A COMPARISON OF TWO METHODS FOR BOOLEAN QUERY RELEVANCE FEEDBACK

G. Salton*
E.A. Fox†
E. Voorhees*

TR 83-564
July 1983

Department of Computer Science
Cornell University
Ithaca, NY 14853

*Department of Computer Science, Cornell University, Ithaca, New York 14853.

†Department of Computer Science, Virginia Polytechnic Institute and State University, Blacksburg, Virginia.

This study was supported in part by the National Science Foundation under grants IST 81-08696 and IST 80-17589.

A Comparison of Two Methods for Boolean Query Relevance Feedback

G. Salton*, E. A. Fox† and E. Voorhees*

Abstract:

The relevance feedback process uses information derived from an initially retrieved set of documents to improve subsequent search formulations and retrieval output. In a Boolean query environment this implies that new query terms must be identified and Boolean operators must be chosen automatically to connect the various query terms. In this study two recently proposed automatic methods for relevance feedback of Boolean queries are evaluated and conclusions are drawn concerning the use of effective feedback methods in a Boolean query environment.

1. Relevance Feedback Methods

Automatic relevance feedback methods were first introduced in vector processing environments nearly twenty years ago. In the vector processing system a query Q, or document D, is represented by a set of possibly weighted, independent terms, unrelated by Boolean operators or other relationship indications. Rocchio has shown that an approximation to an optimal query vector may be generated by adding to an initial query formulation terms extracted from previously retrieved documents identified as relevant to the query, and analogously by subtracting from an initial query formulation terms extracted

* Department of Computer Science, Cornell University, Ithaca, New York 14853.
† Department of Computer Science, Virginia Polytechnic Institute and State University, Blacksburg, Virginia.

This study was supported in part by the National Science Foundation under grants IST 81-08696 and IST 80-17589.

from previously retrieved nonrelevant documents. [1] The following simple feedback process was proposed:

a) given an initial query formulation Q_0, retrieve a set of items N' from the collection;

b) identify a set of n_1 relevant documents from among the N' retrieved ones, and a set of n_2 nonrelevant documents;

c) use vector addition to generate the average vector for the relevant items $(1/n_1 \sum_{Rel} D^{(i)})$ and the average vector for the nonrelevant items $(1/n_2 \sum_{Nonrel} D^{(i)})$, where $D^{(i)}$ represents a particular document vector;

d) produce a new query formulation Q_1 from the initial query Q_0 by adding the average relevant and subtracting the average nonrelevant items as follows:

$$Q_1 = Q_0 + \frac{1}{n_1} \sum_{Rel} D^{(i)} - \frac{1}{n_2} \sum_{Nonrel} D^{(i)} \qquad (1)$$

The formulation of equation (1) may add new terms to the query derived from the set of identified relevant documents, or else it may increase or decrease the weight of terms already present in the initial query formulation. One may expect that the new query formulation will exhibit a greater similarity with the relevant items, and a smaller similarity with the nonrelevant items than the original query formulation. The experimental evidence obtained over the years with the automatic vector reformulation process indicates that substantial increases in retrieval effectiveness are produced for the first few relevance feedback iterations. [2-4]

Relevance feedback methods have also been used advantageously in probabilistic information retrieval. [5-6] In that case, an explicit feedback query is not constructed. Instead, a relevance weight, sometimes known as term relevance or term precision, is computed for the terms contained in previously retrieved relevant, or nonrelevant documents. The relevance weight of a given term x_i is determined as a function of the occurrence probabilities of that term in the relevant and nonrelevant documents. Correspondingly, a document relevance factor may be determined for each document as a function of the relevance weights of the individual terms contained in that document.

The following sample relevance feedback system may be used in probabilistic retrieval environments:

a) given an initial query formulation, perform a retrieval run, and identify some documents as relevant, or nonrelevant;

b) for each term x_i contained in some relevant or nonrelevant document, compute the occurrence probabilities p_i and p_i' of term x_i in the relevant and nonrelevant document subsets, respectively; compute the term relevance weight for term x_i as a function of the values p_i and p_i' (when the terms are assumed to be independently assigned to the relevant and nonrelevant documents, an optimal term relevance weight is computable as $\log[(p_i/1-p_i) * (p_i'/1-p_i')]$) [7];

c) for each document $\underline{x}$ of the collection, compute the document relevance factor $g(\underline{x}) = P(\underline{x}|\text{Rel})/P(\underline{x}|\text{Nonrel})$ as the ratio of the occurrence probabilities of $\underline{x}$ in the relevant and nonrelevant document subsets (when the terms are assigned independently to the documents of a collection, $P(\underline{x}|\text{Rel}) = \prod_i P(x_i|\text{Rel})$ where the product is taken over all document

terms; a similar expression holds for P($\underline{x}$|Nonrel));

d) arrange the documents in decreasing order of the function g($\underline{x}$).

The evidence indicates that the relevance feedback process is useful in probabilistic information retrieval, as it is in the vector processing system. [5-6]

When the relevance feedback system is extended to a Boolean retrieval environment where the queries are expressed as term sets interconnected by the operators and, or and not, additional problems not met in vector processing, or probabilistic retrieval must be considered. First, the conventional Boolean retrieval system does not produce ranked retrieval output in decreasing order of some retrieval function. This makes it difficult to use a specific number of previously retrieved documents for feedback purposes. And second, the need arises in Boolean systems to choose not only the terms to be used in a query reformulation, but also the Boolean operators relating the terms. The more complicated Boolean environment is examined in the next section.

