Aspects of the P-Norm Model of Information Retrieval: Syntactic Query Generation, Efficiency, and Theoretical Properties

Maria E. Smith
Ph.D Thesis

TR 90-1128
May 1990

Department of Computer Science
Cornell University
Ithaca, NY 14853-7501

ASPECTS OF THE P-NORM MODEL OF INFORMATION RETRIEVAL: SYNTACTIC QUERY GENERATION, EFFICIENCY, AND THEORETICAL PROPERTIES

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Maria Elena Smith

May 1990

ASPECTS OF THE P-NORM MODEL OF INFORMATION RETRIEVAL: SYNTACTIC QUERY GENERATION, EFFICIENCY, AND THEORETICAL PROPERTIES

Maria Elena Smith, Ph.D.

Cornell University 1990

A practical information retrieval system must be easy to use by untrained users, and it must provide prompt responses to a user's search requests. In this thesis, these practical aspects of the p-norm model of information retrieval are explored. In addition, a study of theoretical properties of the p-norm model is presented.

A syntactic method for generating p-norm queries from parse trees generated by the PLNLP syntactic analyzer is presented. The effectiveness of the syntactically generated queries is shown to be comparable to the effectiveness of manually constructed queries, and much better than that of statistically generated queries.

The efficiency of a p-norm retrieval is significantly improved with a new p-norm retrieval algorithm which evaluates the entire document collection in one recursive traversal of the query tree. This algorithm is compared against the straightforward algorithm, which requires a traversal of the query tree for each document that is evaluated. The new algorithm is shown to be better both asymptotically and experimentally.

The infinity-one model is introduced as a means of approximating the p-norm model without requiring exponentiation. Experimental results show that infinity-one retrieval is essentially as effective as p-norm retrieval, but much faster. List pruning methods for further efficiency improvements are also introduced and are shown to reduce retrieval time significantly without affecting the precision of top-ranked documents. The retrieval time of the infinity-one model with list pruning is shown to be comparable to that of pure Boolean retrieval.

A theoretical study is also presented in which certain Boolean algebra properties, such as associativity, are shown to be unsatisfiable by any extended Boolean system with weak operators. The p-norm model is shown to satisfy all those properties that can be satisfied. In addition, the p-norm model is evaluated with respect to the Waller-Kraft wish list for extended Boolean systems.

Biographical Sketch

Maria Elena Smith was born in Havana, Cuba on August 11, 1960. From Cuba, she moved with her family to Spain, and then to the United States in December 1970. She completed her high school education at Miami Coral Park Senior High School. She graduated *summa cum laude* with highest honors in both Computer Science and Mathematics from Brandeis University, from which she received her Bachelor of Arts in 1982. She received a fellowship from Bell Laboratories for her graduate studies at Cornell University. In 1985 she earned the Master of Science degree in Computer Science from Cornell University. She is a member of the Association for Computing Machinery, and Phi Beta Kappa. She has a wonderful husband and a charming one and a half year old son.

To Mami

Acknowledgements

I would like to thank my advisor, Gerard Salton, for guiding me into the area of information retrieval and for introducing me to the p-norm model. His enthusiasm for the field was an inspiration to me. I greatly benefitted from Professor Salton's expertise and from the presentations and discussions in our weekly information retrieval seminars. I would also like to thank the other members of the SMART group, especially Chris Buckley, José Araya, and Joel Fagan. Special thanks to Joel for teaching me about the PLNLP system, and for many encouraging and supportive conversations. Thanks also to Charles Van Loan and Lionel Weiss for serving on my committee.

I gratefully acknowledge the support that Bell Laboratories has given me for my graduate studies. I would like to thank all of those involved, especially my mentor James McKenna.

I wish to express my deepest thanks to my husband, Geoff, for all the support that he has given me. Without his encouragement, confidence, willingness to listen to my ideas, patience, and assistance in revising my drafts, the completion of this thesis would not have been possible. I am very grateful for all the sacrifices that he has made for me during this process.

I would like to also thank Danielito, my son, for the joy that he has brought

to my life. This happiness that he has brought me made the stress-filled times involved in the completion of this thesis much more bearable.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

The area of *information retrieval* is concerned with developing techniques that facilitate the search for information. Clearly, a large collection of data is all but useless without some reasonable means of finding items of interest. Hence, the ability to store ever larger collections of data in computer systems demands better and better information retrieval techniques.

If the data in a collection is very structured, the retrieval problem is made easier. For example, if the collection consists of a set of records having various fields (such as name, address, and social security number), then most queries can be handled effectively merely by providing indices that allow one to efficiently locate all records having a given value in a certain field. This situation is typical of database systems.

In an *information retrieval system*, in contrast, the data is unstructured, generally consisting of natural language text. A collection is a set of *documents*, which can be books, articles, chapters, abstracts, electronic messages, or other units of text. A *query* is some description of the content of desired documents. The goal of an information retrieval system is to identify the documents in the collection whose content match the query.

Throughout history, large collections of text have been stored in libraries, and

it is not surprising that this is where the first tools of information retrieval, such as the Dewey decimal system, and the card catalog were developed. With the advent of computers more sophisticated retrieval methods have become possible.

An ideal information retrieval system would actually understand the entire content of each document and query. As this is infeasible, actual systems use some structured means of approximating the content of documents and of queries.

Documents are often represented by sets of content identifiers known as *index terms* or *keywords*. Each of these terms is assumed to describe to some degree the content of the document. The process of assigning these keywords to the individual documents is known as *indexing*.

Queries may also be represented by sets of index terms. However, most commercial retrieval systems are based on the *Boolean model*, in which queries are represented as Boolean expressions, i.e. as formulas consisting of index terms joined by the logical operators **AND**, **OR**, and **NOT**. The Boolean operators are typically used as follows:

- The **AND** is often used to combine terms into phrases. For example, INFORMATION **AND** RETRIEVAL. This operator indicates that only documents indexed by both terms are to be retrieved.

- The **OR** is often used to join synonyms, or any set of terms for which the presence of only one is sufficient for a document to be retrieved. For example, KEYWORD **OR** INDEX-TERM.

- The **NOT** operator is normally used together with an **AND** for term-narrowing purposes. For example, INDEXING **AND NOT** MANUAL. The set of documents retrieved is restricted to those which deal with indexing but not manual indexing.

The retrieval model that is studied in this thesis is the *p-norm model*, also sometimes referred to as the *extended Boolean model* [Wu81,SFW83,Fox83]. Like

the Boolean model, it makes use of the logical operators **AND**, **OR**, and **NOT**, but in a weakened form. The p-norm model overcomes many of the drawbacks of the Boolean model as shall be seen. However, it has some drawbacks of its own. The goal of this thesis is to address these problems in an effort to make the p-norm model practical.

1.1 The Boolean Model

1.1.1 Description

The Boolean model represents documents as sets of index terms. These terms may come from the text of the document itself or from some controlled vocabulary, and they may be assigned automatically or by a trained expert. The index terms used to represent a document are all treated as being equally descriptive of the document's content.

Most commercial systems use an inverted index for accessing the document collection. The *inverted index* [HH70] contains for each index term the list of document identifiers of all the documents indexed by the term. This accessing method makes the retrieval process very simple and efficient.

Boolean formulas are used to represent the queries. The Boolean operators **AND**, **OR**, and **NOT** are implemented as set intersection, set union, and set difference as follows [SM83]:

1. **AND** is used to ensure that all retrieved documents are indexed by all the terms in the **AND** clause. Therefore, given query A **AND** B, the documents retrieved are those in the intersection of the inverted list of term A and the inverted list of term B.

2. An **OR** operator requires that all retrieved documents be indexed by at least one term from the OR clause. Therefore, given query A **OR** B, the documents retrieved are those in the union of the inverted list of term A

and the inverted list of term B.

3. NOT is used to ensure that a certain term does not appear in a retrieved document. Therefore, given query A AND NOT B, the documents retrieved are those in the inverted list of term A, but not in the inverted list of term B.

The retrieval of a more complex query is done by repeating the above process recursively. For example, given query

(MANUAL OR AUTOMATIC) AND (DOCUMENT AND INDEXING)

the following procedure is used:

- Let S_1 = list for MANUAL $\bigcup$ list for AUTOMATIC
- Let S_2 = list for DOCUMENT $\bigcap$ list for INDEXING
- Return $S_1 \bigcap S_2$

Through these simple set manipulation procedures, the conventional Boolean retrieval systems can provide quick responses to user queries.

1.1.2 Limitations

Conventional Boolean logic exhibits many limitations which generally lead to poor effectiveness in a retrieval environment. These limitations are well known [SV85,Lee88]:

- The Boolean AND is very strict. For example, consider the query

$$t_1 \text{ AND } t_2 \text{ AND } \cdots \text{ AND } t_n.$$

The Boolean model considers a document indexed by t_1 through t_{n-1} but not term t_n, to be as bad as a document that is not indexed by any of these terms. This leads to an excessively narrow interpretation of the AND clause.

- The Boolean **OR** has a similar problem. Given query

$$t_1 \textbf{ OR } t_2 \textbf{ OR } \cdots \textbf{ OR } t_n,$$

the Boolean model considers a document indexed by all of these terms to be as good as a document indexed by only one.

- The output produced is not ranked. Any given document is considered to either satisfy or not to satisfy a query, i.e. there is no notion of partial satisfaction, so all documents retrieved are treated as being equally useful to the user. If the amount of output is large, the searcher would be unable to first consider those documents which are most likely to be relevant.

- There is no notion of relative importance among the terms assigned to documents and queries. This is a very serious limitation, and it is overcome in many other retrieval models by associating a value (or *weight*) with each term, typically in the interval [0,1], representing the importance of the term as a content representative.

1.2 The Pnorm Model

1.2.1 Definition

The p-norm model was first introduced in [Wu81]. It overcomes the drawbacks of the Boolean model by representing documents as points in an n-dimensional space, where n is the number of index terms in the collection, and by introducing the notion of *p-norm distances*[1] to evaluate **AND** and **OR** operators. A similarity

[1]The p-norm of a vector $\vec{X} = (x_1, \ldots, x_n)$ is defined as:

$$\|\vec{X}\|_p = \sqrt[p]{|x_i^p| + \cdots + |x_n^p|}$$

The p-norm distance between vectors $\vec{X}$ and $\vec{Y}$ is defined as:

$$\|\vec{X} - \vec{Y}\|_p$$

value between the query and the individual documents is computed using p-norms and used to rank the output in decreasing order of similarity. Furthermore, query terms and clauses can be weighted to specify their relative importance.

Document term weights are restricted to the interval [0,1]. Each document is mapped to a point in $[0,1]^n$, where each axis of the $[0,1]^n$-space corresponds to an index term. A document D is represented by point $(w_1, w_2, \cdots, w_n)$ where w_i is a measure of the importance of term t_i in describing the content of document D. If t_i is not used to index D, then $w_i = 0$.

To simplify the initial description of **AND** and **OR** evaluations in this model, all query terms and clauses are assumed to have weight 1.

Let us first see how the conventional Boolean **AND** and **OR** are interpreted when documents are visualized as points in an n-dimensional space. Since in the Boolean model all weights are binary, all documents are represented by the set of points corresponding to the corners of the n-dimensional cube. For example, when $n = 2$, the only points representing documents are $(0,0), (0,1), (1,0)$, and $(1,1)$. The Boolean query A **OR** B is satisfied by all points except $(0,0)$, and the query A **AND** B is satisfied only by point $(1,1)$.

In the p-norm model, the documents are represented by points *within* the n-dimensional cube, and the evaluation of the **AND** and **OR** operators is based on the distances from the points $(1,1,\cdots,1)$ and $(0,0,\cdots,0)$, respectively. Since only the absence of all terms, i.e. point $(0,0,\cdots,0)$, fails to satisfy a Boolean **OR** clause, the p-norm model **OR** ranks the documents in order of decreasing distance from $(0,0,\cdots,0)$. Since the Boolean **AND** is satisfied only by $(1,1,\cdots,1)$, the p-norm **AND** ranks the documents in order of increasing distance from the point $(1,1,\cdots,1)$.

The evaluation of the **AND** clause:

$$t_1 \textbf{ AND } t_2 \textbf{ AND } \ldots \textbf{ AND } t_m$$

consists of:

1. projecting the document collection from the n-dimensional space to the m-dimensional space

2. computing the distance from each document point in this m-dimensional space to the point $(1, 1, \cdots, 1)$

3. ranking the documents in order of increasing distance.

The evaluation of the **OR** clause:

$$t_1 \textbf{ OR } t_2 \textbf{ OR } \ldots \textbf{ OR } t_m$$

consists of:

1. projecting the document collection from the n-dimensional space to the m-dimensional space

2. computing the distance from each document point in this m-dimensional space to the point $(0, 0, \cdots, 0)$.

3. ranking the documents in order of decreasing distance.

The distance measures used by this model come from the L_p family of norms for $p = 1$ to ∞. The Boolean operators are assigned a special parameter, called the p-value, which indicates which L_p norm is to be used. The different p-values lead to varying degrees of conjunctivity and disjunctivity, where conjunctivity refers to the notion that a retrieved document needs to satisfy all of the operands of the operator, while disjunctivity refers to the notion that a retrieved document may only satisfy one.

All distances are normalized to ensure that all similarity values lie in the interval [0,1]. The normalization factor is $L_p dist((0, 0, \cdots, 0), (1, 1, \cdots, 1))$, the maximum L_p distance between any two points in the n-dimensional cube.

The formulas for the similarity computations where the query weights are all 1, can be summarized as follows:

OR Given query

$$Q = [t_1 \ \mathbf{OR}^p \ t_2 \ \mathbf{OR}^p \cdots \mathbf{OR}^p \ t_n]$$

and document

$$D = (d_1, d_2, \ldots, d_n) \text{ where } d_i\text{=weight of } t_i \text{ in } D,$$

the similarity is defined as:

$$\begin{aligned} \text{Sim}(D,Q) &= \frac{L_p dist((0,0,\ldots,0),(d_1,d_2,\ldots,d_n))}{L_p dist((0,0,\ldots,0),(1,1,\ldots,1))} \\ &= \sqrt[p]{\frac{d_1^p + d_2^p + \cdots + d_n^p}{n}} \end{aligned}$$

AND Given query

$$Q = [t_1 \ \mathbf{AND}^p \ t_2 \ \mathbf{AND}^p \cdots \mathbf{AND}^p \ t_n]$$

and document

$$D = (d_1, d_2, \ldots, d_n) \text{ where } d_i\text{=weight of } t_i \text{ in } D,$$

the similarity is defined as:

$$\begin{aligned} \text{Sim}(D,Q) &= 1 - \frac{L_p dist((1,1,\ldots,1),(d_1,d_2,\ldots,d_n))}{L_p dist((0,0,\ldots,0),(1,1,\ldots,1))} \\ &= 1 - \sqrt[p]{\frac{(1-d_1)^p + (1-d_2)^p + \cdots + (1-d_n)^p}{n}} \end{aligned}$$

NOT Given query $Q = \mathbf{NOT}\ t$ and document $D = (d_t)$, where d_t = weight of t in D, the similarity is defined as:

$$\text{Sim}(D,Q) = 1 - d_t$$

These rules can be applied recursively to evaluate more complex queries. For example, given query

$$Q = [t_1 \ \mathbf{AND}^{p1} \ (t_2 \ \mathbf{OR}^{p2} \ t_3)]$$

and document

$$D = (d_1, d_2, d_3)$$

the similarity is calculated as follows:

$$Sim(D,Q) = 1 - \sqrt[p1]{\frac{(1-d_1)^{p1} + (1 - \sqrt[p2]{\frac{d_2^{p2}+d_3^{p2}}{2}})^{p1}}{2}}$$

The behavior of the $\mathbf{AND}^p$ and $\mathbf{OR}^p$ functions with unweighted query terms is demonstrated by the level curves in Figure 1.1 and Figure 1.2 [Fox83,SFW83]. Any two points lie on the same level curve if and only if they have the same similarity value. In the **OR** case, the farther the level curve is from (0,0,...,0), the higher its corresponding similarity value. Analogously in the **AND** case, the closer the level curve is to the point (1,1,...,1), the higher its corresponding similarity value. It can be seen that when $p = 1$, no curvature is present in the level curves,and as p increases, the curvature also increases. At $p = 2$, the level curves are circular, and at $p = 5$, they have become somewhat rectangular. The level curves of $p = 8$ or higher are not very different from those at $p = \infty$, where the level curves *are* rectangular.

