cosine similarity even though they
are both in the solution region. Moreover, given an “optimal” query \( Q_{opt0} \) in the solution region, and two queries \( Q_1 \) and \( Q_2 \) outside the region, cosine similarity may say that \( Q_1 \) is closer (more similar) to \( Q_{opt0} \) than \( Q_2 \) even though \( Q_2 \) is closer to the solution region than \( Q_1 \). Their proposed query similarity measure overcomes these difficulties.

The essential idea is to compute a normalized “distance” between two queries \( Q_1 \) and \( Q_2 \) based on comparing not the queries themselves but their respective filtering sets. The filtering sets can be compared either on the basis of their IF engine rankings, on the basis of the user’s relevance judgments, or on the basis of some combination of the two measures. For example, they propose a normalized distance measure based on ranking such that the distance between \( Q_1 \) and \( Q_2 \) will be minimum (zero) if they produce identical rankings of common retrieved documents, and maximum (one) if they are in complete disagreement in their rankings. They propose a normalized distance measure based on relevance judgments such that distance will be minimum (zero) if the set of documents judged relevant in the filtering set for \( Q_1 \) is identical to the set of documents judged relevant in the filtering set for \( Q_2 \); the distance for the relevance measure will be maximum (one) if there is no document judged relevant that is common to the filtering sets of \( Q_1 \) and \( Q_2 \). In a combined measure, IF engine rankings for documents retrieved by \( Q_1 \) and \( Q_2 \) are compared as before, but only for document pairs such that the first is drawn from the set of relevant retrieved documents, and the 2nd from the set of non-relevant retrieved documents. In all three cases, the similarity of \( Q_1 \) and \( Q_2 \) is simply \( 1—distance \).

How do they propose to search through query space for an optimal query? Their method assumes that the user supplies more relevance information than is usual in relevance feedback methods. As in other relevance feedback methods, the initial user query goes through a series of refinements. As in other feedback methods, each refinement of the query is applied to the document collection to obtain a set of retrieved documents. And as in other feedback methods, the user provides relevance judgments for each successive filtering set. However, the user supplies not just a judgment as to whether each document in the filtering output is relevant or non-relevant, but pairwise preferences: given any pair of documents \( D_1 \) and \( D_2 \) in the filtering set, the user specifies either \( D_1 \geq D_2 \) (meaning \( D_1 \) is either preferable to \( D_2 \) or equally good) or \( D_2 \geq D_1 \). If \( D_1 \geq D_2 \) and \( D_2 \) is not \( \geq D_1 \),
then $D_1 > D_2$ ($D_1$ is preferable to $D_2$). Transitivity is assumed to hold, i.e., if $D_1 \geq D_2$ and $D_2 \geq D_3$, then $D_1 \geq D_3$, so the number of preferences the user needs to specify does not explode combinatorially. (Also, note that this reduces to the conventional case if the user divides the filtering set into a relevant set and a non-relevant set such that each document in the relevant set is preferable to every document in the non-relevant set.) The query optimization process starts by choosing as the initial query, not the query specified by the user, $Q_{user}$, but the query from the query base that has the maximum similarity to $Q_{user}$. Call this $Q_0$. Then the $k$th iteration, $k = 0, 1, 2$, etc., consists of looking for document pairs $D_1, D_2$ in the filtering set for $Q_k$ such that $D_1 > D_2$ but the IF filtering engine ranks $D_2$ above (more relevant to $Q_k$ than) $D_1$. For each such pair, the difference vector $D_1 - D_2$ is added to $Q_k$. (So, if a given term $t_a$ has a higher weight $w_{a1}$ in $D_1$ than its weight $w_{a2}$ in $D_2$, then $w_{a1} - w_{a2}$ will be added to the weight of $t_a$ in $Q_k$. Similarly, if for some other term $t_b$, $w_{b1} < w_{b2}$, then the weight of $t_b$ in $Q_k$ will be reduced by $w_{b2} - w_{b1}$.

In other words, the query weights for the next iteration $k+1$ are moved in the direction of each “preferable” document $D_i$.) This iterative process continues until it reaches a $Q_n$ whose filtering set is such that whenever $D_1 > D_2$, $D_1$ is ranked by the IF engine above $D_2$. In other words, we have reached a query for which the user’s subjective relevance judgments agree with the IF engine’s rankings. This $Q_n$ is considered the optimal query $Q_{opt}$. 