2. Boolean Query Relevance Feedback

A number of attempts have been made in the past to develop adequate methods for Boolean query relevance feedback. [8,9] Most recently a method was proposed (termed the "Dillon method" in the remainder of this note) which uses the previously described probabilistic approach as a basis. [10] In particular, an initial Boolean search run is first used to retrieve a number of items which are judged to be relevant or nonrelevant to the particular query. For each term included in a previously retrieved item a relevance

weight, known as the prevalence (prev), is computed based on the occurrences of the term in the relevant, or nonrelevant previously retrieved documents. A number of threshold values in the prevalence, known as floors, are then defined and the terms are placed into categories according to their prevalence values. Finally a new Boolean query is constructed by taking the disjunction of the terms in each category, either singly, if included in the highest prevalence category ($prev_t \geq$ floor 1), or as anded pairs if the terms are included in the second highest prevalence category (floor 1 > $prev_t \geq$ floor 2), or as anded triples if the terms are in the third highest prevalence category, and so on. In other words, a final query would be phrased as:

$$(S_1 \text{ or } S_2 \text{ or} \ldots S_s) \text{ or } [(P_1 \text{ and } P_2) \text{ or } (P_3 \text{ and } P_4) \text{ or} \ldots \text{or} (P_{p-1} \text{ and } P_p)]$$
$$\text{or } [(T_1 \text{ and } T_2 \text{ and } T_3) \text{ or } \cdots \text{ or } (T_{t-2} \text{ and } T_{t-1} \text{ and} T_t)] \qquad (2)$$

where the s single terms (S_i) have highest prevalence, the terms included in the p pairs P_j fall into the next highest prevalence category, the terms included in the t triples (T_k) are in the third highest prevalence category, and so on.

Initially, provisions were made in the Dillon process for terms of negative prevalence, that is, those included mostly in the nonrelevant items. Such terms would be included in a query using Boolean negation (and not operators). In practice, the well-known problems which arise in handling negated terms [11,12] proved too difficult to solve, and the attempt to use terms of negative prevalence was abandoned. In fact, in the experiments actually carried out with the Dillon methods only two prevalence thresholds were used, called floor 1 and floor 2, respectively. [10] Terms with a prevalence at least equal to floor 1 are treated as single terms, and terms with prevalence

smaller than floor 1 but at least equal to floor 2 are combined into anded pairs.

The computation of the term prevalence measure may be illustrated by using the frequency chart of Table 1 describing the occurrence characteristics of term t in a given set N' of retrieved documents. (In this study, primed variable names refer to quantities valid in the set of retrieved documents, whereas unprimed variables are valid for the complete document collection, including both retrieved and nonretrieved items.) Table 1 shows that of the set of N' retrieved documents, R' were judged relevant to a given query, and r_t' of the R' relevant retrieved items were found to contain term t.

In the Dillon method the term prevalence, $prev_t$, is computed as the difference of the ratios of the proportion of relevant retrieved containing term t minus the proportion of nonrelevant retrieved containing term t, the whole expression being divided by $freq_t$, the total frequency of occurrence of term t in the whole collection ($freq_t = n_t$), or

$$prev_t = \frac{\frac{r'_t}{\min(R', freq_t)} - \frac{n'_t - r'_t}{\min(N'-R', freq_t)}}{\ln freq_t} \qquad (3)$$

It is clear that the larger the number of occurrences of a term in the relevant retrieved items and the smaller the number of occurrences of the term in the nonrelevant retrieved and in the whole collection, the larger will be the term prevalence. The factor ($\ln freq_t$) in the denominator of expression (3) acts as an inverse document frequency (IDF) factor, which is known to represent a good term weighting criterion in the absence of other more specific information about the relevance or usefulness of the various terms. [13]

An outline of the Dillon relevance feedback algorithm is shown in the program of Fig. 1. The prevalence normalization procedure in step 10 is designed to make it possible to use common prevalence threshold (floor) values for a variety of different user queries. The following properties of the Dillon method are apparent:

a) The terms included in the original user queries (or in any subsequent query reformulation) are not automatically included in the query reformulation process, unless these terms also happen to occur in the items retrieved in a given search. (As will be seen later, query terms could be automatically considered for query reformulation by using a special parameter (the qcount) to adjust the occurrence counts of all query terms in the relevant retrieved documents and in the whole collection.)

b) The term prevalence formula of expression (3) becomes negative when the term occurrences in the nonrelevant retrieved items exceed those in the relevant retrieved documents. Because of the problems inherent in negative relevance feedback, it is advisable to concentrate on the computation of nonnegative prevalence values.

c) All single terms (as well as all anded pairs, anded triples, etc.) are included in a reformulated query depending only on the prevalence values of the individual terms. This implies that a reformulated query may consist of a large number of Boolean clauses--possibly many hundreds of clauses, and that its retrieval performance may then be hard to control.

d) Finally, the distinction between a clause consisting of a single query term alone, and clauses consisting of the conjunction of several terms, depends on the setting of the prevalence thresholds (floors). But no obvious way exists for choosing effective prevalence thresholds automatically.

A different Boolean relevance feedback method, known as the DNF (disjuctive normal form) method was recently developed. The DNF process avoids the pitfalls of the Dillon algorithm while maintaining the basic formulation of expression (2) in terms of a disjunction of single terms, anded pairs, anded triples, and so on. [14,15] The DNF algorithm outlined in the programs of Figs. 2 and 3 exhibits some minor and one major conceptual difference with the Dillon algorithm. Among the minor differences are:

a) the use of a qcount parameter which adjusts the occurrence characteristics of the query terms (see steps 3 and 4 of Fig. 2) by specifying that term occurrences in a query are equivalent to occurrences in m relevant documents, the qcount setting being equal to m. (In practice a qcount of 2 produces good retrieval performance [15].)

and

b) The use of a somewhat different relevance weight, $relwt_t$, equivalent to the term prevalence in the Dillon method, which utilizes term occurrences in the whole collection instead of occurrences in the non-relevant documents previously retrieved (see step 5 of Fig. 2), and the elimination of terms whose relevance weight proves too small (step 6 of Fig. 2).