The effect of query weights on the similarity computation can be most easily visualized as a reshaping of the document space. Given query

$$\text{Q} = (t_1, q_1) \ \mathbf{OR} \ (t_2, q_2), \text{ where } q_i\text{=weight of term } t_i,$$

the document space $[0,1] \times [0,1]$ is compressed to the space $[0, q_1] \times [0, q_2]$, by mapping point (d_1, d_2) to the point $(q_1 d_1, q_2 d_2)$ (See Figure 1.3). All distances that were measured in the binary case from the point (1,1) are now measured from the point (q_1, q_2). So, the generalized formulas become:

OR Given query

$$Q = [(t_1, q_1) \ \mathbf{OR}^p \ (t_2, q_2) \ \mathbf{OR}^p \cdots \mathbf{OR}^p \ (t_n, q_n)]$$

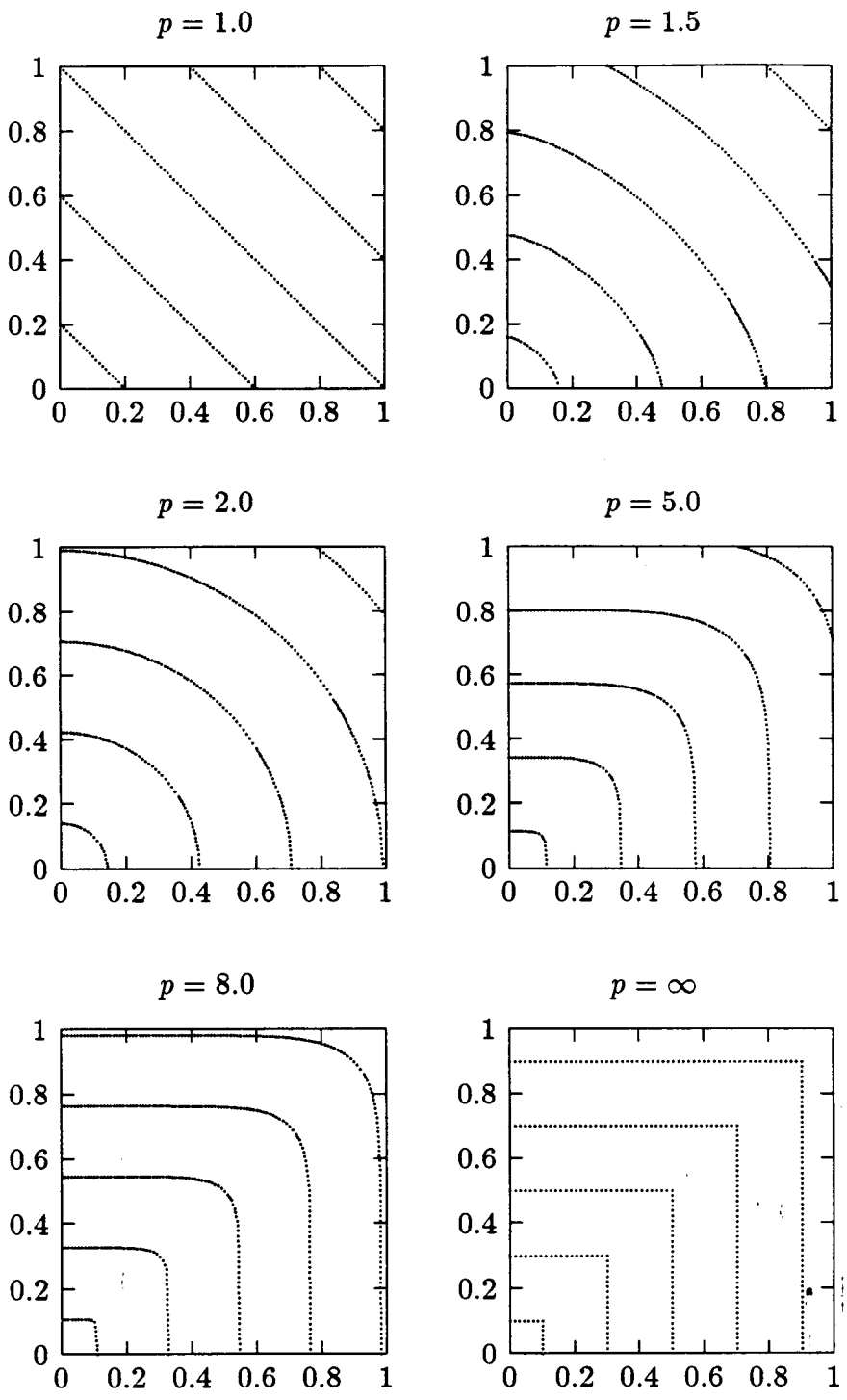

Figure 1.1: **OR** Level Curves

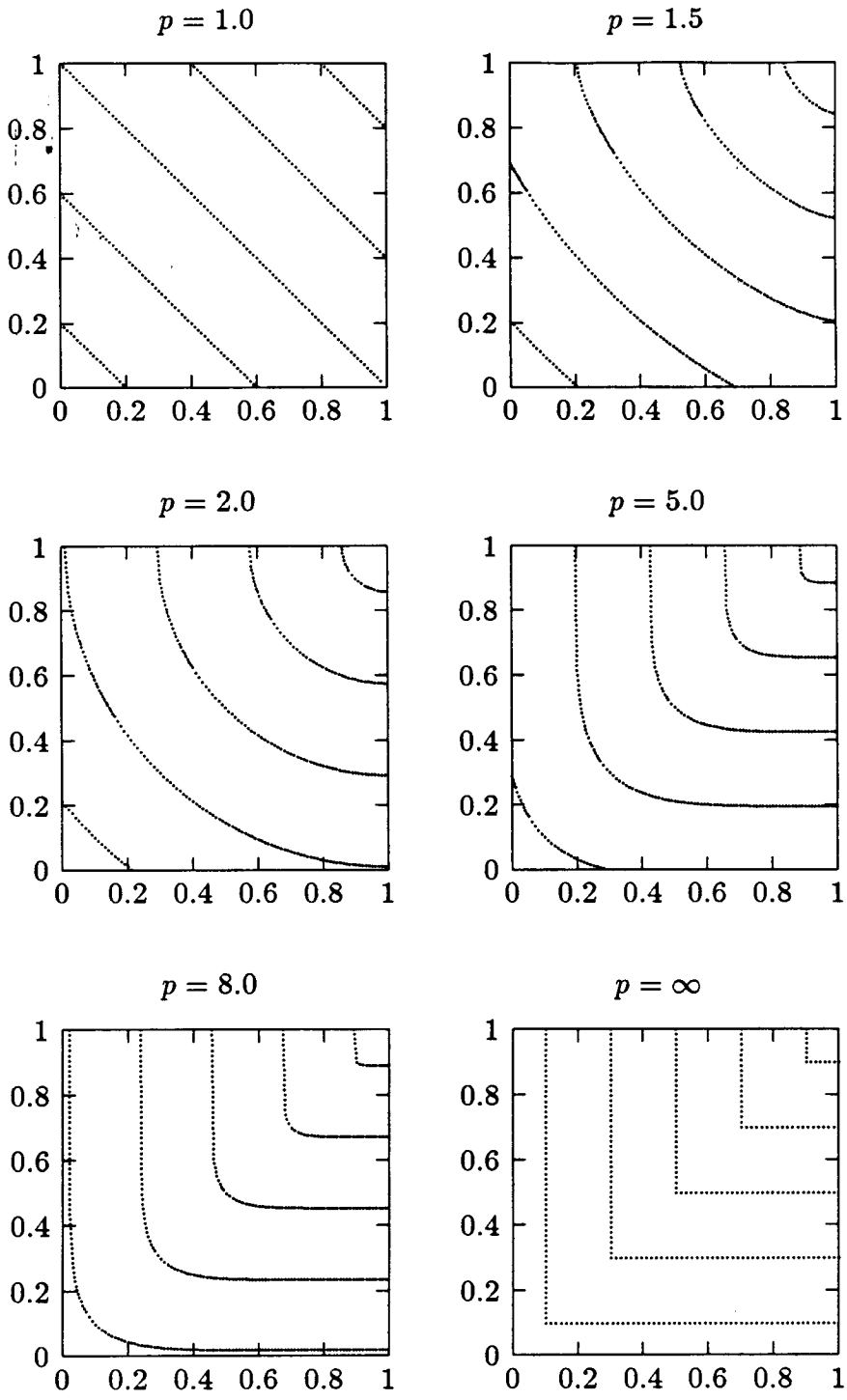

Figure 1.2: **AND** Level Curves

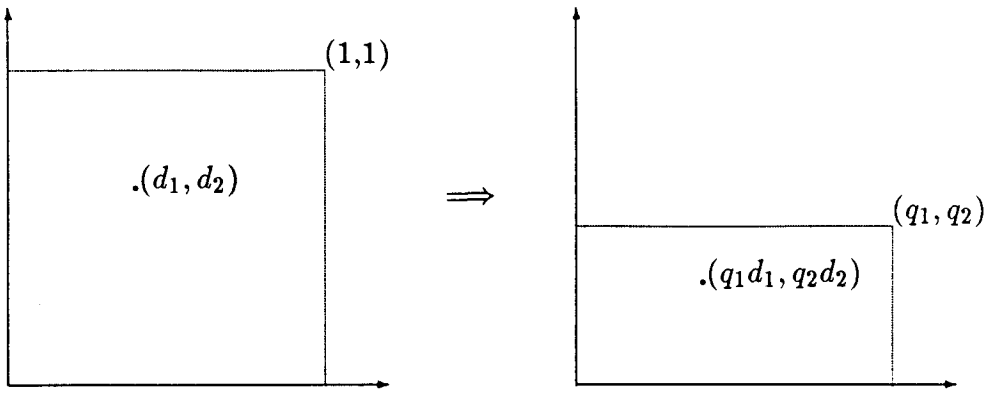

Figure 1.3: Compressing the Document Space

and document

$$D = (d_1, d_2, \ldots, d_n) \text{ where } d_i = \text{weight of } t_i \text{ in } D.$$

Point D gets mapped into $(q_1 d_1, q_2 d_2, \ldots, q_n d_n)$, and the similarity is defined as:

$$\begin{aligned} \text{Sim}(D, Q) &= \frac{L_p dist((0, 0, \ldots, 0), (d_1 q_1, d_2 q_2, \ldots, d_n q_n))}{L_p dist((0, 0, \ldots, 0), (q_1, q_2, \ldots, q_n))} \\ &= \sqrt[p]{\frac{q_1^p d_1^p + q_2^p d_2^p + \cdots + q_n^p d_n^p}{q_1^p + q_2^p + \cdots + q_n^p}} \end{aligned}$$

AND Given query

$$Q = [(t_1, q_1) \ \mathbf{AND}^p \ (t_2, q_2) \ \mathbf{AND}^p \cdots \mathbf{AND}^p \ (t_n, q_n)]$$

and document

$$D = (d_1, d_2, \ldots, d_n) \text{ where } d_i = \text{weight of } t_i \text{ in } D.$$

Point D gets mapped into $(q_1 d_1, q_2 d_2, \ldots, q_n d_n)$, and the similarity is defined as:

$$\text{Sim}(D, Q) = 1 - \frac{L_p dist((q_1, q_2, \ldots, q_n), (q_1 d_1, q_2 d_2, \ldots, q_n d_n))}{L_p dist((0, 0, \ldots, 0), (q_1, q_2, \ldots, q_n))}$$

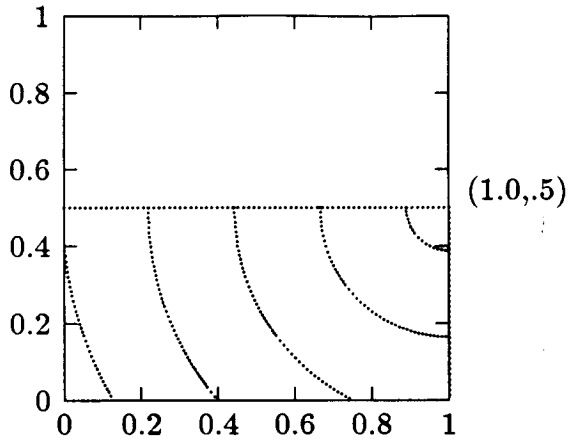

Figure 1.4: Level Curves for (A,1.0) $\mathbf{AND}^{2.0}$ (B,0.5) in Compressed Space

$$= 1 - \sqrt[p]{\frac{(q_1 - q_1 d_1)^p + (q_2 - q_2 d_2)^p + \cdots + (q_n - q_n d_n)^p}{q_1^p + q_2^p + \cdots + q_n^p}}$$

$$= 1 - \sqrt[p]{\frac{q_1^p(1 - d_1)^p + q_2^p(1 - d_2)^p + \cdots + q_n^p(1 - d_n)^p}{q_1^p + q_2^p + \cdots + q_n^p}}$$

The level curves in the compressed space are as before, but in the $\mathbf{AND}^p$ case they are centered around $(q_1, q_2, \ldots, q_n)$ instead of (1,1,...,1). See example in Figure 1.4. Another way of interpreting the effect of query weights was demonstrated in [Fox83] as non-symmetrical level curves. Figure 1.5 shows what the level curves for the same query as in Figure 1.4 look like under this interpretation. In summary, Figure 1.4 views the effect of query weights as a reshaping of the document space, while Figure 1.5 views it as a reshaping of the level curves.

1.2.2 Properties

The level curves in Figure 1.1 and Figure 1.2 demonstrate the effect of the various p-values on the similarity computation. At one extreme, $p = 1$, there is a very softened interpretation of the operator (no conjunctivity or disjunctivity), and at the other extreme $p = \infty$, there is a very strict interpretation of the operator.

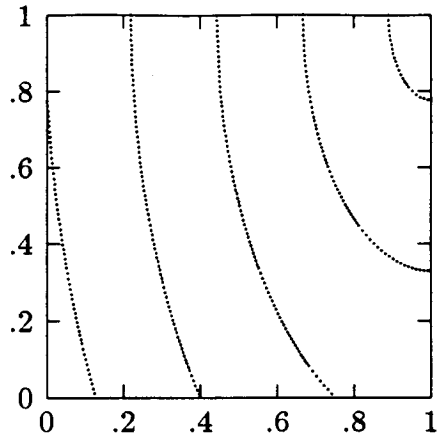

Figure 1.5: Level Curves for (A,1.0) $\mathbf{AND}^{2.0}$ (B,0.5) in Uncompressed Space

These properties were observed in [SFW83]:

1. $\text{Sim}(t_1\ \mathbf{AND}^{\infty}\ t_2\ \mathbf{AND}^{\infty} \cdots \mathbf{AND}^{\infty}\ t_n, D) = \min(d_1, d_2, \ldots, d_n)$ and
   $\text{Sim}(t_1\ \mathbf{OR}^{\infty}\ t_2\ \mathbf{OR}^{\infty} \cdots \mathbf{OR}^{\infty}\ t_n, D) = \max(d_1, d_2, \ldots, d_n)$
   where $D = (d_1, d_2, \ldots, d_n)$ and all query weights are 1.0

2. $\mathbf{AND}^1 = \mathbf{OR}^1$

Property 1 states that the p-norm model is an extension of the fuzzy-set model to be discussed in Section 1.3.1. Furthermore, the p-norm model restricted to binary weights and $p = \infty$ is equivalent to the pure Boolean model.

Property 2 states that when $p = 1$, there is no difference between **AND** and **OR**. The evaluation of a clause with $p = 1$ is simply the weighted average of the document term weights, and this can be viewed as a form of vector processing. See Section 1.4 for a discussion of the vector model.

Another observation that was made in [SFW83] is that many properties of *Boolean Algebras* are not satisfied by the p-norm model. The notion that logically equivalent queries should yield identical results for any given document is known as *Boolean self-consistency*, and it is listed as one of the desirable properties of any generalized Boolean information system in the well-known Waller-Kraft wish

list [WK79]. A study of the pnorm model with respect to this list is found in Chapter 6.

In summary, the p-norm model exhibits the following properties:

- Allows weighting of terms in documents and queries.
- Allows various levels of conjunctivity and disjunctivity.
- Produces ranked output.
- Includes the Boolean and vector models as special cases.
- Does not satisfy the Boolean self-consistency criteria.

1.3 Other Generalized Boolean Systems

1.3.1 Fuzzy Set Model

A *fuzzy set* [Zad65] is a class of objects characterized by a *membership function* which expresses the degree to which an object is a member of the set. This function maps an object to 1 to indicate that it is a full member of the set, it maps an object to 0 to indicate that it is not a member, and it maps an object to intermediate values to indicate its partial degree of membership. This generalized notion of sets can be useful when dealing with imprecisely defined classes of objects and its applicability has been studied for areas such as pattern recognition, information retrieval, and database systems [Tah77,Usz86].

Numerous papers have dealt with the use of fuzzy set theory in information retrieval such as [Tah76,WK79,Rad79,BK81b], where various information retrieval models based on fuzzy sets have been introduced and studied. Fuzzy sets are typically used to represent the set of documents indexed by a given term. The weight of a term t in a document is used as an indicator of the document's degree of membership in the set of documents indexed by t. The standard interpretation of the Boolean connectives in the fuzzy set model is:

Sim(A **OR** B, D) = $\max(d_A, d_B)$

Sim(A **AND** B, D) = $\min(d_A, d_B)$

Sim(**NOT** A, D) = 1 - d_A

These similarity computations are directly based on the definitions of union, intersection and complementation in fuzzy set theory[Zad65]. The appropriateness of this interpretation of Boolean operators in an information retrieval setting has been strongly questioned[Rob78].

The first two limitations of the Boolean model listed in Section 1.1.2 are still present in the fuzzy set model. For example, given query

$$t_1 \textbf{ AND } t_2 \textbf{ AND } \cdots \textbf{ AND } t_n.$$

and documents

$$D_1 = (1, \ldots, 1, 0)$$
$$D_2 = (0, 0, \ldots, 0),$$

the similarities are

$$\text{Sim(Q,}D_1\text{)} = \min(1,\ldots,1,0) = 0$$
$$\text{Sim(Q,}D_2\text{)} = \min(0,0,\ldots,0) = 0$$

So, it can be seen that a document indexed by all but one of the terms is not considered any better than a document not indexed by any of the terms. Similarly, given query

$$t_1 \textbf{ OR } t_2 \textbf{ OR } \cdots \textbf{ OR } t_n.$$

and documents

$$D_1 = (1, 0, \ldots, 0)$$
$$D_2 = (1, 1, \ldots, 1),$$

the similarities are

$$\text{Sim}(Q,D_1) = \max(1,0,\ldots,0) = 1$$
$$\text{Sim}(Q,D_2) = \max(1,1,\ldots,1) = 1$$

Thus, a document indexed by all the terms is considered to be as good as a document indexed by only one.

Extensions that allow query terms and clauses to be weighted has also been considered [Rad79,Boo80]. However, the introduction of query weights has led to additional undesirable properties [BK81b,Bue81,Boo78].

1.3.2 Waller & Kraft's Model

In an attempt to satisfy as many of the properties in the Waller-Kraft wish list as possible, the following function was suggested in [WK79] for evaluating both ANDs and ORs:

$$\text{Sim(A } op \text{ B, D)} = z \min(d_A, d_B) + (1 - z)\max(d_A, d_B)$$

where z is a value in [0,1] and is specified by the user as part of the query. If the user wants to restrict the set of records retrieved to those which evaluate reasonably high for both A and B, then he would use a high z value to obtain the effect of an AND. On the other hand, if the user feels that a document which evaluates reasonably high on only one of these subexpressions is still very likely to be useful, he would use a small z.

It should be noted that a drawback of this model is that when more than two expressions are joined by a Boolean operator, only those expressions evaluating to the min and max have an effect on the similarity. For example, given query

$$Q = t_1 \; op \; t_2 \; op \cdots op \; t_n$$

and documents

$$D_1 = (.8,.1,0,\ldots,0)$$
$$D_2 = (.8\ .1,.7,\ldots,.7)$$

it can be seen that

$$\text{Sim}(Q,D_1) = \text{Sim}(Q,D_2).$$

Therefore, D_1 and D_2 are treated as being equally good in this model. However, intuitively, one would consider D_2 more likely to be relevant than D_1.

1.3.3 Paice's Model

The following variation of the Waller & Kraft's model was proposed in [Pai84]. ANDs and ORs are evaluated with the following function:

$$\text{Sim}(t_1\ op\ t_2\ op \cdots op\ t_n,\ \text{D}) = \frac{\sum_{i=1}^{n} r^{i-1} d_i}{\sum_{i=1}^{n} r^{i-1}}$$

where r is in [0,1] and the d_i's are in ascending order if op = **AND**, or in descending order if op = **OR**. When $r = 0$, this model reduces to the classical fuzzy set model, and when $r = 1$, it reduces to a vector processing system.

This model is not aesthetically appealing since it is not based on any theoretical foundation. For instance, there is no justification for giving each subsequent term weight a weighting factor which is a fixed ratio r of the preceding weighting factor.