The relevance weight currently used is defined as

$$relwt_t = \left[\frac{rq'_t}{R'+qcount} - \frac{freq_t}{N} \right] \ln \left[\frac{N}{freq_t+10} \right] \quad (4)$$

which represents the proportion of relevant documents in which term t occurs minus the proportion of all documents in the collection in which the term occurs. [16] The multiplicative logarithmic factor on the right-hand side of expression (4) represents once again an inverse document frequency factor.

The major concept distinguishing the DNF from the Dillon method is the use of a retrieval parameter T specifying the number of desired documents which the feedback query is ideally expected to retrieve. The T parameter is then used to control the number of clauses to be included in the reformulated query. (Typical values for T, the desired number of documents, might be 50 or 100 [15].)

More specifically, in addition to the term relevance weight, an adjusted postings frequency $freqq_t$, is generated which specifies the expected number of items retrieved by term t. For anded pairs, st, and anded triples, rst, the adjusted postings frequency is defined as $(freqq_s \cdot freqq_t)/N$, and $(freqq_r \cdot freqq_{st})/N$ respectively, representing in fact the expected postings frequencies under the simplifying assumption that the terms r, s, t occur independently of each other in the documents of a collection. A further approximation then leads to the assumption that the number of documents which may be expected to be retrieved by a disjunction of p clauses is simply the sum of the adjusted postings frequencies of the individual clauses. (Thus if clauses (A) and (B and C) are expected to retrieve 10 and 5 documents respectively, the query (A or (B and C)) would be expected to retrieve a total of 15 items in all. This assumption leads to a correct computation of the total number of retrieved items only when the documents retrieved by the individual

clauses happen to form disjoint sets.

The DNF process thus consists of two major steps:

a) In step one, corresponding to the chart of Fig. 2, the single terms (currently limited to 70) with the best relevance weight are chosen and the adjusted postings frequencies are computed. The anded term pairs corresponding to the 70 best singles are generated, and adjusted posting frequencies are obtained for these pairs. Finally, the anded triples corresponding to the 70 best singles and 70 best pairs are generated, and the corresponding postings frequencies are once again computed.

b) In step two, corresponding to the chart of Fig. 3, a broad query is initially constructed consisting of the conjunction of all good single terms, and the number of estimated retrieved items (the "estret" parameter of Fig. 3) is compared with the number of wanted items T. So long as estret > T, a query refinement process is used to delete single terms, replacing them by anded pairs, and eventually to delete pairs replacing them by anded triples. Following each query reformulation, a new estimated number of retrieved items is computed (steps 20 and 24 of Fig. 3) which necessarily approaches T more and more closely as the query is iteratively refined. Eventually the condition $T \geq estret > T/2$ will hold and the query construction process terminates by using the disjunction of all remaining singles, pairs, and triples as the reformulated query. [14,15]

The operations of the Dillon and DNF query reformulation methods are evaluated in the following section.

3. Evaluation of Boolean Relevance Feedback

The Dillon feedback process outlined in Fig. 1 and the DNF method of Figs. 2 and 3 were evaluated by using two iterations of relevance feedback with four test collections in different subject areas. The number of documents involved ranged from 1033 for the Medlars collection to 12684 for Inspec. Relevant collection statistics are included in Table 2. A feedback iteration was carried out in each case by running the original Boolean query, identifying a random set of 10 items from among the retrieved set (or taking all retrieved items for queries that retrieve less than 10 documents), judging those 10 items for relevance, and using the term occurrences in the sets of relevant or nonrelevant retrieved items to generate a first iteration feedback query. The performance of the relevance feedback process could then be evaluated by comparing the effectiveness of the search results obtained with the original queries and with the first iteration feedback queries.

The exact same process can be repeated by taking as original input the results of the first iteration feedback queries, looking at ten more items not yet seen in the original feedback step, and using the relevance judgments for the total of 20 items seen so far to construct second iteration feedback queries. To evaluate the performance of these second iteration feedback queries the corresponding search results must be compared with the results obtained for the first iteration feedback queries.

In a relevance feedback evaluation it is important to insure that the number of documents examined for relevance is the same for the runs being compared, and that previously identified relevant documents that were originally used for the construction of a given feedback query are not given extra weight during the feedback process. This can be done by comparing the output of a

given feedback iteration with a continuation of the search results obtained for the previous query set that was used to construct the feedback queries. For example, the search results obtained with the first iteration feedback queries are compared with the results obtained by looking at additional documents retrieved by the initial query set (beyond the 10 top documents that were used to construct the feedback queries). [15,17]

To eliminate the effect on the retrieval output of the relevant documents initially retrieved which enter into the construction of the feedback queries, a partial rank freezing process is used which counts these documents equally in the two runs being compared. [15,17] Essentially, the output of the first feedback run is compared against a "continued once" ranking of the original output obtained in the following way:

> take the output of the original query set; consider the top 10 retrieved items; freeze the ranks of the relevant items retrieved in the top 10; eliminate the nonrelevant items from the top 10; replace the missing nonrelevant in the top 10 by lower ranking items; evaluate the reranked output in terms of recall and precision.