1.3.4 TIRS

TIRS (Topological Information Retrieval System) is presented in [CK87] as a model satisfying the requirements of the Waller-Kraft wish list. In this system, queries are represented as finite sets of points in the document space and each of these points is considered to represent a "perfect" document. A document description is a point with a finite number of non-zero entries in the document space

$$DS = \prod_{i=1}^{\infty} [0,1].$$

A query is processed as follows:

1. To obtain the set of "perfect" document descriptions, the Boolean query is transformed into DNF (disjunctive normal form) where there is one disjunct for each combination of atoms that makes the Boolean expression evaluate to true. For example, given query

   (A,a) OR (B,b)

   where a and b are the weights of A and B respectively, TIRS transforms it into:

   ((A,a) AND (B,b)) OR ((A,a) AND NOT(B,b))
   OR
   (NOT(A,a) AND (B,b))

2. All atoms of the form **NOT (A,wt)** are replaced by (A,1 - wt). So in the above example, one gets:

   ((A,a) AND (B,b)) OR ((A,a) AND (B,1-b)) OR ((A,1-a) AND (B,b))

3. Each conjunct is mapped into a point in DS. Thus, in this example the query is represented by the three "perfect" documents (a,b), (a,1-b), and (1-a,b).

4. These "perfect" points are located in the projection of DS onto $[0,1]^n$ where n is the number of terms in the query.

5. Using a metric, the documents closest to the query points are located. A weight w_i is associated with each query point representing the maximum allowable distance between this point and a retrieved document. In other words, given a query represented by the points $Q_1, Q_2, \ldots, Q_n$, the set of documents retrieved is given by

$$\{D|\exists i(i \in 1,\ldots,m) \text{ s.t. } dist(Q_i, D) \leq w_i\}$$

6. The documents are ranked in order of decreasing distance from the query points.

Among the drawbacks associated with this model is the fact that its computational requirements are too high for this model to be useful in practice. In the experiments presented in [Lee88], it was found that performing a retrieval with TIRS was often infeasible due to the large number of minterms (i.e. conjunctive clauses) generated in the transformation of the Boolean query into its disjunctive normal form. For example, in the CISI collection (35 queries), two queries had over 3000 minterms, and 30% had over 100 minterms. In the CACM collection (64 queries), one query generated 24,171 minterms.

1.4 Vector Model

The vector space model represents both documents and queries as vectors in an n-dimensional space, where n is the number of index terms in the collection. Each element of the vector is assigned a value in [0,1] to represent the importance of the associated term. Thus, given a collection with n index terms, a document D is represented in the form:

$$D = (d_1, d_2, \ldots, d_n)$$

where d_i indicates how good term i is in describing the content of document D. Queries are similarly represented.

Various methods for computing the similarity between two vectors have been described in the literature. Among the most commonly used measures is the *cosine coefficient* defined as follows:

$$\text{Sim(D,Q)} = \frac{\sum_{i=1}^{n} q_i d_i}{\sqrt{\sum_{i=1}^{n} q_i^2 \sum_{i=1}^{n} d_i^2}}$$

where

$$D = (d_1, d_2, \ldots, d_n) \text{ and}$$
$$Q = (q_1, q_2, \ldots, q_n).$$

The simplicity of this model, and its ability to handle document term weights and query term weights in a natural way make this model very attractive. However, the model has been criticized for assuming that the term vectors are orthogonal (i.e. terms are assumed to be independent) [WZW85], and for the absence of a theoretical justification for the vector similarity measures [Sal89].

This model produces much better retrieval output than the Boolean model, and has been used in the literature as the standard for comparison with newly proposed retrieval processes. All of the experimental data, will therefore, be compared with the vector model's retrieval output.

1.5 Retrieval Environment

The experiments were performed using the SMART information retrieval system [Buc85] extended by programs implementing the various p-norm retrieval algorithms presented in this thesis.

1.5.1 Experimental Collections

Three document collections were utilized in this study. Each collection includes the text-form of document titles and abstracts, a set of natural language queries, a set of Boolean queries manually constructed from the natural language statements, and a set of relevance judgements describing which documents are relevant to the various queries.

- The CACM collection contains the articles in issues of *Communications of the ACM* from 1958 to 1979. There are a total of 3204 documents and 52

natural language queries. Two graduate students at Cornell created the associated Boolean queries.

- The Medlars collection consists of 1033 documents selected from a medical collection at the National Library of Medicine. The 30 natural language queries associated with this collection were provided by the National Library of Medicine, and the corresponding Boolean queries are based on the Boolean expressions used by actual searchers but adapted to the information in document representations by expanding the queries through the use of the MESH thesaurus[Fox83].

- The INSPEC collections consists of 12,684 articles focusing mostly on electrical engineering and computer science. It contains 84 natural language queries submitted by Syracuse University students and faculty. The associated Boolean queries were built up from Diatom[2] searches. Since in a Diatom search various sets are retrieved until eventually the result of one is selected for printing, the Boolean query associated with a search combines the Boolean expressions representing each retrieval set into one Boolean expression. As a result, the Boolean queries in this collection tend to be very deep and long.

1.5.2 Evaluation

Retrieval effectiveness is typically measured in terms of *recall* and *precision*. Recall measures the proportion of relevant documents that are retrieved, and precision measures the proportion of retrieved documents that are relevant. In other words, recall is defined as:

$$R = \frac{\text{number of relevant retrieved}}{\text{number relevant}}$$

[2]Diatom is a Syracuse system simulating DIALOG (trademark of LMSC, Inc.)

and precision is defined as:

$$P = \frac{\text{number of relevant retrieved}}{\text{number retrieved}}$$

In retrieval systems in which ranking is produced, the precision obtained at various recall levels is often used. For example, the precision at a recall level of .5 measures the precision obtained when 50% of the relevant documents have been retrieved. In this study, the following two measurements will be used:

1. Average precision at recall levels of .25, .50, and .75
2. Precision obtained in the 10 top documents retrieved

1.6 Thesis Synopsis

The goal of this thesis is to make the p-norm model practical. The issues that are considered are:

1. automatic generation of p-norm queries from natural language search requests, and
2. efficiency of p-norm retrieval.

Chapter 2 presents a syntactic p-norm query generation algorithm using the PLNLP natural language processing system developed at IBM, and reviews past work in the area of automatic p-norm query generation.

Chapter 3 studies the effect of various parameter settings on the effectiveness of p-norm retrieval, and compares the automatically generated p-norm queries against manually constructed p-norm queries and automatically constructed vector queries.

Chapters 4 and 5 deal with the efficiency aspects of p-norm retrieval. A new p-norm retrieval algorithm is presented in Chapter 4, and approximations to the p-norm model are presented in Chapter 5.

Chapter 6 analyzes the Boolean Algebra properties with respect to any system containing softened boolean operators, and shows that the p-norm model satisfies all the properties that any such system can satisfy.

Chapter 2

Automatic Generation of P-norm Queries

Studies have shown that it is very difficult for untrained users to express their search requests in terms of Boolean logic [BVW72]. Even though the p-norm model is more forgiving than the pure Boolean model, it is useful to have a method of generating p-norm queries automatically from natural language search requests. Previous work in this area [SBF83] takes into consideration only the frequency characteristics of the terms in the user's request. In this study, a syntactic approach to the query generation problem is instead considered.

The objective of this chapter is:

1. to review the statistical query generation process presented in [SBF83], and
2. to introduce a syntactic query generation process.

These two methods will be compared in Chapter 3, where the output of retrieval runs using the queries generated by these methods is evaluated.

2.1 Statistical Query Generation

The method proposed in [SBF83] relies solely on the terms of the natural language search request and their related frequencies, without any regard for sentence structure. Co-occurrence information about the terms is the key factor used in this method to decide on how to combine the terms into a p-norm formula.

The first step of the statistical query generation algorithm is to construct a Boolean formula such that the estimated number of documents retrieved by the formula in a pure Boolean system is some specified number of desired documents. This Boolean formula is then transformed into a p-norm query by weighting each query term with an *idf* weight[1], weighting each query clause with the average weight of its operands, and assigning a common p-value to the Boolean operators. Low p-values of $p = 1$ or $p = 2$ were found to be best. Chapter 3 discusses these parameter settings more thoroughly.

The constructed Boolean formulas are in disjunctive normal form where the disjuncts can be single terms, *pairs* (i.e. two terms *anded* together) or *triples* (i.e. three terms *anded* together). A typical query looks as follows:

$$t_1 \textbf{ OR } t_2 \textbf{ OR } (t_3 \textbf{ AND } t_4) \textbf{ OR } (t_5 \textbf{ AND } t_6 \textbf{ AND } t_7) \textbf{ OR } \cdots$$

The **NOT** is not used in the generated queries. The estimated number of documents retrieved by such a query is taken to be the sum of the document frequencies[2] of the single terms, and the estimated document frequencies of the pairs and triples in the query[3]. By assuming that terms occur independently of each other, the document frequency for a pair (t_i **AND** t_j) is estimated by:

$$n_{ij} = \frac{n_i \cdot n_j}{N}$$

[1] *idf*, inverse document frequency, weights vary inversely with the number of documents indexed by the term, thereby giving preference to terms concentrated in only a few documents. See Chapter 3 for a more detailed discussion.

[2] The document frequency of a term is the number of documents indexed by the term.

[3] This is a very rough estimate since it assumes that there is no overlap among documents retrieved by the individual clauses.

and for a triple (t_i **AND** t_j **AND** t_k) by:

$$n_{ijk} = \frac{n_i \cdot n_j \cdot n_k}{N^2}$$

where N is the number of documents in the collection, and n_i, n_j, and n_k are the document frequencies of the terms t_i, t_j, and t_k, respectively.

The process for constructing the Boolean formula can be summarized as follows:

1. Take the terms from the natural language search request, use a stop list to remove common function words, and remove common suffixes from the remaining terms.

2. Eliminate terms with excessive document frequencies.

3. Let the initial query be the **OR** of a few single terms with highest idf weight, and all pairs which do not include any of these single terms.

4. If the estimated number of retrieved documents by the query is smaller than the desired number, the query formulation is broadened by adding single terms in idf weight order and removing the pairs subsumed by the added singles. For example, if term t_i is added as a single term to the query, then all pairs of the form (t_i **AND** t_j) become unnecessary, and are therefore removed.

5. If the estimated number of retrieved documents by the query is larger than the desired number, a narrower query formulation is constructed by eliminating single terms in increasing idf weight order and adding the missing term pairs. If, after all singles have been deleted, the expected number of retrieved documents is still too large, then term pairs are deleted in increasing idf weight order and triples consisting of terms from the deleted pairs are added.

Table 2.1: Term Frequencies and Weights

Term	Document Frequency	Idf Weight[a]
effect	248	.7602
excres (ex)	52	.9497
hormon (ho)	81	.9217
kidney (ki)	78	.9246
parathyr (pa)	27	.9739
phosp (ph)	43	.9584
urin (ur)	78	.9246

[a]The idf weight used here is: 1 - n/(N+1).

6. Steps 4 and 5 are repeated until the estimated number of retrieved documents is the desired number.

The following example from [SBF83] illustrates this query generation process.

Example: Statistical Generation Process for the search request:

> *Excretion of phosphate or pyrophosphate in the urine or the effect of parathyroid hormone in the kidney.*

given a desired number of retrieved documents of 20.

After the initial step of removing terms from the stop list, and performing a suffix removal, the list of terms in Table 2.1 is obtained. The term *effect* is eliminated because of its excessive document frequency, and only the remaining terms are used in the query generation. These terms will be referred to by the abbreviations appearing in parentheses in Table 2.1. The estimated frequencies and weights of pairs are given in Table 2.2, where these abbreviations are used. A similar table is constructed for triples.

The initial query is obtained by taking the **OR** of the two single terms with lowest frequency and all the pairs not involving these two terms. Thus, the initial query becomes:

Table 2.2: Pair Frequencies and Weights

Pair	Estimated Frequency	Weight
ho-ex	4.1	.9961
ki-ex	3.9	.9962
ki-ho	6.1	.9941
pa-ex	1.4	.9987
pa-ho	2.1	.9980
pa-hi	2.0	.9980
ph-ex	2.2	.9979
ph-ho	3.4	.9967
ph-ki	3.2	.9969
ph-pa	1.1	.9989
ur-ex	3.9	.9962
ur-ho	6.1	.9941
ur-ki	5.9	.9943
ur-pa	2.0	.9980
ur-ph	3.2	.9969

Pa OR Ph OR (Ki AND Ex) OR (Ur AND Ex) OR (Ho AND Ex) OR (Ur AND Ki) OR (Ur AND Ho) OR (Ki AND Ho).

The estimated number of documents retrieved by this query is 100. Therefore, the highest frequency single is replaced by all pairs with this term, giving:

Pa OR [(Ph AND Ex) OR (Ph AND Ho) OR (Ph AND Ki) OR (Ph AND Ur)] OR (Ki AND Ex) OR (Ur AND Ex) OR (Ho AND Ex) OR (Ur AND Ki) OR (Ur AND Ho) OR (Ki AND Ho).

The estimated frequency of the query is now 69. Since this is still considered too high, the next single has to be replaced by all pairs that involve it, giving:

[(Pa AND Ex) OR (Pa AND Ho) OR (Pa AND Ki) OR (Pa AND Ph) OR (Pa AND Ur)] OR (Ph AND Ex) OR (Ph AND Ho) OR (Ph AND Ki) OR (Ph AND Ur) OR (Ki AND Ex) OR (Ur AND Ex) OR (Ho AND Ex) OR (Ur AND Ki) OR (Ur AND Ho) OR (Ki AND Ho).

The estimated number of documents retrieved by this query is 50.6. Therefore, it needs to be made more narrow still. Since there are no single terms remaining, the highest frequency pair has to be removed. So, the pair (Ki **AND** Ho) is removed. With this done, the estimated frequency is 44.5. Since this is still too high, the next highest frequency pair, namely (Ur **AND** Ho), is removed. The estimated frequency of the query is now 38.4. When the next highest pair is removed, (Ur **AND** Ki), the nonredundant triple (Ur **AND** Ki **AND** Ho) is inserted into the query, giving us:

(Pa **AND** Ex) **OR** (Pa **AND** Ho) **OR** (Pa **AND** Ki) **OR**
(Pa **AND** Ph) **OR** (Pa **AND** Ur) **OR** (Ph **AND** Ex) **OR**
(Ph **AND** Ho) **OR** (Ph **AND** Ki) **OR** (Ph **AND** Ur) **OR**
(Ki **AND** Ex) **OR** (Ur **AND** Ex) **OR** (Ho **AND** Ex) **OR**
(Ur **AND** Ki **AND** Ho).

At this point, the estimated frequency is 33, and the process continues by removing pairs (Ho **AND** Ex) and (Ki **AND** Ex), and adding triple (Ki **AND** Ex **AND** Ho), and producing the final query with estimated frequency of 22, which is considered to be close enough to the desired number of documents.

The query produced in the above example does not reflect the meaning of the original search request in any obvious way. In the next section an approach that generates the p-norm query based on the syntactic structure of the search request is described.

2.2 Syntactic Query Generation

The goal of the syntactic query generation algorithm is to better represent the content of the search request by grouping the terms in a meaningful way. For example, a more appropriate p-norm query for the search request given in the previous section would be

(*excretion* **AND** (*phosphate* **OR** *pyrophosphate*) **AND** *urine*) **OR**
(*effect* **AND** (*parathyroid* **AND** *hormone*) **AND** *kidney*)

The structure given to the p-norm query is determined by the syntactic structure of the search request. For example, the proposed algorithm groups the terms making up a noun phrase or prepositional phrase into one **AND** clause, and it joins sets of enumerated concepts into an **OR** clause.

The method that is proposed makes use of the PLNLP natural language processing system developed at the IBM Research Laboratory in Yorktown Heights. The system provides:

1. a syntactic analyzer.

2. facilities for manipulating the output of the analyzer.

Given an English sentence, a parse tree is produced by the syntactic analyzer through the use of a general purpose English dictionary and a set of grammar rules for the English language included in the system. The query generation method builds a p-norm query from the parse trees obtained for each of the sentences in the natural language search request provided by the user.

2.2.1 Overview of the PLNLP System

PLNLP, Programming Language for Natural Language Processing, was used in the implementation of the syntactic analyzer and it is the programming language required by the parse tree manipulating facilities of the system. Since the query construction process depends on these system facilities, it was implemented in PLNLP.

A program in PLNLP is simply a set of *augmented phrase structure rules*[4] where the left-hand side of the rule identifies the type of object that the rule can

[4]See [Hei72,HJM+82,Win83] for more information on augmented phrase structure grammars.

DECL	NP	NP	NOUN*	"memory"	
		NP	NOUN*	"management"	
		NOUN*	"units"		
	VERB*	"regulate"			
	NP	NOUN*	"access"		
		PP	PREP	"to"	
			AJP	ADJ*	"dedicated"
			NP	NOUN*	"storage"
			NOUN*	"segments"	
	PUNC "."				

Figure 2.1: Example of a Parse Tree Produced by PLNLP

be applied to, and the right-hand side specifies the objects that the left-hand side of the rule gets replaced by. Each rule can be augmented by:

1. a set of conditions that must hold in order for the rule to be applied, and

2. a set of actions that are to be performed when the rule is applied.

A detailed description of the syntax and capabilities of this language can be found in [Hei72]. Due to the complexity and typical unreadability of the syntax of this language, the p-norm query generation method will be presented in higher level terms.

The parse tree produced by the analyzer for a sample sentence is found in Figure 2.1. This can be viewed as a tree placed on its side, where the root is in the first column, the children of the root are in the second column, and so forth. By turning the example in Figure 2.1 right-side up, Figure 2.2 is obtained.

In the example from Figure 2.1, the root is labeled by DECL, meaning that this is a declarative sentence. It is then subdivided into NP (noun phrase), VERB, and NP. The noun phrases are further subdivided and so on.