Correspondingly, the second iteration feedback run may be compared against a "continued twice" ranking, in which the "continued once" ranking is used as input, the next 10 items not yet seen are judged for relevance, once again the ranks of the relevant in all items seen so far are frozen, the nonrelevant are eliminated, and the empty ranks are filled by lower ranking new items. A total of 10 items are thus judged for relevance for the first feedback iteration and a total of 20 for the second iteration.

The performance of the various search runs is evaluated by using recall

and precision figures, or more precisely by computing the search precision averaged over a given query set at fixed values of the recall. Table 2 shows that for a number of available queries no relevant items could be found among the top 10 retrieved documents. Since the Dillon feedback process, as currently conceived, cannot be used to construct a feedback query in the absence of some previously retrieved, relevant items, the query set used for the current experiments was reduced to 25, 23, 24, and 54 for the Medlars, CISI, CACM, and Inspec collections respectively.

Typical recall-precision tables are included in Table 3 for two particular relevance feedback runs performed with each of the four collections. In each case, the precision is averaged over the particular query set at fixed recall values ranging from 0.1 to 1.0 in intervals of 0.1. A common DNF procedure is used with a qcount value of 2, a wanted number of 50 retrieved (T value) for the 3 small collections, and a T value of 100 for the larger Inspec collection. The first iteration feedback query is designated as Q_1 in the headings of Table 3, and the notation $Q = Q_0 \vee Q_1$ indicates that the feedback query actually used for search purposes consists of the disjunction of the initial query Q_0, and the feedback query Q_1 constructed by the methods outlined in Figs. 2 and 3. (The DNF method could also be used when $Q = Q_1$ implying that the feedback query actually used includes only the feedback component.)

In the output of Table 3, the Dillon run being compared uses prevalence thresholds of 0.9 for floor 1 and 0.5 for floor 2 with the Medlars and CISI collections. The corresponding thresholds used with CACM and Inspec are 0.8 and 0.6 respectively. The run parameters used in Table 3 represent the best prevalence threshold for the Dillon process for each of the four test collec-

tions. Table 3 shows that the DNF process produces substantially better first iteration feedback queries than the corresponding Dillon process. Since this result holds for all four test collections, one may expect that the same outcome will generally hold for any collection.

It is seen in Table 3 that the differences between average precision values for the two relevance feedback systems are smallest for the CACM collection where the Dillon process is actually slightly superior at a low recall of 0.1. Overall the advantage of the DNF process ranges from an average of about 20 percent for CACM to about 50 percent for Inspec.

The improvements obtained by the Boolean relevance feedback methods are outlined in Tables 4 to 7 covering the four test collections. To save space only a single average precision value is included for each case, replacing the full recall-precision tables used in Table 3, and representing the average precision at a low recall of 0.25, a medium recall of 0.50, and a high recall of 0.75. The first feedback iteration is shown in the top part of Tables 4 to 7, whereas the bottom sections of the Tables cover the second feedback iteration. As explained previously, the operations of the original Boolean queries continued once are used as a base run against which the first iteration feedback queries are compared. The original Boolean queries continued twice are used as a base for the second feedback iteration.

The output of Tables 4 to 7 shows that the Dillon relevance feedback fails to perform adequately for 3 of the 4 collections. Only for the Medlars collection do the Dillon feedback queries perform somewhat better than a continuation of the initial query output, the advantage there ranging from about 5 percent to 9 percent in the first iteration depending on the floor values used. In the second feedback iteration the Dillon relevance feedback process

actually produces an advantage of 53.5 percent over the continuation of the initial output using floor values of 0.8 and 0.6 for the Medlars collection (last line of Table 4). For the CISI, CACM, and Inspec collections, the Dillon feedback process produces a deterioration in performance of about 18 percent for CACM and 50 to 60 percent for CISI and Inspec, respectively. In other words, for those collections it would be better not to go through the Dillon feedback query construction method, but instead to keep examining more documents retrieved by the initial Boolean queries.

The situation is quite different for the DNF feedback method. For that relevance feedback method, the improvements in the first feedback iteration range from over 80 percent for Medlars to 9 or 10 percent for the other 3 collections when the feedback query actually used for a first iteration search is defined as $Q_0 \vee Q_1$ (in other words, when the initial query is maintained intact as a part of the feedback query). The second feedback iteration serves further to improve the output for 3 of the 4 collections since the advantage now ranges from 112 percent for Medlars, to about 7 percent, 11 percent, and 13 percent for CISI, CACM and Inspec respectively. Since the DNF relevance feedback process brings improvements for all test collections, one may expect a similar performance for collections in other subject areas as well.

The last column of Tables 4 to 7 shows the average number of clauses contained in the first and second iteration feedback queries for each of the methods under consideration. The four values shown in the right-most column of the Tables correspond to the average number of single terms, anded pairs, anded triples, and finally the total number of all clauses contained in the feedback queries, a clause being defined as a single, anded pair, or anded triple. The basic difference in the number of clauses for the Dillon and DNF

methods is immediately apparent. For the DNF method used with retrieval thresholds of T = 50, or T = 100 for Inspec, the total average number of query clauses is always less than 10*. For the Dillon process, on the other hand, the number of clauses included in a feedback query varies widely depending on the floor values used. When the value of floor 2 is low (0.5), many hundreds of clauses may be generated. In fact over 500 different clauses are generated for Medlars and ISI in certain cases, whereas nearly 200 clauses are used in some cases for CACM and Inspec. Boolean queries with such a large number of clauses may lack the necessary homogeneity in subject description to render them effective in information retrieval.