Each non-lexical construction, such as NP and PP (prepositional phrase), has a *head.* The head is identified by an asterisk in the parse tree. In the example from Figure 2.1, the first NP represents the noun phrase *memory management units*, and its head is *units*. The constituents preceding the head are known as

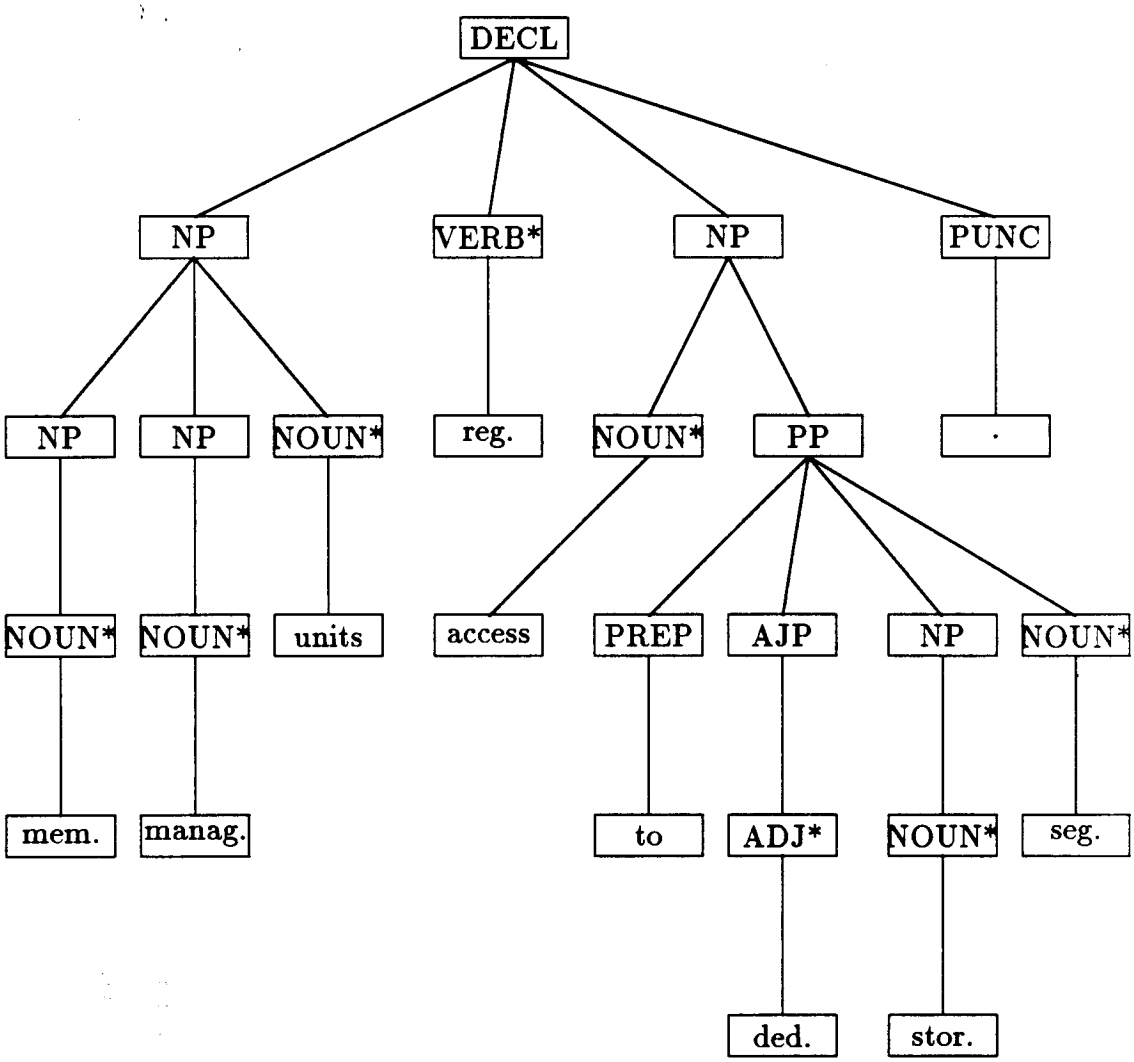

Figure 2.2: Output of Parser Viewed as a Tree

```
NP      NP      NOUN*   "code"
        NOUN*   "optimization"
        PP      PREP    "for"
                NP      NOUN*   "space"
                NOUN*   "efficiency"
        PUNC "."
```

Figure 2.3: Example of a Sentence Fragment Analysis

premodifiers and those following the head are referred to as postmodifiers. In the first NP, the head noun *units* has premodifiers *memory* and *management*, and it has no postmodifiers. In the second NP, the head noun is *access* and it only has one postmodifier, namely the prepositional phrase *to dedicated storage segments*.

The PLNLP system is very flexible with regard to the type of input that it is able to analyze.

1. PLNLP is not restricted to a particular subject domain.

2. PLNLP is able to analyze sentence fragments, as well as complete sentences. For example, the phrase *code optimization for space efficiency* is analyzed as a noun phrase as shown in Figure 2.3.

3. PLNLP can handle ambiguous sentences by producing multiple parses and applying a parse metric to rank them in decreasing order of assumed correctness.

4. When parsing fails, PLNLP produces a *fitted parse*, which identifies the lower level structures that were successfully parsed. PLNLP first attempts to parse the text as a sentence. If that fails, it tries to parse it as a verb phrase, and if that fails, it then tries to parse it as a noun phrase. If this also fails, PLNLP parses smaller portions of the text in whatever way it can. Figure 2.4 shows an example of a fitted parse produced by PLNLP. The root is labelled by XXXX to indicate that this is a fitted parse. PLNLP failed in its attempt to parse *shape descriptions, shape recognition by computer*,

XXXX	NP*	NP	NOUN*	"shape"
		NOUN*	"descriptions"	
		PUNC	","	
	NP	NP	NOUN*	"shape"
		NOUN*	"recognition"	
		PP	PREP	"by"
			NOUN*	"computer"
	PUNC "."			

Figure 2.4: Example of a Fitted Parse

and so it produced the parse trees for the two fragments, namely, *shape descriptions* and *shape recognition by computer.*

Because of its flexibility, PLNLP is well-suited to information retrieval applications requiring the syntactic analysis of natural language text. Some applications in the literature may be found in [Fag87,SS89].

2.2.2 Syntactic Query Generation Algorithm

The approach to p-norm query generation introduced here is based on a postorder tree traversal of the parse tree produced by the syntactic analyzer. Each node of the tree represents a syntactic category, such as: NP (noun phrase), PP (prepositional phrase), VERB, NOUN, ADJ (adjective), ADV (adverb), etc. A tree node contains pointers to its children, which represent how the syntactic structure associated with the tree node can be broken down into lower-level structures. In the example from Figure 2.1, the NP node can be seen to be associated with *memory management units* point to three lower-level structures, namely the NP node associated with *memory*, the NP node associated with *management*, and the NOUN node associated with *units*.

In general, at each tree node the algorithm does the following:

1. Recursively calls itself on the children (if any).

2. If the node is a non-leaf, determines how the p-norm formulas returned by

the children are to be combined, if at all, based on the syntactic category associated with the tree node. As will be seen, a node may return more than one p-norm query.

3. If the node is a leaf, determines whether to return the associated word based on the syntactic category of the node.

4. If the node is the root of the tree, combines the p-norm formulas that were to be returned are combined into a single formula using an **AND** operator.

After all the sentences of a search request are processed, the p-norm formulations generated for them are combined into an **OR** clause with p-value 1.0.

The rules used by the algorithm to determine what to return when processing a particular tree node are presented in the following subsections.

Lexical Categories:

A tree node belonging to a lexical category appears as a leaf of the syntax tree, and has associated a single word of the text. When processing such a tree node, the algorithm returns either a query consisting only of the associated word, or an empty query, according to the following rules:

- DET, PRON, CONJ, PREP, and PUNC return empty query.
  Terms from these categories are not considered to be good content indicators, and they are therefore not included in the generated p-norm formula.

- ADV return empty query.
  Since adverbs are only occasionally good content identifiers, it was decided to always omit them.

- All other lexical categories check whether the word associated with the node is included in an ad-hoc list of words to be excluded. The lists used by the algorithm may be found in Figures 2.5, 2.6, and 2.7. These are all-purpose

appear	do	like	relate
associate	exist	list	result
base	find	look	see
be	get	make	shall
can	give	may	state
cause	have	might	study
compare	help	need	undertake
consider	hope	note	use
deal	include	obtain	want
describe	interest	oppose	will
determine	involve	pertain	write
discuss	know	regard	

Figure 2.5: Verbs to be Excluded

words typically found in queries; they generally are not good document identifiers. If the word is on any of these lists, it is omitted. Otherwise, the p-norm query consisting only of the single word associated with the node is returned.

Examples of these lexical categories include: NOUN, ADJ, and VERB. Adjectives are further restricted to be non-quantifying. Examples of adjectives considered by PLNLP to be quantifying include: *all, many, some, first, second*, etc.

Noun Phrases:

The description presented in this section is for non-conjoined[5] noun phrases only. The processing of conjoined noun phrases is described in the section dealing with conjoined clauses in general.

The structure of a noun phrase is:

$$premod_1 \ldots premod_k \; head \; postmod_1 \ldots postmod_m.$$

[5]A conjoined noun phrase is simply a noun phrase made up of several noun phrases joined by some conjunction, such as *and*. An example of a conjoined noun phrase is: *Journal articles, technical reports, and conference proceedings.*

able	easy	just	recent
above	either	likely	same
additional	entire	major	simple
all	esp.	main	some
another	essential	many	special
any	first	more	specialized
applicable	following	most	specific
available	further	much	such
basic	general	new	together
better	given	no	unable
best	helpful	only	useful
both	here	opposed	various
current	how	other	very
complete	important	particular	well
different	including	present	what
each	interested	primary	which

Figure 2.6: Adjectives to be Excluded

addition	e.g.	mechanism	relationship
application	emphasis	method	result
approach	etc.	name	role
area	example	notion	significance
aspect	explanation	one	special
association	factor	other	study
change	i.e.	overview	subject
comparison	interest	paper	subtopic
concept	introduction	problem	thing
consideration	issue	purpose	topic
description	literature	question	use
detail	material	regard	way
discussion	mean	relation	work

Figure 2.7: Nouns to be Excluded

The constituents preceding the head are the premodifiers, which may consist of adjectives, nouns, adjective phrases, etc. The head is a noun, and the postmodifiers are typically clauses such as prepositional phrases and participial clauses.

The algorithm processes each of the children of the node, combines the p-norm formulas from the premodifiers and the head into an **AND** clause, and then returns this formula along with the p-norm formulas returned by the postmodifiers. This process can be described formally as follows:

- Let Q_{prei} be the set of p-norm formulas returned by premodifier i.
- Let Q_h be the set of p-norm formulas returned by the head.
- Let Q_{posti} be the set of p-norm formulas returned by postmodifier i.
- Let Q_{join} be the **AND** clause whose operands are the p-norm formulas from Q_{pre1}, ..., Q_{prek} and Q_h.
- Return the p-norm formulas from Q_{post1}, ..., Q_{postm} and Q_{join}. These p-norm formulations will be joined by some ancestor of this tree node.

The postmodifiers are not ANDed into Q_{join} because of the difficulty in determining the correct placement of clauses such as prepositional phrases. The example in Figure 2.8 demonstrates this process.

Prepositional Phrases:

Non-conjoined prepositional phrases are handled the same way as noun phrases. The only difference between these two categories is that prepositional phrases include a preposition as the first premodifier of the head noun. Since PREP always returns an empty query, this preposition is ignored. See the example of Figure 2.8.

NP	NP	NOUN*	"code"	
	NOUN*	"optimization"		
	PP	PREP	"for"	
		NP	NOUN*	"space"
		NOUN*	"efficiency"	
	PUNC "."			

Leaf Nodes:

NOUN* (associated with *code*) returns query: Q_1 = *code*.
NOUN* (associated with *optimization*) returns query: Q_2 = *optimization*.
PREP (associated with *for*) returns empty query Q_3.
NOUN* (associated with *space* returns query: Q_4 = *space*.
NOUN* (associated with *efficiency* returns query: Q_5 = *efficiency*.
PUNC (associated with the period) returns empty query Q_6.

Prepositional Phrases:

PP (associated with *for space efficiency*) combines the p-norm formulas received from the two premodifiers, Q_3 and Q_4, and the p-norm formula received from its head, Q_5. This results in query: Q_7 = *space* **AND** *efficiency*.

Noun Phrases:

NP (associated with *code*) returns the only p-norm formula that it receives from its one child, namely Q_1.

NP (associated with *space*) returns the only p-norm formula that it receives from its one child, namely Q_4.

NP (associated with *code optimization for space efficiency*) joins the only p-norm formula from its premodifier, Q_1, with the p-norm formula from its head, Q_2. Thus, query

$$Q_{join} = \textit{code}\ \textbf{AND}\ \textit{optimization}.$$

is constructed. If this node had not been the root, it would have returned the two queries Q_{join} and the query from its postmodifier, Q_7. However, since this is the root, these two queries are combined into one **AND** clause, producing:

(*code* **AND** *optimization*) **AND** (*space* **AND** *efficiency*)

Figure 2.8: Example of a Syntactic Query Generation

Conjunctive Clauses:

A study presented in [DG87] discusses the ambiguity of the natural language conjunction *and* and presents some insights as to how it is best translated into a Boolean operator. An example of the ambiguity inherent in this conjunction is the following:

1. *I am interested in computers and education.*

2. *I am interested in compilers and interpreters.*

One needs to decide whether to translate the *and* into an AND or an OR. In both sentences the conjunction *and* is used, but the intended meanings are probably different. In particular, the first search request is better represented if the *and* is translated into an **AND** operator, since documents that are not about both *computers* and *education* are unlikely to be relevant. On the other hand, the *and* of the second search request is better translated as an **OR**, since documents dealing only with *compilers* or only with *interpreters* are likely to be relevant.

The interesting observation made in [DG87] is that when the conjuncts are semantically similar, as was the case in the second search request above, the intention behind the conjunction is to include alternative ways of expressing the concept of interest, and therefore the conjunction is best translated as an **OR**. On the other hand, if the conjuncts are semantically dissimilar as was the case in the first search request, the intention behind the conjunction is to identify a concept that is different from either conjunct alone — the concept *computers and education* is different from *computers* and different from *education*. Through the use of dictionary definitions, it is possible to determine whether two terms are semantically similar or dissimilar [CBH85]. This was the approach taken by the algorithm presented in [DG87] to automatically generate the proper translation of the *and* conjunction for a set of natural language search requests. In 90% of the cases, this algorithm agreed with a human's translation. However, it was pointed

out in [DG87] that in 72% of the conjunctions tested, the proper translation of the conjunction was the **OR**. Since the translation to an **OR** is so much more common, the algorithm will always translate the conjunction *and* into an **OR**.

An exception is made in the case of a conjoined prepositional phrase where the preposition is *between*. For example, if the natural language search request is:

I am interested in the relationship between data types and concurrency.

one would like to generate the following p-norm formulation for the prepositional phrase *between data types and concurrency*:

(data **AND** types) **AND** concurrency.

Whenever the searcher uses the preposition *between*, he is indicating that it is important that a retrieved document deal with both concepts. This special case is important, because it is common for a query to ask for comparisons, relationships, or interactions between concepts.

The conjunction *or* seems to be more straightforward than *and*. It may be used to enumerate several unrelated concepts of interest, as in:

I am interested in documents about programming languages, distributed systems, or information retrieval,

or it may be used to introduce alternative ways of expressing a given concept. In the latter case, the semantic similarity of the concepts joined by *or* may vary from being exact synonyms, as in:

I am interested in concurrent processing or parallel computation,

to being somewhat related, as in:

I am interested in the use of go-to or assignment in programming languages.

Regardless of how much similarity or dissimilarity there is between the concepts, the best translation for the *or* seems always to be the **OR** operator. The query

generation algorithm will therefore translate all natural language *ors* to **OR** operators.

Other items considered by PLNLP to be conjunctions include:

- certain punctuation marks such as commas, and
- certain phrases such as *as well as*.

These conjunctions are generally used for enumerating concepts, and therefore, when they are used as the head of a conjoined clause, they will also be translated to **OR** operators.

The structure of a conjunctive clause is:

conjunct CONJ *conjunct* CONJ ... CONJ* *conjunct.*

Conjunctive clauses can be identified by checking whether the head of the clause is labelled as CONJ* (PLNLP considers the last conjunction to be the head of the clause). A *conjunct* may consist of one or more constituents, as the conjoined noun phrase of the example in Figure 2.9 illustrates. The first conjunct consists of two constituents, namely, the NP for *performance* and the NP for *evaluation*, and the second conjunct consists of a single constituent, namely, the NP for *modelling of computer systems.* Furthermore, not all items labelled as CONJ in the parse tree for a conjunctive clause need to be the same, as can be seen in the example of Figure 2.10, where the conjoined prepositional phrase *in local networks, network operating systems, and distributed systems* contains two conjunctions, namely, the comma and the word *and.* However, the conjunction labelled as the head is the one that determines whether the conjoined clause generates an **AND** clause or an **OR** clause when joining the p-norm formulas returned by its children.

The processing of a conjunctive clause consists of first combining the p-norm formulas for the individual conjuncts into **AND** clauses, and then joining these **AND** clauses with an **OR** operator. Formally, one can describe this process as follows:

```
XXXX    NP*     NP      NOUN*   "performance"
                NP      NOUN*   "evaluation"
                CONJ*   "and"
                NP      NOUN*   "modelling"
                        PP      PREP    "of"
                                NP      NOUN*   "computer"
                                NOUN*   "systems"
        PUNC "."
```

Figure 2.9: Example of the Parse of a Conjunctive Clause

```
XXXX    NP*     NP      NOUN*   "security"
                NOUN*   "considerations"
                PP      PREP    "in"
                        NP      AJP     ADJ*    "local"
                                NOUN*   "networks"
                        CONJ    ","
                        NP      NP      NOUN*   "network"
                                AJP     ADJ*    "operating"
                                NOUN*   "systems"
                        CONJ*   "and"
                        NP      ADJP    ADJ*    "distributed"
                                NOUN*   "systems"
        PUNC "."
```

Figure 2.10: Conjoined Clause with Different Conjunctions

- Let $Q_{i1}, Q_{i2}, \ldots, Q_{in_i}$ be the p-norm formulas returned by the constituents of the i-th conjunct.
- Let $Q_i = Q_{i1}$ **AND** Q_{i2} **AND** ... **AND** Q_{in_i} for each conjunct i.
- Return Q_1 **OR** Q_2 **OR** ... **OR** Q_m, where m is the number of conjuncts in the clause.

Figure 2.11 illustrates this process.

This strategy is adequate for conjunctive clauses in which each premodifier and postmodifier serves as a modifier of exactly one conjunct. However, conjoined clauses may have a more complex construction, in which an individual constituent serves as a modifier of more than one conjunct. For example, the query

fast algorithm for context-free language recognition or parsing,

whose parse appears in Figure 2.12, has a conjoined prepositional phrase *for context-free language recognition or parsing* with the adjective *context-free* and the noun *language* serving as modifiers of both *recognition* and *parsing.* In order for the p-norm formula to accurately reflect the structure of this natural language statement, it must distribute the modifiers over all the conjuncts that they modify. This is accomplished with the p-norm formula:

(*context-free* **AND** *language* **AND** (*recognition* **OR** *parsing*)).

It is important to note that it is not always appropriate to distribute the modifiers over all the conjuncts. For example, the noun *data* in the sentence:

articles describing the relationship between data types and concurrency

only modifies *types.* Without semantic information, it is impossible to know whether it is appropriate to distribute the modifier over all conjuncts. However, in the phrase generation study presented in [Fag87], it was found that the following strategy for deciding whether to distribute a modifier was appropriate for a large number of cases:

PP	PREP	"in"	
	NP	NOUN*	"embryo"
	CONJ	","	
	NP	NOUN*	"fetus"
	CONJ	","	
	NP	NOUN*	"newborn"
	NP	NOUN*	"infant"
	CONJ*	"or"	
	NP	NOUN*	"animal"
PUNC "."			