The DNF query construction process outlined in Figs. 2 and 3 covers the relevance feedback process for conventional Boolean queries only. It has been shown elsewhere that substantial improvements in Boolean information retrieval are obtainable by relaxing the interpretation of the Boolean operations, and using so-called extended Boolean retrieval operations. [15,18] The DNF relevance feedback process of Figs. 2 and 3 can be applied unchanged to the extended Boolean query environment, lending potentially to much larger performance improvements than those shown in Tables 4 to 7 for the pure Boolean environment. Furthermore, the DNF query process can also be applied with small modifications to the automatic construction of initial Boolean query statements instead of the reformulation of already existing queries. [19] The DNF method illustrated in this study should prove effective in operational retrieval environments. One may hope that it will be tried in practice before long.

---

* Since the number of clauses contained in the original queries is not well-defined, a clause count is not included in the Tables for the DNF method in which the new query Q includes the original Q_0 as a component.

References

[1] J.J. Rocchio Jr., Relevance Feedback in Information Retrieval, Chapter 14, in The Smart System--Experiments in Automatic Document Processing, Prentice Hall Inc., Englewood Cliffs, NJ, 1971 (reprinted from Scientific Report ISR-9, Computation Laboratory, Harvard University, August 1965).

[2] E. Ide, New Experiments in Relevance Feedback, Chapter 16, in The Smart System--Experiments in Automatic Document Processing, Prentice Hall Inc., Englewood Cliffs, NJ, 1971.

[3] E. Ide and G. Salton, Interactive Search Strategies and Dynamic File Organization in Information Retrieval, Chapter 18, in The Smart System--Experiments in Automatic Document Processing, Prentice Hall Inc., Englewood Cliffs, NJ, 1971.

[4] G. Salton, Relevance Feedback and the Optimization of Retrieval Effectiveness, Chapter 15, in The Smart System--Experiments in Automatic Document Processing, Prentice Hall Inc., Englewood Cliffs, NJ, 1971.

[5] C.J. van Rijsbergen, D.J. Harper, and M.F. Porter, The Selection of Good Search Terms, Information Processing and Management, Vol. 17, 1981, p. 77-91.

[6] D.J. Harper and C.J. van Rijsbergen, An Evaluation of Feedback in Document Retrieval Using Co-occurrence data, Journal of Documentation, Vol. 34, 1978, p. 189-216.

[7] S.E. Robertson and K. Sparck Jones, Relevance Weighting of Search Terms, Journal of the ASIS, Vol. 23, No. 3, 1976, p. 129-146.

[8] J.T. Rickman, Design Considerations for a Boolean Search System with Automatic Relevance Feedback Processing, Proceedings of the ACM National Meeting, Association for Computing Machinery, New York, August 1971, p. 478-481.

[9] M. Dillon and J. Desper, Automatic Relevance Feedback in Boolean Retrieval Systems, Journal of Documentation, Vol. 36, 1980, p. 197-208.

[10] M. Dillon, J. Ulmschneider, and J. Desper, A Prevalence Formula for Automatic Relevance Feedback in Boolean Systems, Information Processing and Management, Vol. 19, No. 1, 1983, p. 27-36.

[11] R.A. Fairthorne, Towards Information Retrieval, Butterworths Ltd., London, 1961.

[12] C.N. Mooers, A Mechanical Theory of Language Symbols in Retrieval, Int. Conference on Scientific Information, National Academy of Sciences, Washington, D.C., 1958.

[13] K. Sparck Jones, A Statistical Interpretation of Term Specificity and its Application in Retrieval, Journal of Documentation, Vol. 28, No. 1, March 1972, p. 11-20.

[14] E.A. Fox, Extending the Boolean and Vector Space Models of Information Retrieval with P-Norm Queries and Multiple Concept Types, Doctoral Thesis, Cornell University, August 1983.

[15] G. Salton, E.A. Fox, C. Buckley and E. Voorhees, Boolean Query Formulation with Relevance Feedback, Technical Report TR 83-539, Department of Computer Science, Cornell University, January 1983.

[16] M.F. Porter, Implementing a Probabilistic Information Retrieval System, Information Technology: Research and Development, Vol. 1, No. 2, April 1982, p. 131-156.

[17] Y.K. Chang, C. Cirillo, and J. Razon, Evaluation of Feedback Retrieval Using Modified Freezing, Residual Collection, and Test and Control Groups, Chapter 17, in The Smart System--Experiments in Automatic Document Processing, Prentice Hall Inc., Englewood Cliffs, NJ, 1971.

[18] G. Salton, E.A. Fox and H. Wu, Extended Boolean Information Retrieval, Technical Report TR 82-511, Department of Computer Science, Cornell University, August 1982 (to be published in Communications of the ACM).

[19] G. Salton, C. Buckley and E.A. Fox, Automatic Query Formulations in Information Retrieval, Journal of the ASIS, Vol. 34, No. 4, July 1983, p. 262-280.

	Retrieved Relevant	Retreived Nonrelevant	
Term t present $x_t = 1$	r'_t	$n'_t - r'_t$	n'_t
Term t absent $x_t = 0$	$R' - r'_t$	$N' - R' - n'_t + r'_t$	$N' - n'_t$
	R'	$N' - R'$	N'

N' : total number of retrieved documents

R' : total number of relevant documents retrieved

r'_t : total number of relevant documents retrieved that contain term t

$n'_t - r'_t$: total number of nonrelevant documents retrieved that contain term t

Basic Parameter Set in the Set of Retrieved Documents

Table 1

Collection Statistics	Medlars (biomedicine)	CISI (documentation)	CACM (computer science)	Inspec (electrical engineering)
Number of documents	1033	1460	3204	12684
Number of original queries	30	35	52	77
Number of queries with no relevant items in top 10 retrieved	5	12	18	23
Number of queries used in current experiments	25	23	34	54
Average number of relevant documents per query in reduced query set	23.6	59.3	17.3	37.6
Average number of relevant documents retrieved in top 10 by original Boolean queries	4.6	2.6	2.8	2.8

Collection Statistics

Table 2

Recall	Medlars 1033 DNF first iteration $Q=Q_0 \vee Q_1$ qcount = 2 T = 50	Medlars 1033 Dillon first iteration Floor 1 = .9 Floor 2 = .5	Recall	CISI 1460 DNF first iteration $Q=Q_0 \vee Q_1$ qcount = 2 T = 50	CISI 1460 Dillon first iteration Floor 1 = .9 Floor 2 = .5
.1	.8476	.7246 (-15%)	.1	.3985	.2261 (-43%)
.2	.7245	.5843 (-19%)	.2	.3104	.1428 (-54%)
.3	.6310	.4319 (-32%)	.3	.2489	.0983 (-61%)
.4	.5675	.3236 (-43%)	.4	.2015	.0789 (-61%)
.5	.4745	.2581 (-47%)	.5	.1654	.0693 (-58%)
.6	.4374	.1517 (-65%)	.6	.1363	.0583 (-57%)
.7	.3733	.1100 (-71%)	.7	.0956	.0529 (-45%)
.8	.3024	.0999 (-67%)	.8	.0607	.0493 (-19%)
.9	.1536	.0713 (-54%)	.9	.0489	.0463 (- 5%)
1.0	.0755	.0346 (-54%)	1.0	.0419	.0416 (- 1%)
		average (-46.7%)			average (-40.4%)

Recall	CACM 3204 DNF first iteration $Q=Q_0 \vee Q_1$ qcount = 2 T = 50	CACM 3204 Dillon first iteration Floor 1 = .8 Floor 2 = .6	Recall	Inspec 12684 DNF first iteration $Q=Q_0 \vee Q_1$ qcount = 2 T = 100	Inspec 12684 Dillon first iteration Floor 1 = .8 Floor 2 = .6
.1	.6319	.6726 (+ 6%)	.1	.4521	.4011 (-11%)
.2	.5397	.4911 (- 9%)	.2	.3121	.1694 (-46%)
.3	.4600	.3321 (-28%)	.3	.2596	.1116 (-57%)
.4	.3986	.3021 (-24%)	.4	.2183	.0732 (-66%)
.5	.3567	.2811 (-21%)	.5	.1891	.0606 (-68%)
.6	.2975	.2344 (-21%)	.6	.1509	.0590 (-61%)
.7	.1860	.1055 (-43%)	.7	.1119	.0572 (-49%)
.8	.1501	.0962 (-36%)	.8	.0897	.0429 (-52%)
.9	.1188	.0949 (-20%)	.9	.0478	.0115 (-76%)
1.0	.1026	.0940 (- 8%)	1.0	.0117	.0110 (- 6%)
		average (-20.4%)			average (-49.2%)

Detailed Comparisons of First Iteration Relevance Feedback Methods
(for four collections)

Table 3

Medlars 1033 - 25 queries

Description of Run	Average Precision at recall of .25, .50 and .75	Percent Difference from Base Run	Average Number of Singles, Pairs, Triples, All Clauses
Original Boolean Queries continued once (a total of 10 retrieved items judged for relevance). Base run for first iteration.	.2686	-	-
First Iteration Feedback based on 10 initially retrieved items			
DNF method, $Q = Q_1$ qcount = 2, T = 50	.5064	+88.5%	(1.9, 3.8, 0.6, 6.3)
DNF method $Q = Q_0 \vee Q_1$ qcount = 2, T = 50	.4933	+83.7%	(includes original query)
Dillon feedback floor 1 = .9, floor 2 = .5	.2926	+ 9.0%	(11.7, 503.4, -, 515.0)
Dillon feedback floor 1 = .9, floor 2 = .6	.2920	+ 8.7%	(11.7, 151.0, -, 162.7)
Dillon feedback floor 1 = .8, floor 2 = .5	.2815	+ 4.8%	(15.0, 421.2, -, 436.2)
Dillon feedback floor 1 = .8, floor 2 = .6	.2843	+ 5.8%	(15.0, 80.0, -, 94.9)
Original Boolean Queries continued twice (a total of 20 retrieved items judged for relevance). Base run for second iteration feedback.	.2780	-	-
Second Iteration Feedback based on 10 items retrieved by first iteration feedback			
DNF method, $Q = Q_1$ qcount = 2, T = 50	.6194	+122.8%	(1.5, 3.6, 1.5, 6.6)
DNF method, $Q = Q_0 \vee Q_1$ qcount = 2, T = 50	.5915	+112.8%	(includes original query)
Dillon feedback floor 1 = .9, floor 2 = .5	.3749	+34.9%	(13.9, 757.5, -, 771.4)
Dillon feedback floor 1 = .8, floor 2 = .6	.4268	+53.5%	(17.9, 172.2, -, 190.1)

Comparison of Dillon and DNF Feedback Methods
Two Feedback Iterations
(Medlars 1033 collection, 25 queries)