Conjunct 1: Consists of $Q_1 = \textit{embryo}$

Conjunct 2: Consists of $Q_3 = \textit{fetus}$

Conjunct 3: Consists of

- $Q_4 = \textit{newborn}$
- $Q_5 = \textit{infant}$

Conjunct 4: Consists of $Q_6 = \textit{animal}$

PP node:

- Joins the p-norm formulas for conjunct 3 into an **AND** clause:

$$Q_7 = (\textit{newborn}\ \textbf{AND}\ \textit{infant})$$

- Since the other conjuncts consist of only one p-norm query, no joins are necessary for them.
- Returns the pnom formula obtained by joining the p-norm formulas corresponding to its conjuncts into an **OR** clause:

$$\textit{embryo}\ \textbf{OR}\ \textit{fetus}\ \textbf{OR}\ (\textit{newborn}\ \textbf{AND}\ \textit{infant})\ \textbf{OR}\ \textit{animal}$$

Figure 2.11: Processing of a Conjoined Prepositional Phrase

XXXX	NP*	AJP	ADJ*	"fast"	
		NOUN*	"algorithm"		
		PP	PREP	"for"	
			AJP	ADJ*	"context-free"
			NP	NOUN*	"language"
			NP	NOUN*	"recognition"
			CONJ*	"or"	
			NP	NOUN*	"parsing"
	PUNC "."				

Figure 2.12: Example of a Complex Conjoined Clause

> The only modifiers that can be distributed are the premodifiers of the first conjunct, and the postmodifiers of the last conjunct. Furthermore, the premodifiers of the first conjunct are distributed only if none of the other conjuncts have premodifiers. Similarly, the postmodifiers of the last conjunct are distributed only if none of the other conjuncts have a postmodifier.

The algorithm adopts this strategy in its handling of complex conjoined clauses.

The processing of conjoined clauses presented earlier can now be refined to include modifier distribution using the strategy from [Fag87]. Informally, modifier distribution is accomplished by constructing an **AND** clause whose operands are:

1. the modifiers to be distributed, and

2. the **OR** clause of the conjuncts (omitting the modifiers to be distributed).

As an example, consider the noun phrase

automatic analysis, storage, and retrieval of information.

In the parse that PLNLP produced, shown in Figure 2.13, one can see that only the first conjunct has a premodifier, namely, the adjective *automatic*, and only the last conjunct has a postmodifier, namely, the prepositional phrase *of information.* Thus, according to the strategy, they need to be distributed.

NP	NP	AJP	ADJ*	"automatic"
		NOUN*	"analysis"	
	CONJ	","		
	NP	NOUN*	"storage"	
	CONJ*	", and"		
	NP	NOUN*	"retrieval"	
		PP	PREP	"of"
			NOUN*	"information"
	PUNC "."			

Figure 2.13: Example of another Complex Conjoined Clause

The p-norm formulas returned by these two modifiers are:

1. $Q_{mod1} = automatic$

2. $Q_{mod2} = information$

The p-norm formulation for the first conjunct without the modifiers being distributed is:

$$Q_{conj1} = analysis.$$

Similarly, the p-norm formulation for the second conjunct is:

$$Q_{conj2} = storage,$$

and for the last conjunct is:

$$Q_{conj3} = retrieval.$$

To distribute the modifiers, the formulas from the conjuncts are ORed and then joined to the p-norm formulas from the modifiers being distributed in an AND clause. Thus, the following p-norm formula is constructed:

(*automatic* **AND** *information* **AND** (*analysis* **OR** *storage* **OR** *retrieval*)).

Formally, this more refined process for conjoined clauses can be described as follows:

XXXX	NP	NOUN*	"palliation"		
		NAPPOS	PUNC	"("	
			AJP	ADJ*	"temporary"
			NOUN*	"improvement"	
			PUNC	")"	
		PP	PREP	"of"	
			NP	NOUN*	"cancer"
			NOUN*	"patients"	
	PUNC "."				

Figure 2.14: Example of a Noun Appositive

- Let M be the set of modifiers to be distributed based on the above strategy.
- Let Q_{i1}, Q_{i2}, ..., Q_{in_i} be the p-norm formulas returned by all the constituents of the i-th conjunct excluding the p-norm formulas corresponding to the modifiers in M.
- Let $Q_i = Q_{i1}$ **AND** Q_{i2} **AND** ... **AND** Q_{in_i} for each conjunct i.
- Let $Q_{or} = Q_1$ **OR** Q_2 **OR** ... **OR** Q_m, where m is the number of conjuncts in the clause.
- Return the **AND** whose operands are Q_{or} and the p-norm formulas corresponding to the modifiers in M.

Noun Appositives:

A noun appositive is a grammatical construction immediately following a noun or noun phrase typically used to introduce an alternative way of expressing the noun that it modifies or as a means of clarification. For example, the noun phrase

palliation (temporary improvement) of cancer patients

contains the noun appositive *temporary improvement*, which clarifies the meaning of *palliation*. PLNLP was able to recognize the noun appositive, as shown in Figure 2.14.

The processing of a noun appositive node is then to simply combine the p-norm formulas returned by its children in an **AND** clause. However, the processing of all tree nodes with a NAPPOS child will have to be extended to do the following:

> Any time a child labelled as NOUN is followed by a NAPPOS node join the p-norm formula for the NOUN and the p-norm formula for the NAPPOS in an **OR** clause.

In the example of Figure 2.14, the following p-norm formula is produced:

> ((*palliation* **OR** (*temporary* **AND** *improvement*)) **AND** (*cancer* **AND** *patients*))

Parenthesized Expressions:

The constituents within a parenthesized expression generally make up a concept which is inserted into a sentence for explanatory purposes. In order to accurately represent the concept associated with the parenthesized expression, all the constituents within the parentheses need to be combined into an **AND** clause. However, PLNLP often has difficulties parsing sentences with parenthesized expressions. As the example of Figure 2.15 illustrates, PLNLP sometimes does not group together the constituents of the parenthesized expression into one construct and often has difficulties in determining how the expression fits in with the rest of the sentence. In this example, all the individual constituents of the expression appear at the same level as the noun phrase *the role of information retrieval in knowledge based systems*, and at the same level as the punctuation marks. To remedy this situation somewhat, the algorithm does the following:

- identifies parenthesized expressions when processing a node by searching for a child whose node is associated with the punctuation mark "(", followed by a set of children, and then followed by another child whose node is associated with the punctuation mark ")", and

```
XXXX NP*   DET     ADJ*    "the"
           NOUN*   "role"
           PP      PREP    "of"
                   NP      NOUN*   "information"
                           NOUN*   "retrieval"
                   PP      PREP    "in"
                           NP      NOUN*   "knowledge"
                                   PTPRTCL VERB*   "based"
                           NOUN*   "systems"
     PUNC  "("
     NP    NOUN*   "i.e."
     AJP   PUNC    ","
           ADJ*    "expert"
     NP    NOUN*   "systems"
     PUNC  ")"
     PUNC  "."
```

Figure 2.15: Example of a Failed Parse with a Parenthesized Expression

- joins the constituents in between the parentheses into an **AND** clause.

All other Non-Lexical Categories:

The algorithm recursively calls itself on the children, and returns the set of p-norm formulas that it receives. These formulas are joined only if the node is the root of the parse tree.

This concludes the description of the algorithm's processing of the various syntactic categories.

Before the syntactic query generation method processes the parse tree with this set of rules, the pre-processing steps described below are performed as a further refinement.

- **Including NOTs:** PLNLP, in addition to producing a parse, can recognize when some construction appears negated in a sentence. For example, if the query is:

I am not interested in logic programming,

PLNLP labels the entire sentence with a NEG tag. In the preprocessing step, one take advantage of this by setting a flag indicating that the entire query generated is to be embedded inside a NOT operator.

PLNLP is able to recognize when only a portion of the sentence is being negated. For example, the sentence

I am interested in interfaces for users with no computer background

would produce a parse in which the noun phrase *computer background* is labelled with a NEG tag. However, the algorithm is restricted to applying a NOT operator only when the entire sentence is negated. The reasoning behind this is that a portion being negated in the text does not necessarily indicate that a document must not satisfy it. The sentence above shows that, on the contrary, a relevant document retrieved must deal with *computer background.*

- **Recognizing Typical Query Patterns:** It is common for the structure of a natural language search request to follow certain patterns. Examples of queries that have typical patterns include:

  1. *I am interested in X.*
  2. *I want information about X.*
  3. *I would like documents about X.*
  4. *I want references on X.*
  5. *List all articles about X.*
  6. *Give me information about X.*
  7. *Information on X.*
  8. *Articles about X.*

The first four queries start with a pronoun, followed by a verb requesting or expressing interest, which may be followed by some *information noun*, and end with a phrase describing the information desired. The *information nouns* in the sample queries above are information, documents, references, and articles. Queries five and six are imperative sentences consisting first of a verb for the request, which may be followed by a pronoun, followed by an *information noun*, and ending with a phrase describing the requested information. The last two queries are simply noun phrases consisting of an information noun followed by the description of the desired information.

These common query patterns can be generalized as follows:

1. PRON WANT-VERB INFO-NOUN REST-OF-QUERY
2. WANT-VERB INFO-NOUN REST-OF-QUERY
3. INFO-NOUN REST-OF-QUERY

where:

WANT-VERB = {find, get, give, include, interest, like, list, look, need, describe, note, show, want}, and

INFO-NOUN = {article, book, citation, document, information, reference, research, study, work}.

If the natural language statement follows any of these three patterns, the p-norm query will be constructed from the section labelled REST-OF-QUERY.

The WANT-VERBs are all included in the list of words to be excluded shown in Figures 2.5, 2.6, and 2.7. However, the INFO-NOUNs are not included because an INFO-NOUN can sometimes be an important part of the query, and thus, appear in the section labelled as REST-OF-QUERY as illustrated in the following examples:

- *I am interested in information retrieval.*
- *List all articles about document indexing.*

Clearly, not all queries follow these patterns. The searcher may simply go straight to the point and avoid all of this superfluous information. Whenever this happens, the p-norm query is constructed from the entire search request.

2.3 Conclusion

Now that we have a method for automatically generating p-norm queries, we need to evaluate the retrieval output produced by these queries, and to analyze the amount of computational time required to perform a retrieval run with these queries. These questions will be dealt with in the following chapters.

Chapter 3

Effectiveness of P-norm Queries

The purpose of this chapter is to evaluate the retrieval output produced by the syntactically generated queries. In order to accomplish this, the following parameters must first be considered:

1. weighting of document terms,
2. weighting of query terms,
3. weighting of query clauses, and
4. assignment of p-values.

The effectiveness of the syntactically generated queries is compared to the effectiveness of statistically constructed queries, manually constructed queries, and standard vector queries. The possibility of combining vector and p-norm retrieval is considered as well.

3.1 Document Term and Query Term Weighting

Two methods of weighting document terms in the p-norm model are described in the literature [SV85,Voo85,Fox83,Lee88,SBF83]. This section compares the

retrieval output produced by the syntactically generated queries under the two weighting schemes.

Both schemes weight a document term with a normalized product of a *term frequency* component and an *inverse document frequency* component. The formula used to compute the term frequency component in both schemes is

$$\begin{cases} 0.5 + 0.5\frac{tf}{maxtf} & \text{if } tf > 0 \\ 0 & \text{if } tf = 0 \end{cases}$$

where tf is the number of times the term appears in the text of the document and $maxtf$ is the maximum frequency of any term (excluding stop words) in the document. The frequency of the term is normalized by $maxtf$ and then further normalized to lie between 0.5 and 1.0. The intuition behind this formula is that terms appearing frequently in the text of a document are likely to be better representatives of the document's content than are terms appearing infrequently. The nonzero values are made to lie in the interval [0.5, 1.0] to make a significant distinction between the weight of an infrequent term and the weight of a term that does not occur at all in the text of the document.

The inverse document frequency component varies inversely with the number of documents to which the term is assigned. This reflects the fact that terms that appear distributed throughout the entire collection are less likely to be important than terms that are concentrated in only a few documents. The formula used by both schemes is again the same:

$$\log(N/n),$$

where N is the number of documents in the collection and n is the number of documents to which the term is assigned.

In determining the actual document term weights, both weighting schemes start with a vector $\overline{w}$ consisting of unnormalized weights:

$$\overline{w} = (w_1, w_2, \ldots, w_m)$$

where

$$w_i = \begin{cases} \left(0.5 + 0.5\frac{tf_i}{maxtf}\right) \cdot log(N/n_i) & \text{if } tf_i > 0 \\ 0 & \text{if } tf_i = 0 \end{cases}$$

Because the p-norm model requires that each w_i lie in the interval [0,1], $\overline{w}$ must be normalized. The two schemes differ only in the way they normalize $\overline{w}$.

The weighting scheme used in [SV85,Voo85] normalizes $\overline{w}$ so that its 2-norm is 1. That is, each w_i is divided by

$$\|\overline{w}\|_2 = \sqrt{\sum_{i=1}^{m} w_i^2}$$

This is known as *cosine normalization.* The purpose of normalizing the weights with respect to the norm of the vector is to downweight long documents, which would otherwise be more likely to have higher similarity values than short documents. The weights produced by this scheme will be referred to as *tf·idf-cosine weights.*

The weighting scheme introduced by Edward Fox [Fox83,Lee88] simply divides each w_i by $log(N)$. Thus, a document term weight becomes

$$\begin{cases} (0.5 + 0.5\frac{tf}{maxtf}) \cdot \frac{\log(N/n)}{\log(N)} & \text{if } tf > 0 \\ 0 & \text{if } tf = 0. \end{cases}$$

These weights will be referred to as *Fox weights.*

Retrieval runs were performed to compare the two document weighting methods. The following parameter settings were used in these experiments:

1. All p-values are set to 1.0.

2. The query terms are given weight

$$\frac{\log(N/n)}{log(N)}$$

where N is the collection size and n is the number of documents to which the query term is assigned.

Table 3.1: Tf·idf-cosine Weights vs. Fox Weights

Collection		tf·idf-cosine weights	Fox Weights
CACM	Avg. Prec.	.2835	.3170
	Prec. top 10	.3098	.3078
INSPEC	Avg. Prec.	.2403	.2667
	Prec. top 10	.3753	.4104
MEDLARS	Avg. Prec.	.5833	.5932
	Prec. top 10	.6100	.6367

3. **AND** clauses are given weight equal to the sum of the weights of the operands.

4. **OR** clauses are given weight equal to the average weight of the operands.

5. **NOT** clauses are given the weight of their operand.

Various methods of clause weighting and p-value assignment are presented in the next section. The settings used here are chosen for simplicity of presentation; other parameter settings lead to the same conclusions.

Table 3.1 contains the retrieval performance obtained with these two weighting methods, where retrieval effectiveness is measured in terms of average precision at recall-levels of .25, .50, and .75, and precision in the top 10 ranked documents. This table shows that Fox document weights yield better retrieval output than tf·idf-cosine weights in both CACM and INSPEC. In CACM an increase of 11.8% in average precision with essentially identical precision in the top 10 documents was observed. In INSPEC an increase of 11.0% in average precision and an increase of 9.4% in precision of the top 10 documents was observed. In MEDLARS, only a minimal increase of 1.7% in average precision and an increase of 4.4% in the precision of the top 10 documents was observed. These results suggest that Fox document weights are in general preferable to

tf·idf_cosine weights.

Normalization factors, such as cosine normalization, that equalize the norm of the document vectors have been shown to improve retrieval output significantly[SB88] in experiments using automatically indexed vector queries. The question then arises why unnormalized Fox weights perform better than normalized weights in these p-norm experiments. In order to determine whether the superiority of the unnormalized weights was due to the hierarchical structure of the queries, the syntactically generated p-norm queries were transformed into vectors and the retrieval output of these vectors was then analyzed. The transformation consists in the removal of the p-norm operators by simply constructing a vector made up of all the terms from the p-norm query. For example, query

$$t_1 \textbf{ AND } (t_2 \textbf{ OR } t_3)$$

is transformed into vector

$$(t_1,t_2,t_3).$$

A retrieval run computing inner product similarities between these vectors and the document collection weighted by the two weighting schemes was then performed. The results in Table 3.2 show that Fox weights still produce superior retrieval output. An increase of 12.6% in the average precision and 7.4% in the top 10 precision of CACM, and an increase of 10.7% in the average precision and 8.9% in the top 10 precision of INSPEC was obtained. As before, there was not much change in MEDLARS.

Since these experiments were performed on vectors, the results indicate that the superiority of Fox weights is not due to the p-norm query structure. The difference between these vectors and the standard automatically constructed vector queries [SB88] is the vocabulary. The set of query terms used in the syntactically generated p-norm queries was more carefully selected — a large stop word list was used and syntactic information was also used to remove other words, such

Table 3.2: Tf-idf-cosine Weights vs. Fox Weights on Flattened P-norm Queries

Collection		Tf-idf-cosine Weights	Fox Weights
CACM	Avg. Prec.	.2784	.3135
	Prec. top 10	.2902	.3118
INSPEC	Avg. Prec.	.2459	.2723
	Prec. top 10	.3818	.4156
MEDLARS	Avg. Prec.	.5542	.5541
	Prec. top 10	.5800	.5833

as adverbs and INFO-NOUNs. This suggests that when the vocabulary used in the queries is carefully selected, it is best not to normalize the document term weights over the length of the document vectors. A simple measure of the amount of matching between a document and a query seems to be good enough without taking document length into account.

We now turn to query term weights. The only proposed method for weighting query terms is the normalized idf weight

$$\frac{log(N/n)}{log(N)},$$

where N is the collection size and n is the number of documents to which the query term is assigned. This weighting is used in all experiments in this thesis.

3.2 Clause Weighting and P-value Assignment

In previous work with p-norm queries, both **AND** and **OR** clauses have been given weight equal to the average weight of the operands. Since an **OR** operator is generally used as an indication that it is acceptable for only one of the operands to be satisfied, it seems appropriate for an **OR** clause to be given a weight representing the typical importance of its operands. However, this approach

for weighting AND clauses seems less appropriate. In this section, alternative weighting schemes for AND clauses are presented and analyzed.

In the syntactic query generation algorithm, an AND operator was used to join constituents of grammatical structures which typically represent a concept when combined. For example, the AND operator was used to combine the premodifiers with the heads of NPs and PPs.

A concept that is described by two or more terms is generally going to be much more specific than a concept that is described by only one term. These concepts are also less likely to be ambiguous — a common problem with one term concepts. So, it seems that, in general, AND clauses should receive higher weights than single terms.