Table 4

CISI 1460 - 23 queries

Description of Run	Average Precision at recall of .25, .50 and .75	Percent Difference from Base	Average Number of Singles, Pairs, Triples, All Clauses
Original Boolean Queries continued once (a total of 10 retrieved items judged for relevance). Base run for first iteration.	.1627	-	-
First Iteration Feedback based on 10 initially retrieved items			
DNF method, $Q = Q_1$ qcount = 2, T = 50	.1470	- 9.7%	(0.6, 1.8, 3.4, 9.7)
DNF method $Q = Q_0 \vee Q_1$ qcount = 2, T = 50	.1775	+ 9.1%	(includes original query)
Dillon feedback floor 1 = .9, floor 2 = .5	.0792	-51.3%	(5.4, 507.3, -, 512.7)
Dillon feedback floor 1 = .9, floor 2 = .6	.0644	-60.4%	(5.4, 102.7, -, 108.0)
Dillon feedback floor 1 = .8, floor 2 = .5	.0746	-54.2%	(7.5, 455.9, -, 463.4)
Dillon feedback floor 1 = .8, floor 2 = .6	.0660	-59.4%	(7.5, 73.6, -, 81.1)
Original Boolean Queries continued twice (a total of 20 retrieved items judged for relevance). Base run for second iteration feedback.	.1865	-	-
DNF method, $Q = Q_1$ qcount = 2, T = 50	.1858	- 0.4%	(0.6, 1.9, 2.1, 4.6)
DNF method, $Q = Q_0 \vee Q_1$ qcount = 2, T = 50	.2002	+ 7.3%	(includes original query)
Dillon feedback floor 1 = .9, floor 2 = .5	.0952	-49.0%	(7.2, 489.0, -, 496.3)
Dillon feedback floor 1 = .8, floor 2 = .6	.0949	-49.1%	(8.3, 90.9, -, 99.2)

Comparison of Dillon and DNF Feedback Methods
Two Feedback Iterations
(CISI 1460 collection, 23 queries)

Table 5

CACM 3204 - 34 queries

Description of Run	Average Precision at recall of .25, .50 and .75	Percent Difference from Base	Average Number of Singles, Pairs, Triples, All Clauses
Original Boolean Queries continued once (a total of 10 retrieved items judged for relevance). Base run for first iteration.	.3131	-	-
First Iteration Feedback based on 10 initially retrieved items			
DNF method, $Q = Q_1$ qcount = 2, T = 50	.3360	+ 7.3%	(2.0, 3.2, 0.7, 5.9)
DNF method $Q = Q_0 \vee Q_1$ qcount = 2, T = 50	.3457	+10.4%	(includes original query)
Dillon feedback floor 1 = .9, floor 2 = .5	.2573	-17.8%	(5.1, 190.8, -, 195.9)
Dillon feedback floor 1 = .9, floor 2 = .6	.2551	-18.5%	(5.1, 24.1, -, 29.2)
Dillon feedback floor 1 = .8, floor 2 = .5	.2560	-18.2%	(6.5, 168.9, -, 175.3)
Dillon feedback floor 1 = .8, floor 2 = .6	.2735	-12.6%	(6.5, 15.8, -, 22.2)
Original Boolean Queries continued twice (a total of 20 retrieved items judged for relevance). Base run for second iteration feedback.	.3391	-	-
Second Iteration Feedback based on 10 items retrieved by first iteration feedback			
DNF method, $Q = Q_1$ qcount = 2, T = 50	.3845	+13.4%	(1.2, 2.8, 1.8, 5.8)
DNF method, $Q = Q_0 \vee Q_1$ qcount = 2, T = 50	.3788	+11.7%	(includes original query)
Dillon feedback floor 1 = .9, floor 2 = .5	.2971	-12.4%	(5.8, 260.3, -, 266.1)
Dillon feedback floor 1 = .8, floor 2 = .6	.3386	- 0.1%	(7.4, 47.6, -, 55.0)

Comparison of Dillon and DNF Feedback Methods
Two Feedback Iterations
(CACM 3204 collection, 34 queries)

Table 6

Inspec 12684 - 54 queries

Description of Run	Average Precision at recall of .25, .50 and .75	Percent Difference from Base Run	Average Number of Singles, Pairs, Triples, All Clauses
Original Boolean Queries continued once (a total of 10 retrieved items judged for relevance). Base run for first iteration.	.1838	-	-
First Iteration Feedback based on 10 initially retrieved items			
DNF method, $Q = Q_1$ qcount = 2, T = 100	.1869	+ 1.7%	(0.9, 4.2, 2.7, 7.8)
DNF method $Q = Q_0 \vee Q_1$ qcount = 2, T = 100	.1961	+ 6.7%	(includes original query)
Dillon feedback floor 1 = .9, floor 2 = .5	.0825	-55.1%	(4.8, 25.7, -, 193.1)
Dillon feedback floor 1 = .9, floor 2 = .6	.0814	-55.7%	(4.8, 25.7, -, 30.5)
Dillon feedback floor 1 = .8, floor 2 = .5	.0878	-52.2%	(5.9, 167.1, -, 173.0)
Dillon feedback floor 1 = .8, floor 2 = .6	.0863	-53.0%	(5.9, 17.7, -, 23.6)
Original Boolean Queries continued twice (a total of 20 retrieved items judged for relevance). Base run for second iteration feedback.	.1885	-	-
Second Iteration Feedback based on 10 items retrieved by first iteration feedback			
DNF method, $Q = Q_1$ qcount = 2, T = 100	.2149	+14.0%	(1.0, 3.2, 2.4, 6.5)
DNF method, $Q = Q_0 \vee Q_1$ qcount = 2, T = 100	.2136	+13.3%	(includes original query)
Dillon feedback floor 1 = .9, floor 2 = .5	.1132	-39.9%	(4.9, 229.2, -, 234.2)
Dillon feedback floor 1 = .8, floor 2 = .6	.1166	-38.1%	(6.4, 55.4, -, 61.8)