A weighting scheme considered here is to give AND clauses a weight equal to the sum of the weights of its operands, i.e. the importance in satisfying an AND clause is considered to be the importance of satisfying the individual constituents. So, for example, the AND clause

$$(\textit{database},.3)\ \textbf{AND}\ (\textit{sorting},.5)$$

is given weight .8, which is the importance of a document being about *databases* plus the importance of a document being about *sorting*. This clause weighting method will be referred to as *sum-weights*[1]. This weighting approach can be viewed as a method of assigning idf weights to AND clauses, where the frequency of the concept represented by the clause is calculated by assuming that the concepts the clause is composed of are independent. For example, suppose that the clause consists of terms t_1 and t_2 with frequency n_1 and n_2, respectively. If independence is assumed, the frequency of the clause is estimated as

$$\frac{n_1 \cdot n_2}{N},$$

[1]Note that the sum of the weights may result in a value > 1. This is not a problem in the p-norm model because of the normalization that is performed in evaluating a clause. An equivalent formula with weights in the interval [0,1] can be obtained by dividing all the weights by the maximum weight assigned in the query.

where N is the size of the collection. It is easy to show that the idf weight corresponding to this frequency is

$$\frac{log(\frac{N}{n_1})}{logN} + \frac{log(\frac{N}{n_2})}{logN}.$$

The other method of weighting **AND** clauses that is proposed here is based on the observation that sometimes terms that are not very specific can represent very specific topics when combined into a clause. For example, in the CACM collection, the terms *data* and *type* are very general. However, when combined into the **AND** clause

data **AND** *type*,

they represent a very good concept, namely the concept *data type*. The importance of a document being indexed by both *data* and *type* is much greater than it would be if the terms *data* and *type* were two general independent terms. Thus, in such cases, the importance associated with the clause based on independence assumptions is inaccurately low. The approach taken in this method is to assign such clauses a somewhat higher weight than the *sum-weights* method as follows:

1. Let S be the sum of the weights of the operands.

2. If $S < 1.0$, then the clause is assigned weight

$$min(\ 1.0,\ S \text{ increased by } x\%).$$

3. If $S \geq 1.0$, then the clause is assigned weight S.

This method will be referred to as *sum-weights-modified*. The percentage increases considered in the experiments are 75% and 100%.

If the value obtained when S is increased by $x\%$ is greater than 1.0, it is truncated down to 1.0. The intuition behind this truncation is that if the components

of a clause represent a single concept when combined, then the maximum appropriate weight that the clause can have is 1.0[2]. Because of the truncation, the weight of a clause whose sum weight is not much lower than 1.0 is only increased slightly, while the weight of a clause whose sum weight is very low is increased by the full $x\%$. The effect of this approach is that the percentage increase is higher for clauses consisting of only very general terms than for clauses consisting of medium specificity terms. A clause for which the sum of the weights of the operands is greater than 1.0 probably consists of specific terms, and therefore, need not have its weight increased.

These clause weighting methods were compared in a set of retrieval runs with various combinations of schemes for p-value assignment to **AND** and **OR** operators. In past work, p-values were assigned uniformly to both the **AND**and **OR**operators. Only minimal improvements (if any) to the retrieval effectiveness was produced by the uniform assignment of p-values greater than 1.0 when compared to the retrieval output produced by a p-value assignment of 1.0 to all operators [SV85]. Clearly sometimes a strict operator is desirable while at other times it is not. So, a non-uniform scheme for assigning p-values to **AND** operators is considered.

The non-uniform scheme presented here assigns p-value $p > 1.0$ to all **AND** operators[3] except when:

1. Some concept in the clause is very specific (even if the other concepts are general). For example,

sorting **AND** *methods.*

[2]The maximum weight assigned to a single term with the *idf* formula being used here is

$$\frac{log\left(\frac{N}{1}\right)}{log(N)} = 1.0.$$

[3]The **AND** operators that are assigned p-value $p > 1.0$ under the non-uniform scheme will be referred to as *selected* **AND**s.

Since a strict **AND** has the undesirable effect of downweighting those documents that are only indexed by sorting, p-value 1.0 is used.

2. None of the concepts are general.

In other words, a p-value greater than 1.0 is assigned when all the concepts in the clause are sufficiently general. This scheme can be described more precisely, as follows:

1. Let *maxwt* be the maximum weight of the operands.

2. Let *minwt* be the minimum weight of the operands.

3. Estimate the frequency of the operand with the maximum weight by solving for *f1* in

$$\frac{log\left(\frac{N}{f1}\right)}{log(N)} = maxwt$$

4. Similarly, estimate the frequency of the operand with the minimum weight. Call it *f2*.

5. All clauses for which *f1* is greater than 3% of the collection size and *f2* is greater than 5% of the collection size are assigned some p-value $p > 1.0$. In other words, only clauses containing a general concept[4] and not containing a specific concept[5] are given a p-value > 1.0. All other clauses are assigned p-value 1.0.

The following p-value assignment schemes for **AND** and **OR** operators were used in the experiments:

1. the **OR** operators were uniformly given p-value 1.0, 2.0, 3.0, or 4.0.

2. Both uniform and non-uniform schemes were used for assigning p-values to **AND** operators:

[4]The general concepts are taken to be those with frequency > 5%.
[5]The very specific concepts are taken to be those with frequency < 3%.

- All **AND** operators were uniformly given p-value 1.0, or 2.0.
- The non-uniform assignment of p-values to **AND** operators described above was used with $p = 1.5$, 2.0, 3.0, and 4.0.

The results obtained from these retrieval runs are shown in Tables 3.3, 3.4, 3.5, and 3.6 for MEDLARS, Tables 3.7, 3.8, 3.9, and 3.10 for CACM, and Tables 3.11, 3.12, 3.13, and 3.14 for INSPEC. For each collection and for each clause weighting method, a table is given containing the average precision and the precision of the top 10 documents retrieved.

From these tables, one can see that:

- The traditional average weighting method for **AND** clauses performs very poorly as compared with either the *sum-weights* or *sum-weights-modified* methods. All of the entries in the average clause weighting tables can be seen to be significantly lower than the corresponding entries in the *sum-weights* or *sum-weights-modified* clause weighting tables.
- In all three collections, the best precision in the top 10 documents appears to be in either of the *sum-weights-modified* tables with the following parameter settings:
  - **OR** operators with p-value 3.0 or 4.0
  - Selected **AND** operators with p-value 3.0
- There is little difference between the *sum-weights-modified* method with 75% as compared with 100% increase.
- With respect to average precision, there is little difference between the *sum-weights* and either of the *sum-weights-modified* methods, with the exception of CACM, in which there is some improvement in the *sum-weights-modified* as compared with the *sum-weights* clause weighting method.

- Using p-values greater than 1.0 on the **OR** operators generally has a positive effect.

- The non-uniform p-value assignment for **AND** operators seems to have a minimal effect. With this selective approach, only about 15% of **AND** clauses get assigned a p-value greater than 1.0, making it very difficult for these clauses to have a significant effect on the overall effectiveness.

In conclusion, the best parameter settings appear to be

- *sum-weights-modified* clause weighting with a 75% increase,

- **OR** p-values of 3.0, and

- selective assignment of p-value 3.0 on **AND** operators.

These settings are used in the comparisons with other systems presented in the following sections.

Table 3.3: Average Clause Weighting for MEDLARS

AND p-value Assignment		OR p-values			
		p=1.0	p=2.0	p=3.0	p=4.0
All ANDs with p=1.0	Avg. Prec.	.5188	.5257	.5206	.5179
	Prec. Top 10	.5533	.6000	.6067	.6033
All ANDs with p=2.0	Avg. Prec.	.5214	.5202	.5195	.5163
	Prec. Top 10	.5533	.5900	.5867	.5867
Selected ANDs with p=1.5	Avg. Prec.	.5207	.5281	.5238	.5214
Others with p=1.0	Prec. Top 10	.5533	.6000	.6067	.6067
Selected ANDs with p=2.0	Avg. Prec.	.5235	.5300	.5281	.5242
Others with p=1.0	Prec. Top 10	.5533	.6033	.6067	.6033
Selected ANDs with p=3.0	Avg. Prec.	.5242	.5305	.5274	.5238
Others with p=1.0	Prec. Top 10	.5467	.6067	.6100	.6067
Selected ANDs with p=4.0	Avg. Prec.	.5233	.5276	.5262	.5243
Others with p=1.0	Prec. Top 10	.5500	.6533	.6133	.6067

Table 3.4: *Sum-weights* Clause Weighting for MEDLARS

AND p-value Assignment		OR p-values			
		p=1.0	p=2.0	p=3.0	p=4.0
All ANDs with p=1.0	Avg. Prec.	.5932	.5984	.5976	.5944
	Prec. Top 10	.6367	.6500	.6533	.6533
All ANDs with p=2.0	Avg. Prec.	.5855	.5824	.5799	.5775
	Prec. Top 10	.6100	.6300	.6333	.6333
Selected ANDs with p=1.5	Avg. Prec.	.5946	.6021	.6001	.5974
Others with p=1.0	Prec. Top 10	.6367	.6500	.6533	.6533
Selected ANDs with p=2.0	Avg. Prec.	.5954	.6025	.6022	.6000
Others with p=1.0	Prec. Top 10	.6400	.6533	.6533	.6500
Selected ANDs with p=3.0	Avg. Prec.	.5969	.6051	.6043	.6025
Others with p=1.0	Prec. Top 10	.6400	.6533	.6533	.6500
Selected ANDs with p=4.0	Avg. Prec.	.5989	.6066	.6059	.6028
Others with p=1.0	Prec. Top 10	.6433	.6533	.6533	.6467

Table 3.5: *Sum-weights-modified* with 75% increase for MEDLARS

AND p-value Assignment		OR p-values			
		p=1.0	p=2.0	p=3.0	p=4.0
All ANDs with p=1.0	Avg. Prec.	.5949	.6048	.6078	.6055
	Prec. Top 10	.6400	.6667	.6667	.6633
All ANDs with p=2.0	Avg. Prec.	.5748	.5779	.5784	.5790
	Prec. Top 10	.6200	.6500	.6500	.6500
Selected ANDs with p=1.5	Avg. Prec.	.5955	.6066	.6086	.6082
Others with p=1.0	Prec. Top 10	.6367	.6667	.6700	.6633
Selected ANDs with p=2.0	Avg. Prec.	.5961	.6075	.6092	.6079
Others with p=1.0	Prec. Top 10	.6367	.6667	.6700	.6667
Selected ANDs with p=3.0	Avg. Prec.	.5957	.6062	.6089	.6084
Others with p=1.0	Prec. Top 10	.6367	.6700	.6733	.6733
Selected ANDs with p=4.0	Avg. Prec.	.5930	.6036	.6067	.6041
Others with p=1.0	Prec. Top 10	.6333	.6600	.6733	.6700

Table 3.6: *Sum-weights-modified* with 100% increase for MEDLARS

AND p-value Assignment		OR p-values			
		p=1.0	p=2.0	p=3.0	p=4.0
All ANDs with p=1.0	Avg. Prec.	.5948	.6048	.6080	.6059
	Prec. Top 10	.6400	.6667	.6667	.6633
All ANDs with p=2.0	Avg. Prec.	.5747	.5779	.5777	.5783
	Prec. Top 10	.6200	.6500	.6500	.6500
Selected ANDs with p=1.5	Avg. Prec.	.5957	.6067	.6089	.6082
Others with p=1.0	Prec. Top 10	.6367	.6667	.6700	.6633
Selected ANDs with p=2.0	Avg. Prec.	.5962	.6075	.6097	.6079
Others with p=1.0	Prec. Top 10	.6367	.6667	.6700	.6667
Selected ANDs with p=3.0	Avg. Prec.	.5955	.6061	.6084	.6082
Others with p=1.0	Prec. Top 10	.6367	.6700	.6733	.6733
Selected ANDs with p=4.0	Avg. Prec.	.5930	.6036	.6067	.6040
Others with p=1.0	Prec. Top 10	.6333	.6600	.6733	.6700

Table 3.7: Average Clause Weighting for CACM

AND p-value Assignment		OR p-values			
		p=1.0	p=2.0	p=3.0	p=4.0
All ANDs with p=1.0	Avg. Prec.	.2942	.2928	.2933	.2956
	Prec. Top 10	.2765	.2706	.2765	.2745
All ANDs with p=2.0	Avg. Prec.	.2657	.2720	.2727	.2732
	Prec. Top 10	.2529	.2471	.2490	.2490
Selected ANDs with p=1.5	Avg. Prec.	.2950	.2929	.2941	.2963
Others with p=1.0	Prec. Top 10	.2784	.2745	.2804	.2765
Selected ANDs with p=2.0	Avg. Prec.	.2945	.2899	.2919	.2943
Others with p=1.0	Prec. Top 10	.2804	.2725	.2745	.2725
Selected ANDs with p=3.0	Avg. Prec.	.2919	.2892	.2891	.2915
Others with p=1.0	Prec. Top 10	.2745	.2765	.2725	.2706
Selected ANDs with p=4.0	Avg. Prec.	.2904	.2896	.2884	.2907
Others with p=1.0	Prec. Top 10	.2745	.2784	.2725	.2686

Table 3.8: *Sum-weights* Clause Weighting for CACM

AND p-value Assignment		OR p-values			
		p=1.0	p=2.0	p=3.0	p=4.0
All ANDs with p=1.0	Avg. Prec.	.3170	.3209	.3219	.3253
	Prec. Top 10	.3078	3176	.3176	.3196
All ANDs with p=2.0	Avg. Prec.	.2915	.2951	.2955	.2967
	Prec. Top 10	.2824	.2902	.2961	.2961
Selected ANDs with p=1.5	Avg. Prec.	.3173	.3205	.3216	.3232
Others with p=1.0	Prec. Top 10	.3118	.3157	.3196	.3157
Selected ANDs with p=2.0	Avg. Prec.	.3181	.3203	.3200	.3222
Others with p=1.0	Prec. Top 10	.3118	.3137	.3196	.3157
Selected ANDs with p=3.0	Avg. Prec.	.3188	.3198	.3192	.3210
Others with p=1.0	Prec. Top 10	.3118	.3118	.3176	.3196
Selected ANDs with p=4.0	Avg. Prec.	.3193	.3189	.3198	.3211
Others with p=1.0	Prec. Top 10	.3118	.3118	.3176	.3176

Table 3.9: *Sum-weights-modified* with 75% increase for CACM

AND p-value Assignment		OR p-values			
		p=1.0	p=2.0	p=3.0	p=4.0
All ANDs with p=1.0	Avg. Prec.	.3314	.3340	.3360	.3387
	Prec. Top 10	.3294	.3314	.3373	.3392
All ANDs with p=2.0	Avg. Prec.	.3030	.3064	.3078	.3075
	Prec. Top 10	.3078	.3157	.3196	.3176
Selected ANDs with p=1.5	Avg. Prec.	.3319	.3356	.3388	.3399
Others with p=1.0	Prec. Top 10	.3314	.3333	.3373	.3392
Selected ANDs with p=2.0	Avg. Prec.	.3315	.3344	.3382	.3403
Others with p=1.0	Prec. Top 10	.3294	.3294	.3353	.3392
Selected ANDs with p=3.0	Avg. Prec.	.3324	.3339	.3381	.3401
Others with p=1.0	Prec. Top 10	.3314	.3275	.3412	.3412
Selected ANDs with p=4.0	Avg. Prec.	.3304	.3346	.3370	.3385
Others with p=1.0	Prec. Top 10	.3333	.3275	.3392	.3392

Table 3.10: *Sum-weights-modified* with 100% increase for CACM

AND p-value Assignment		OR p-values			
		p=1.0	p=2.0	p=3.0	p=4.0
All ANDs with p=1.0	Avg. Prec.	.3309	.3338	.3354	.3387
	Prec. Top 10	.3314	.3314	.3373	.3392
All ANDs with p=2.0	Avg. Prec.	.3048	.3059	.3067	.3070
	Prec. Top 10	.3098	.3176	.3176	.3137
Selected ANDs with p=1.5	Avg. Prec.	.3312	.3336	.3374	.3388
Others with p=1.0	Prec. Top 10	.3314	.3314	.3392	.3392
Selected ANDs with p=2.0	Avg. Prec.	.3308	.3333	.3371	.3391
Others with p=1.0	Prec. Top 10	.3294	.3294	.3373	.3392
Selected ANDs with p=3.0	Avg. Prec.	.3301	.3330	.3368	.3383
Others with p=1.0	Prec. Top 10	.3294	.3275	.3412	.3412
Selected ANDs with p=4.0	Avg. Prec.	.3292	.3335	.3354	.3368
Others with p=1.0	Prec. Top 10	.3294	.3235	.3373	.3373

Table 3.11: Average Clause Weighting for INSPEC

AND p-value Assignment		OR p-values			
		p=1.0	p=2.0	p=3.0	p=4.0
All ANDs with p=1.0	Avg. Prec.	.2196	.2179	.2145	.2124
	Prec. Top 10	.3545	.3429	.3390	.3377
All ANDs with p=2.0	Avg. Prec.	.2040	.2007	.1980	.1959
	Prec. Top 10	.3325	.3143	.3104	.3117
Selected ANDs with p=1.5	Avg. Prec.	.2203	.2182	.2146	.2126
Others with p=1.0	Prec. Top 10	.3532	.3442	.3390	.3351
Selected ANDs with p=2.0	Avg. Prec.	.2204	.2179	.2146	.2125
Others with p=1.0	Prec. Top 10	.3545	.3455	.3403	.3364
Selected ANDs with p=3.0	Avg. Prec.	.2204	.2179	.2144	.2123
Others with p=1.0	Prec. Top 10	.3532	.3455	.3377	.3364
Selected ANDs with p=4.0	Avg. Prec.	.2198	.2179	.2146	.2121
Others with p=1.0	Prec. Top 10	.3571	.3481	.3377	.3364

Table 3.12: *Sum-weights* Clause Weighting for INSPEC

AND p-value Assignment		OR p-values			
		p=1.0	p=2.0	p=3.0	p=4.0
All ANDs with p=1.0	Avg. Prec.	.2667	.2703	.2712	.2719
	Prec. Top 10	.4104	.4156	.4169	.4182
All ANDs with p=2.0	Avg. Prec.	.2509	.2516	.2495	.2486
	Prec. Top 10	.3870	.3792	.3805	.3779
Selected ANDs with p=1.5	Avg. Prec.	.2666	.2711	.2722	.2724
Others with p=1.0	Prec. Top 10	.4117	.4156	.4195	.4182
Selected ANDs with p=2.0	Avg. Prec.	.2658	.2709	.2717	.2724
Others with p=1.0	Prec. Top 10	.4104	.4143	.4182	.4182
Selected ANDs with p=3.0	Avg. Prec.	.2644	.2699	.2710	.2719
Others with p=1.0	Prec. Top 10	.4091	.4130	.4117	.4143
Selected ANDs with p=4.0	Avg. Prec.	.2637	.2691	.2702	.2709
Others with p=1.0	Prec. Top 10	.4130	.4130	.4065	.4104

Table 3.13: *Sum-weights-modified* with 75% increase for INSPEC

AND p-value Assignment		OR p-values			
		p=1.0	p=2.0	p=3.0	p=4.0
All ANDs with p=1.0	Avg. Prec.	.2671	.2748	.2772	.2778
	Prec. Top 10	.4117	.4221	.4234	.4247
All ANDs with p=2.0	Avg. Prec.	.2456	.2454	.2454	.2449
	Prec. Top 10	.3883	.3818	.3805	.3792
Selected ANDs with p=1.5	Avg. Prec.	.2664	.2739	.2763	.2777
Others with p=1.0	Prec. Top 10	.4104	.4221	.4221	.4234
Selected ANDs with p=2.0	Avg. Prec.	.2649	.2728	.2752	.2758
Others with p=1.0	Prec. Top 10	.4117	.4247	.4221	.4247
Selected ANDs with p=3.0	Avg. Prec.	.2641	.2708	.2731	.2740
Others with p=1.0	Prec. Top 10	.4117	.4247	.4260	.4247
Selected ANDs with p=4.0	Avg. Prec.	.2635	.2698	.2718	.2727
Others with p=1.0	Prec. Top 10	.4117	.4247	.4260	.4247

Table 3.14: *Sum-weights-modified* with 100% increase for INSPEC

AND p-value Assignment		OR p-values			
		p=1.0	p=2.0	p=3.0	p=4.0
All ANDs with p=1.0	Avg. Prec.	.2661	.2748	.2766	.2774
	Prec. Top 10	.4117	.4208	.4195	.4221
All ANDs with p=2.0	Avg. Prec.	.2453	.2457	.2456	.2454
	Prec. Top 10	.3857	.3844	.3857	.3805
Selected ANDs with p=1.5	Avg. Prec.	.2653	.2738	.2756	.2766
Others with p=1.0	Prec. Top 10	.4091	.4169	.4182	.4195
Selected ANDs with p=2.0	Avg. Prec.	.2643	.2729	.2749	.2755
Others with p=1.0	Prec. Top 10	.4091	.4221	.4195	.4195
Selected ANDs with p=3.0	Avg. Prec.	.2627	.2709	.2722	.2733
Others with p=1.0	Prec. Top 10	.4117	.4260	.4247	.4273
Selected ANDs with p=4.0	Avg. Prec.	.2619	.2699	.2714	.2719
Others with p=1.0	Prec. Top 10	.4117	.4260	.4247	.4273

3.3 The Effects of Hierarchical Structure and P-values

The goal of this section is to determine the effect that the hierarchical structure of p-norm queries and the p-values greater than 1 have on retrieval effectiveness. To determine the effect of the structure, the p-norm queries are compared against their *flattened* version. The flattening process consists in removing the structure of the query by applying an $\mathbf{OR}^{1.0}$ to all the terms in the query. For example, query

$$(t_1, wt_1)\ \mathbf{AND}^{p_1}\ (\ (t_2, wt_2)\ \mathbf{OR}^{p_2}\ (t_3, wt_3)\ )$$

is flattened into

$$(t_1, wt_1)\ \mathbf{OR}^{1.0}\ (t_2, wt_2)\ \mathbf{OR}^{1.0}\ (t_3, wt_3).$$

Table 3.15 shows how the retrieval output of the structured p-norm queries with the p-value of 1.0 on all operators and *sum-weights-modified* clause weighting compares with that of the flattened p-norm query. This table also shows how the optimal run of the p-norm queries compares with the flattened p-norm queries. As can be seen, the results are not consistent. MEDLARS and CACM benefit a moderate amount from the hierarchical structure, but INSPEC does not benefit at all.

Table 3.16 compares the retrieval output of the p-norm queries with p-value 1.0 and with the set of p-values that was found best, namely **OR** operators with p-value 3.0 and selected **AND** operators with p-value 3.0. In this Table, only those queries with p-value other than 1.0 are used for comparison, since we are only interested in finding the effect that p-values have. Only small gains are observed — the most being again in MEDLARS, and the least in INSPEC. The precision in the top 10 documents is improved a bit more than the average precision.

Table 3.15: Flat P-norm Queries vs. Structured P-norm Queries

Collection	Flat Queries	Structured p=1.0	% Change	Structured mixed p	% Change
MEDLARS					
Avg. Prec.	.5541	.5949	+7.4%	.6089	+9.9%
Prec. top 10	.5833	.6400	+9.7%	.6733	+15.4%
CACM					
Avg. Prec.	.3135	.3314	+5.7%	.3381	+7.8%
Prec. top 10	.3118	.3294	+5.6%	.3412	+9.4%
INSPEC					
Avg. Prec.	.2723	.2671	-1.9%	.2731	+0.3%
Prec. top 10	.4156	.4117	-0.9%	.4260	+2.5%

Table 3.16: P-norm Queries with p=1.0 vs. P-norm Queries with mixed p-values

Collection		p=1.0	mixed p-values	Improvement
MEDLARS	Avg. Prec.	.5869	.6038	+2.9%
	Prec. top 10	.6560	.6960	+6.1%
CACM	Avg. Prec.	.3200	.3293	+2.9%
	Prec. top 10	.3216	.3378	+5.0%
INSPEC	Avg. Prec.	.2822	.2897	+2.7%
	Prec. top 10	.4500	.4677	+3.9%

The effectiveness is best when the hierarchical structure and the p-value assignment are combined, since both of these aspects of p-norm queries lead to some improvement.

3.4 Statistically vs. Syntactically Generated Queries

In this section, the retrieval effectiveness of the queries generated by the two p-norm query generation algorithms is compared. The statistically generated queries were derived from the algorithm described in [SBF83], and the syntactically generated queries were derived from the algorithm presented in Chapter 2.

A set of retrieval experiments using the statistically generated p-norm queries was performed with the following settings:

1. The number of desired documents was taken to be 30.
2. P-values used were uniformly 1.0 or 2.0. These were found generally to be best in [SBF83].
3. The clause weighting schemes considered were:
   (a) Traditional average weighting of operands for **AND** and **OR** operators.
   (b) *Sum-weights* method.

The retrieval output from these experiments is compared against the best retrieval output from the syntactically generated experiments, namely *sum-weights-modified* clause weighting with a 75% increase, **OR** p-values of 3.0, and a non-uniform assignment of p-value 3.0 on **AND** operators.

The retrieval effectiveness obtained from these methods is presented in table 3.17, where the top number for each entry represents the average precision at

Table 3.17: Statistically vs. Syntactically Generated Queries

	MEDLARS	% change	CACM	% change	INSPEC	% change
Syntactic	.6089	-	.3381	-	.2731	-
(Best Params)	.6733	-	.3412	-	.4260	-
Statistical	.5013	-17.7%	.2634	-22.1%	.1758	-35.6%
Avg Wts, p=1	.5700	-15.3%	.2654	-22.2%	.2896	-32.0%
Statistical	.4981	-18.2%	.2560	-24.3%	.1665	-39.0%
Avg Wts, p=2	.5367	-20.3%	.2462	-27.8%	.2558	-40.0%
Statistical	.4954	-18.6%	.2550	-24.6%	.1692	-38.0%
Sum-weights,p=1	.5800	-13.9%	.2712	-20.5%	.2896	-32.0%
Statistical	.5154	-15.4%	.2644	-21.8%	.1684	-38.3%
Sum-weights,p=2	.5733	-14.9%	.2558	-25.0%	.2636	-38.1%

recall points of .25, .50 and .75, and the bottom number represents the precision of the top 10 documents retrieved. The columns labelled *% change* contain the percent change of the various statistical runs when compared with the syntactic run. It can be seen that both the average precision and the precision of the top 10 documents produced by the statistically constructed queries range from 15% to 40% worse than the precision of the syntactically constructed queries. Thus, the syntactically generated p-norm queries are far better in the three collections.

3.5 Manual vs. Syntactically Generated Queries

In this section the retrieval effectiveness of the p-norm queries constructed manually is compared against the retrieval effectiveness of the syntactically generated p-norm queries. The manually constructed queries were created from the natural language text by Cornell graduate students for CACM, by Syracuse University students and faculty for INSPEC, and by actual searchers for MEDLARS.

The following four runs were performed with the manually constructed queries:

1. Queries treated as pure Boolean.

Table 3.18: Precision from Manual P-norm Runs

Collection		Boolean	Avg. Clause Wts. p=1.0	sum-weights-modified	
				p=1.0	sel. AND p=3.0 OR p=3.0
CACM	Avg.	.1395	.3344	.3361	.3154
	Top 10	.1780	.3380	.3480	.3240
INSPEC	Avg.	.0998	.2958	.2897	.2818
	Top 10	.1857	.4169	.4325	.4195
MEDLARS	Avg.	.1943	.5385	.5375	.5288
	Top 10	.3867	.6133	.6033	.6100

2. Traditional average clause weighting with p-value p=1.0.

3. *Sum-weights-modified* clause weighting with 75% increase, and p-value assignment of 1.0.

4. *Sum-weights-modified* clause weighting with 75% increase, and p-value 3.0 for **OR** operators, and p-value 3.0 for selected **AND** operators.

Table 3.18 shows the retrieval performance of these runs. The runs for MEDLARS were included for completeness, but due to some modifications that were made to the manually constructed queries through the use of the MESH thesaurus [Fox83], one cannot make any generalizations about manually constructed queries based on these runs.

The *Sum-weights-modified* clause weighting method can be seen to slightly improve the precision in the top 10 of INSPEC and CACM, as with the syntactically generated queries. However, the runs including p-values of 3.0 performed very poorly with the manually constructed queries. An observation that can be made is that these manual queries were written with a pure Boolean system in mind, and thus, there can be significant differences between these queries and

the queries that a user would have constructed for use in a p-norm system. For instance, in a pure Boolean system the **AND** operator has to be used very carefully because it can lead to a very small number of documents being retrieved or even no documents at all being retrieved. Thus, the **OR** operator is more heavily used than it would be in a p-norm system. That is, in a pure Boolean system an **OR** is often used to combine concepts to avoid restricting the size of the retrieved sets, even when the concepts being combined are not synonymous and even when the user is not indifferent to whether one or all of the concepts are satisfied by a retrieved document. For example, the manually constructed query corresponding to the natural language search request

> The use of operations research models to optimize information system performance. This includes fine tuning decisions such as secondary index selection, file reorganization, and distributed databases.

was the following:

(*operations* **AND** *research* **AND** *optimize* **AND** *performance*)

OR

(*secondary* **AND** *index* **AND** *selection*)

OR

(*file* **AND** *reorganization*)

OR

(*distributed* **AND** *databases*)

Because of the *weak* interpretation of the operators in the p-norm model, there is no harm done by using many **AND** operators in a query. In fact, this is what the syntactic p-norm query generation algorithm does — it uses **OR** operators only in representing conjoined natural language constructs and noun appositives. So, it is not surprising that the parameter settings that were found best for the syntactically generated p-norm queries are not necessarily good for the manually constructed queries.

Table 3.19: Syntactic Queries vs. Manual Queries

Collection		Manual	Syntactic
CACM	Avg. Prec.	.3361	.3381
	Prec. Top 10	.3480	.3412
INSPEC	Avg. Prec.	.2897	.2731
	Prec. Top 10	.4325	.4260
MEDLARS	Avg. Prec.	.5375	.6089
	Prec. Top 10	.6033	.6733

The best p-norm retrieval of the manually constructed queries is obtained with *sum-weights-modified* clause weighting and p-value 1.0. Table 3.19 shows how the best retrieval of manually constructed queries compares with the best retrieval of syntactically generated queries. As can be seen, the syntactic queries produce comparable results.

3.6 Standard Vector vs. Syntactically Generated Queries

In this section, the syntactically generated queries are compared against vector queries that are also automatically constructed from the natural language search request. The vector queries are constructed as follows:

1. All words from a *stop word list* are removed.

2. The remaining words are placed in vector form with *tf·idf* weight

$$\left(0.5 + 0.5 \cdot \frac{tf}{maxtf}\right) \cdot log\left(\frac{N}{n}\right),$$

where N is the number of documents in the collection, n is the number of documents to which the term is assigned, tf is the number of times the term occurs in the query text, and $maxtf$ is the maximum tf in the vector.

Table 3.20: Standard Vector Queries vs. Syntactic P-norm Queries

Collection		Vector Queries	P-norm Queries	% Change
CACM	Avg. Prec.	.3630	.3534[a]	-2.6%
	Prec. Top 10	.3635	.3423	-5.8%
INSPEC	Avg. Prec.	.2626	.2731	+4.0%
	Prec. Top 10	.4286	.4260	-0.6%
MEDLARS	Avg. Prec.	.5628	.6089	+8.2%
	Prec. Top 10	.6367	.6733	+5.7%

[a]In addition to the natural language statement of need, some queries in CACM include a list of names of individuals who have written articles on the subject of interest. These names are part of the standard vector query, and so, they were joined to the syntactically generated p-norm via an **AND** operator with p-value 1.0.

In the vector retrieval runs, the document terms are weighted according to the following *tf·idf* formula:

$$tf \cdot log\left(\frac{N}{n}\right),$$

where *tf* is the number of times the term occurs in the document text, N is the collection size, and n is the number of documents indexed by the term. These weights are then cosine normalized. These weights are standard and were found best in [SB88]. The similarity function used in these runs is inner product.

Table 3.20 shows how the effectiveness of a retrieval with the syntactically generated queries compares with that of the standard vector queries. Overall, the two methods are similar in retrieval effectiveness. However, it can be seen that the vector queries are somewhat better on CACM, while the p-norm queries are somewhat better on MEDLARS.

Possibly this difference between the performances on CACM versus on MEDLARS is of no significance. But it has been observed by Salton and Buckley

[SB88] that the vocabulary of MEDLARS is especially technical, and they conclude that as a consequence, document terms should be weighted differently in MEDLARS than in collections with more varied vocabulary, such as CACM. One might speculate, then, that terms in MEDLARS are in general more reliable than terms in CACM. Indeed, one can easily give examples of terminological difficulties in computer science:

- Certain concepts in computer science are commonly referred to by a variety of different names. For example, the terms *functional language*, *applicative language*, and *declarative language* all refer to more or less the same concept. So a search for papers about functional languages might easily miss a relevant paper that used the term *applicative language* instead.

- Many ordinary words are used in computer science with specialized meaning. Consider, for example, *system*, *type*, *object*, and *method*. A search for papers about type systems might easily be fooled into thinking that a paper called *"A new type of operating system"* was relevant.

In summary, in the CACM collection neither the absence nor the presence of any term in a document can be taken too seriously.

Now consider the characteristics of vector retrieval. Vector retrieval is very "relaxed": the similarity between a document and a vector is the result of accumulating all the terms in the document that match terms in the vector. The vector model does regard some terms as more important than others, but it doesn't make a big fuss over the presence or absence of any term. A good similarity value simply means that the document abstract uses a lot of the same words that the natural language query uses.

In contrast, p-norm retrieval is structured. The essence of this structure is that some terms *are* emphasized. Indeed, disjunctivity can loosely be described as making a big fuss over the presence of certain terms, while conjunctivity can

be described as making a big fuss over the absence of certain terms. As discussed above, in the CACM collection such behavior can easily lead to mistakes.

This discussion suggests that vector and p-norm retrieval might complement one another, especially in collections with unreliable terms. Possibly the vector model could exert a steadying influence on the p-norm model in such collections. The next section describes a system that uses a combination of vector and p-norm retrieval.

3.7 Combined Vector and P-norm Retrieval

The combined system averages the results of vector and p-norm retrieval. More precisely, to perform retrieval in the combined system do the following:

1. Perform vector retrieval.
2. Normalize the vector similarity values by dividing each similarity by the maximum similarity of the vector retrieval.
3. Perform p-norm retrieval.
4. Normalize the p-norm similarity values by dividing each similarity by the maximum similarity of the p-norm retrieval.
5. Rank the documents based on the average of the normalized vector and p-norm similarity values.

Table 3.21 shows how the combined system compares with the standard vector system. It can be seen that the combined system outperforms the vector system on all collections. The largest gains are in average precision. INSPEC, in particular, improves by 21% in average precision.

Table 3.22 shows how the combined system compares with the p-norm system. It is interesting to note that the combined system outperforms the p-norm

Table 3.21: Standard Vector System vs. Combined System

Collection		Vector Retrieval	Combined Retrieval	% Change
CACM	Avg. Prec.	.3630	.3987	+9.8%
	Prec. Top 10	.3635	.3692	+1.6%
INSPEC	Avg. Prec.	.2626	.3181	+21.1%
	Prec. Top 10	.4286	.4766	+11.2%
MEDLARS	Avg. Prec.	.5628	.6028	+7.1%
	Prec. Top 10	.6367	.6767	+6.3%

model on all collections except MEDLARS, the collection with the most technical vocabulary.

3.8 Conclusion

In this chapter, the quality of the syntactically generated p-norm queries was investigated. First a variety of parameter settings for these queries were explored, leading to a number of conclusions:

- Document terms are better weighted with uniformly normalized Fox weights [Fox83] than with the cosine normalized weights of [SV85,Voo85].
- **OR** clauses are best weighted with the average of the weights of their operands.
- **AND** clauses are best weighted with sum of the weights of their operands, and certain **AND** clauses should be further boosted in weight.
- **OR** operators are best given p-value 3.0.
- **AND** operators should usually be given p-value 1.0, but selected **AND**s with the non-uniform scheme should be given p-value 3.0.

Table 3.22: P-norm System vs. Combined System

Collection		P-norm Retrieval	Combined Retrieval	% Change
CACM	Avg. Prec.	.3534	.3987	+12.8%
	Prec. Top 10	.3423	.3692	+7.9%
INSPEC	Avg. Prec.	.2731	.3181	+16.5%
	Prec. Top 10	.4260	.4766	+11.9%
MEDLARS	Avg. Prec.	.6089	.6028	-1.0%
	Prec. Top 10	.6733	.6767	+0.5%

It is encouraging that **AND** and **OR** clauses are best weighted differently and that p-values larger than 1.0 are sometimes useful, for these facts suggest that the p-norm structure of the queries does, to some extent, model the meaning of the natural language search requests. Experiments with flattened versions of the queries further support this conclusion: both hierarchically derived query weights and boosted p-values lead to improved retrieval effectiveness.

Comparisons between the syntactic query generation method and other methods of generating p-norm queries are also favorable. The syntactically generated queries are much better than statistically generated queries, and are comparable in quality to manually generated queries.

Comparisons between the syntactically generated p-norm queries and standard vector queries produce no clear-cut winner. However, it seems possible that the two systems complement one another. This observation led to consideration of a combined system that averages the results of vector and p-norm retrieval. The combined system produces significant gains in all collections as compared to the standard vector system alone.

In the next two chapters, attention is turned to the question of p-norm retrieval efficiency.

Chapter 4

Improving the Retrieval Time of the P-norm Model

To be practical, an information retrieval system must provide prompt responses to a user's search requests. This chapter compares the efficiency of two algorithms for p-norm retrieval:

1. the straightforward algorithm formerly implemented in the SMART system, and
2. a new algorithm, based on inverted list manipulation, which is introduced here.

An asymptotic analysis of these two algorithms is given, and the actual running times obtained in experimental retrieval runs are given as well.

4.1 Document Collection and Query Representation

Both algorithms assume that the document collection is represented by a set of inverted lists [Sal89]. There is an inverted list for each index term consisting of the document identifiers of all documents indexed by that term. Paired with each

document identifier is the weight of the term in that document. Furthermore, these document identifiers are assumed to appear in sorted order in the inverted lists.