Comparison of Dillon and DNF Feedback Methods
Two Feedback Iterations
(Inspec 12684 collection, 54 queries)

Table 7

```
for (each query)                                                  |   Comments:
    {
1   obtain N', R', N'-R'
2   if (R' = 0) go to next query
3   for (each term t in a document in {N'})
        {
4       obtain r_t', n_t'-r_t', freq_t = n_t                      | /* note that n'_t ≤ freq_t */
5       if n_t' = freq_t {set prev_t = -∞;                        | /* this implies that all
                          go to next term}                            items with term t were
                                                                      already retrieved */
6       set prev_t = (r'_t / min(R',freq_t) - (n'_t - r'_t) / min(N'-R',freq_t)) / ln freq_t   | /* prev_t can be
                                                                      positive or negative */
        }
7   set AVE = Σ_{prev_t≥0} prev_t / # of prev_t≥0                 | /* AVE is the mean of
                                                                      the prev_t values
                                                                      that are positive */
8   set STDDEV = Σ_{prev_t≥0} (prev_t-AVE)^2 / # of prev_t≥0 -1   | /* STDDEV is the standard
                                                                      deviation of prev_t values
                                                                      that are positive */
9   for (each term such that prev_t≥0)
        {
10      norm prev_t = (prev_t-AVE) / STDDEV .25 + .5              | /* normalization is
                                                                      designed to make
                                                                      same threshold
                                                                      values apply to
                                                                      different queries */
11      if (norm prev_t ≥ floor 1) put term t on singles list
12      if { (floor 1 > norm prev_t ≥ floor 2)
                put term t on pairs list
        }
13  if (the number of terms on pairs list is > 50)
    keep only the 50 terms with highest norm prev on list
14  for (every two terms A,B on pairs list)
    form a pair (A and B)
15  form feedback query by or-ing together all terms on
    singles list and all formed anded pairs
    }
```

Outline of Dillon Algorithm

Fig. 1

```
for (each query)
    {
1   for (each term t in a document in R' or in query Q)   | /* find and mark the
                                                             best single terms */
        {
2       obtain r'_t, freq_t
3       freqq_t = { freq_t + qcount   if t in query        | /* compute adjusted
                  { freq_t            if t not in query       term occurrence count */
4       rq'_t   = { r'_t              if t in query        | /* computed adjusted
                  { r'_t + qcount     if t not in query       occurrence count in
                                                              relevant items retrieved */
5       relwt_t = [ rq'_t/(R' + qcount) - freq_t/N ] ln [ N/(freqq_t + 10) ]   | /* compute term
                                                                                   relevance weight */
6       if relwt_t > 0.05 and
           relwt_t ≥ [1/(R' + qcount)] - 0.2
           then keep term t in heap
        }
7   mark the best 70 terms in heap (or all terms if fewer
    than 70 terms are available) as "good" single terms
8   for (each "good" single term s that appears            | /* obtain list of
        before term t on list of "good" terms)                 good term pairs */
        {
9       freq q_st = max [ (freq q_s · freq q_t)/N ,  number of items in R' with terms
                                                     t and s plus qcount if t and s
                                                     appear in query Q ]
10      rq'_st = number of relevant retrieved items
                 with terms t and s plus qcount
                 if t and s appear in query
        compute relwt_st as above
        }
11  mark the best 70 (or all) pairs as "good" pairs
12  for (each "good" pair t,s and each good single         | /* obtain list of
        term r not included in the pair)                       good term triples */
        {
13      obtain freq q_rst and rq'_rst
        and compute relwt_rst
        }
14  mark the best 70 (or all) triples as "good" triples
15  goto query generation process (Fig. 3)
    }
```

Generation of Good Single Terms, Term Pairs and Term Triples in DNF Process

```
for (each query)
    {
16      add all "good" single terms to heap                  | /* compute estimated number
                                                                 of items retrieved by
17      estret = Σ_{t ∈ good terms} freq_t                       disjunction of "good"
                                                                 single terms */
18      while (estret > T)
                                                             | /* T is maximum number of
          {                                                      desired retrieved items */
19        remove item with smallest relwt from heap
20        if (this item is a single term t)                  | /* remove single terms one
                                                                 at a time worst terms
            estret = estret - freq_t                             first */
21          if (estret < T/2)
              {put t back in heap; exit}
            mark t as "discarded"
22          add all "good" pairs t,s where s is                | /* add term pairs not
              another "discarded" single                         subsumed by singles
                                                                 still present */
23          estret = estret + Σ_{pairs added} freq pr_{t,s}

24        else if (this item is a term pair t,s)             | /* repeat the process for
                                                                 term pairs */
            estret = estret - freq pr_{t,s}
25          if (estret < T/2)
                {put pair back in heap; exit}
            mark pair as "discarded"
26          add all "good" triples t,s,r where r is a
              discarded single and triple is not already
              covered by another pair
27          estret = estret + Σ_{triples added} freq trp_{t,s,r}

28        else if (this item is a triple, t,s,r)             | /* repeat the process for
                                                                 term triples */
            estret = estret - freq trp_{t,s,r}
29          if (estret < T/2)
                {put triple back in heap; exit}
          }
30      form query by or-ing together all items left in heap;
        optionally add original query by or-ing

    }
```

Generation of Feedback Query in DNF Process

Fig. 3