For example, suppose that the collection consists of the following documents

$$\begin{aligned} D_1 &= ((t_1, w_{11}), (t_4, w_{14}), (t_8, w_{18})) \\ D_2 &= ((t_3, w_{23}), (t_4, w_{24})) \\ D_3 &= ((t_1, w_{31}), (t_2, w_{32}), (t_3, w_{33}), (t_4, w_{34})) \end{aligned}$$

where w_{ij} is the weight of term t_j in document D_i. Then the set of inverted lists representing the document collection is

$$\begin{aligned} t_1 &: ((D_1, w_{11}), (D_3, w_{31})) \\ t_2 &: (D_3, w_{32}) \\ t_3 &: ((D_2, w_{23}), (D_3, w_{33})) \\ t_4 &: ((D_1, w_{14}), (D_2, w_{24}), (D_3, w_{34}) \\ t_8 &: (D_1, w_{18}) \end{aligned}$$

P-norm queries are stored in the form of a tree, where each tree node corresponds to either a term or a Boolean operator. A node corresponding to a term appears as a leaf, and a node corresponding to a Boolean operator is an internal node with the operands appearing as the children. For example, the query A **AND** (B **OR** C) is represented by the tree in Figure 4.1.

4.2 Straightforward Algorithm

The straightforward algorithm evaluates one document at a time by traversing the query tree recursively beginning at the root. The value returned by a recursive call to a tree node is simply the similarity between the document and the subquery corresponding to the subtree with this tree node as the root. So to evaluate a document with respect to a node, one must

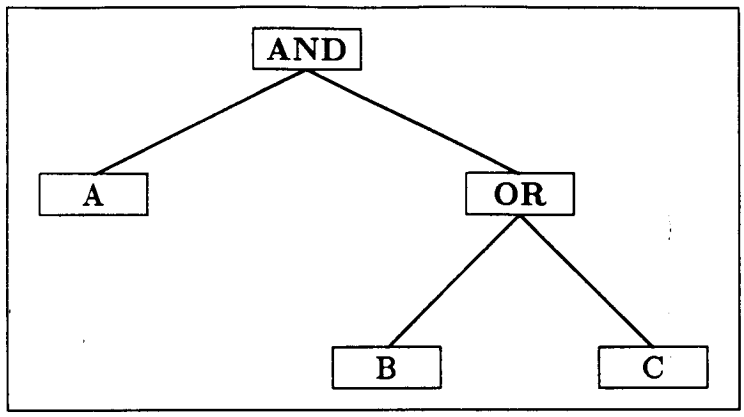

Figure 4.1: P-norm Query Represented as a Tree

- Recursively evaluate the document with respect to the children, if any.
- If the node is a leaf, then it corresponds to a query term, and one simply returns the weight of the query term in the document.
- If the node is not a leaf, then it corresponds to a clause; one returns the value calculated by combining the values returned by the children with the appropriate p-norm formula, which depends on the operator and p-value associated with the node.

Only documents having at least one term in common with the query are evaluated. All the other documents have similarity 0, so they are ignored[1]. The query evaluation method is demonstrated in Example 1, and stated formally in Figure 4.3.

Example 1: (Document Evaluation with the Straightforward Algorithm) Suppose that document

$$\mathrm{D} = ((\textit{clustering},.5),\ (\textit{efficiency},.2))$$

is to be evaluated with respect to p-norm query

[1]Actually, the similarity of these other documents may be nonzero when the NOT operator is used. However, those documents are not retrieved.

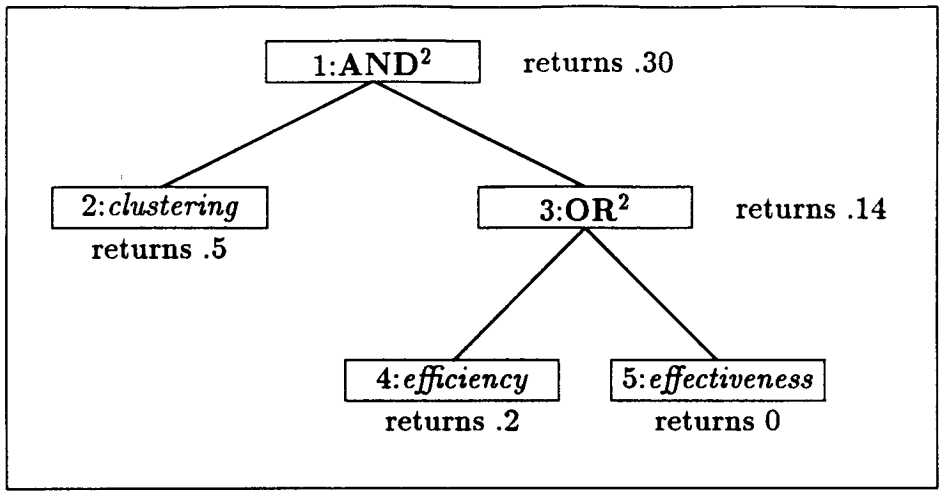

Figure 4.2: Query Tree for Example 1

$$Q = \textit{clustering}\ \mathbf{AND}^2\ (\textit{efficiency}\ \mathbf{OR}^2\ \textit{effectiveness})$$

Assume that all query term weights and clause weights are 1.0 for simplicity.

The corresponding query tree is shown in Figure 4.2, where each node is labelled with a number to facilitate the description of the evaluation process described below.

Node 1:

- Asks node 2 for evaluation of document D and receives .5
- Asks node 3 for evaluation of document D and receives .14
- Returns 1 - [L_2 distance[2] from (.5,.14) to (1,1)] = .30

[2]All L_p distances mentioned in this chapter are normalized L_p distances.

Node 2: Returns .5 (i.e. the weight of *clustering* in document D)

Node 3:

- Asks node 4 for evaluation of document D and receives .2
- Asks node 5 for evaluation of document D and receives 0
- Returns [L_2 distance from (.2,0) to (0,0)] = .14

Node 4: Returns .2 (i.e. the weight of *efficiency* in D)

Node 5: Returns 0 (D does not contain *effectiveness*)

This tree traversal is performed once for each document that is evaluated.

4.3 New Algorithm

Whereas the straightforward algorithm traverses the entire query tree once for each document, the new algorithm that is proposed here traverses the query tree only once as it evaluates the entire collection. The tree is traversed recursively, as in the straightforward algorithm. However, the value returned by a recursive call to a tree node is now an inverted list instead of a similarity value. The inverted list resulting from processing a tree node contains the document identifiers of all the documents with non-zero similarity with respect to the subquery with this tree node as the root. These document identifiers are paired with their associated similarity values. This can be accomplished by processing a tree node as follows:

- Recursively process the children, if any.
- If the tree node is a leaf, then its corresponding subquery is just a query term. Therefore, return the inverted list associated with the term.
- If the tree node is not a leaf, then the corresponding subquery is a clause. Return the list consisting of all documents that appear on any of the lists

```
Let Q be a p-norm query.
Let the terms in Q be q_1, q_2, ..., q_n.
for i = 1 to n
    get inv_i = inverted list for term q_i;
/* Process documents and remove them from lists in sorted order */
while any inverted list is not empty {
    Let D = minimum doc id in inv. lists inv_1, ..., inv_n;
            (Can be computed in O(n) time since the front of each
            contains the document with lowest id.)
    Sim = p_eval(D, Q);
    Add pair (D, Sim) to result_inv_list;
    Remove D from inverted lists
}
Return result_inv_list sorted by similarity value.

p_eval (D, Q) {
        if (Q is a term)
          return weight of term in document D;
          (This value is obtained from the inv. list entry).
        if (Q = NOT Q_1)
          return 1 - p_eval(D, Q_1);
        if (Q = (Q_1, wt_1) AND^p (Q_2, wt_2) AND^p ... AND^p (Q_m, wt_m)) {
          for i = 1 to m
              sim_i = p_eval(D, Q_i);
          return 1 - p-root( (wt_1^p (1-sim_1)^p + ... + wt_m^p (1-sim_m)^p) / (wt_1^p + ... + wt_m^p) )
        }
        if (Q = (Q_1, wt_1) OR^p (Q_2, wt_2) OR^p ... OR^p (Q_m, wt_m)) {
          for i = 1 to m
              sim_i = p_eval(D, Q_i);
          return p-root( (wt_1^p sim_1^p + ... + wt_m^p sim_m^p) / (wt_1^p + ... + wt_m^p) )
        }
}
```

Figure 4.3: Straightforward Algorithm

returned by the children. Each document is paired with the value obtained by combining into the proper p-norm formula the document's similarity values with each of the operands. If the document does not appear in some list, then it is handled as if it appeared on that list with value 0.

Thus, the value that the root of the query tree returns consists of the list of documents evaluating to a non-zero similarity along with their similarity values.

This query evaluation method is illustrated in the following example.

Example 2: (P-norm Retrieval with New Algorithm)
Suppose that a retrieval is to be performed with query

$$Q = \textit{clustering}\ \mathbf{AND}^2\ (\textit{efficiency}\ \mathbf{OR}^2\ \textit{effectiveness})$$

and suppose that the collection contains

$$\begin{aligned} D_1 &= ((\textit{clustering}, .5), (\textit{efficiency}, .2)) \\ D_2 &= ((\textit{efficiency}, .8), (\textit{algorithms}, .3)) \\ D_3 &= ((\textit{clustering}, .9)) \\ D_4 &= ((\textit{clustering}, .4), (\textit{effectiveness}, .2)) \end{aligned}$$

The query tree along with the values returned by each of the tree nodes is shown in Figure 4.4. The following is a description of the retrieval process:

Node 1:

- Asks node 2 for the list of documents that match the sub-query with root at node 2 (i.e. *clustering*), along with the values that they evaluate to. It receives list [(D_1,.5),(D_3,.9),(D_4,.4)].

- Asks node 3 for the list of documents that match the subquery with root at node 3 (i.e. *efficiency* $\mathbf{OR}^2$ *effectiveness*), along with the values that they evaluate to. It receives list [(D_1,.14),(D_2,.57),(D_4,.14)].

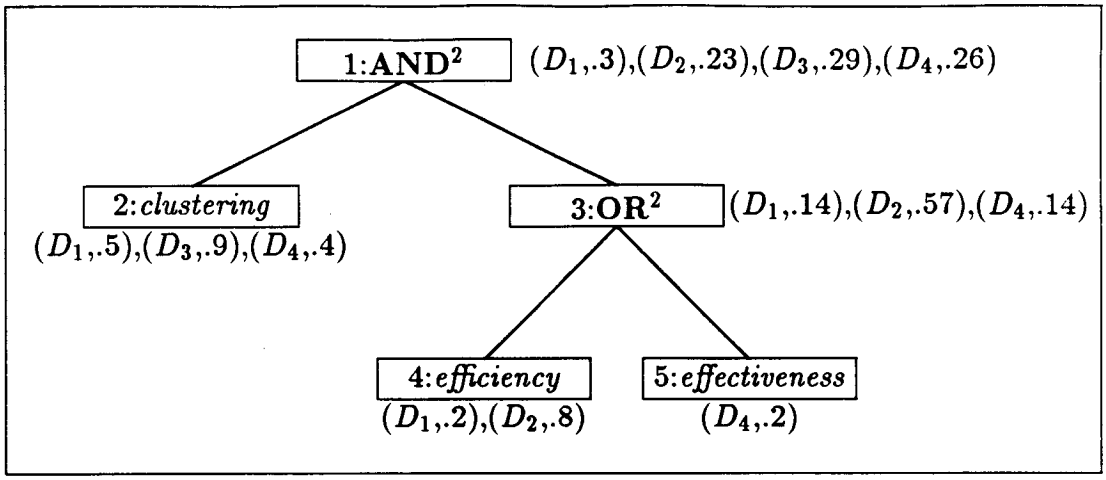

Figure 4.4: Query Tree for Example 2

- Computes the weight of document D_i as 1 - L_2((p,q),(1,1)) where

  p = weight of D_i in the list from Node 2

  and

  q = weight of D_i in the list from Node 3.

- Returns the list [(D_1,.3),(D_2,.23),(D_3,.29),(D_4,.26)]

Node 2: Returns the inverted list of *clustering*: [(D_1,.5),(D_3,.9),(D_4,.4)]

Node 3:

- Asks node 4 for the list of documents that match the subquery *efficiency*. It receives the list [(D_1,.2),(D_2,.8)].

- Asks node 5 for the list of documents that match the subquery *effectiveness*. It receives the list [(D_4,.2)].

- Computes the weight of document D_i as L_2((x,y),(0,0)) where

  x = weight of D_i in the list from Node 4

and

$$y = \text{weight of } D_i \text{ in the list from Node 5}$$

- Returns the list $[(D_1,.14),(D_2,.57),(D_4,.14)]$

Node 4: Returns the inverted list of *efficiency*: $[(D_1,.2),(D_2,.8)]$

Node 5: Returns the inverted list of *effectiveness*: $[(D_4,.2)]$

Note that only one tree traversal is performed in the evaluation of the entire collection.

A possible drawback of this method is that the inverted lists returned by a recursive call to this algorithm can be very long if many documents have terms in common with the subquery that the algorithm is called with. In particular, the **NOT** operator can create especially long inverted lists if we do not deal with it carefully. For example, the inverted list returned by a call to subquery "**NOT** Q_1" would contain almost the entire collection since

$$\begin{aligned} Sim(D, \mathbf{NOT}Q_1) \neq 0 &\Leftrightarrow 1 - Sim(D, Q_1) \neq 0 \\ &\Leftrightarrow Sim(D, Q_1) \neq 1 \end{aligned}$$

and generally very few documents are going to have similarity 1.0 with Q_1. Such a list would take up vast amounts of space for reasonably sized collections. However, the following observation can be made:

> Sim(D,**NOT** Q_1) is the same for all documents that are not indexed by any terms from Q_1.

This similarity value can be computed by simply associating a weight of 0.0 with all the leaves of the subtree of Q_1. Thus, the new algorithm is revised to return the list containing the documents having a term in common with the subquery, as before. In addition, the value of the similarity between the subquery and any document that is not indexed by any of the subquery terms is also returned. This

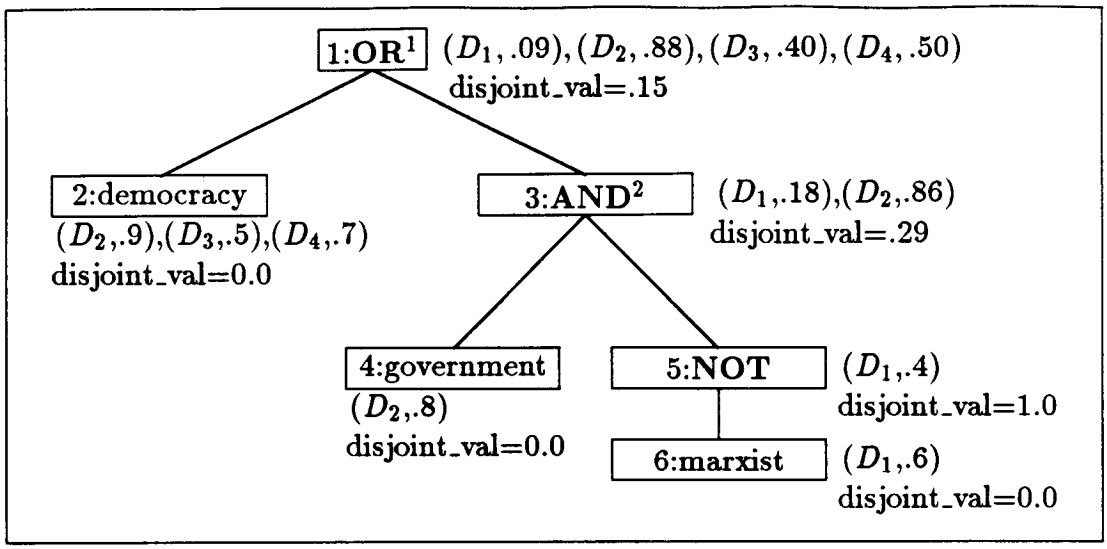

Figure 4.5: Query Tree for Example 3

value will be referred to as *disjoint_val.* The following example shows how this is accomplished.

Example 3: (Handling of NOT in New Algorithm)
Suppose that a retrieval is to be performed with query

$$Q = democracy\ \mathbf{OR}^1\ (government\ \mathbf{AND}^2\ \mathbf{NOT}\ marxist),$$

whose query term weights and clause weights are assumed to be 1.0 for simplicity, and suppose that the collection contains:

$$\begin{aligned} D_1 &= ((marxist, .6)) \\ D_2 &= ((government, .8), (democracy, .9)) \\ D_3 &= ((democracy, .5)) \\ D_4 &= ((democracy, .7), (elections, .8)) \end{aligned}$$

The query tree along with the values returned by each of the tree nodes is found in Figure 4.5, and the following is a description of the retrieval process:

Node 1:

- Asks node 2 for the list of documents that match the sub-query with root at node 2 (i.e. *democracy*), along with the value that all the non-matching documents would evaluate to. It receives the list $[(D_2, .9), (D_3, .5), (D_4, .7)]$ and disjoint_val = 0.

- Asks node 3 for the list of documents that match the subquery with root at node 3 (i.e. *government* **AND**^{2} **NOT** *marxist*) and for the value that all the non-matching documents would evaluate to. It receives the list $[(D_1, .18), (D_2, .86)]$ and disjoint_val = .29.

- Computes the weight for D_1 as L_1((p,.18),(0,0)) where

  p = disjoint_val returned by node 2 = 0.

- Computes the weight for D_2 as L_1((.9,.86),(0,0))

- Computes the weight for D_3 as L_1((.5,q)(0,0)) where

  q = disjoint_val returned by node 3 = .29

- Computes the weight for D_4 as L_1((.7,q)(0,0))

- Returns the list $[(D_1, .09), (D_2, .88), (D_3, .40), (D_4, .50)]$

- Returns disjoint_val = L_1((p,q),(0,0)) = .15

Node 2:

- Returns the inverted list of *democracy*: $[(D_2, .9), (D_3, .5), (D_4, .7)]$

- Returns disjoint_val = 0 since any document not matching query *democracy* evaluates to 0.

Node 3:

- Asks node 4 for the list of documents that match the subquery *government* and the disjoint_val associated with it. It receives the list [(D_2,.8)] and disjoint_val 0.
- Asks node 5 for the list of documents matching the subquery "**NOT** *marxist*" and the value that the non-matching documents evaluate to. It receives the list [(D_1, .4)] and disjoint_val = 1.
- Computes the weight for D_1 as 1 - L_2((x,.4),(1,1)) where

  x = disjoint_val returned by node 4 = 0.

- Computes the weight for D_2 as $1 - L_2$((.8,y)(1,1)) where

  y = disjoint_val returned by node 5 = 1.

- Returns the list [(D_1, .18), (D_2, .86)].
- Returns disjoint_val = $1 - L_2$((x,y),(1,1)) = .29

Node 4:

- Returns the inverted list of *government*: [(D_2,.8)].
- Returns disjoint_val = 0 since any document not matching the query *government* evaluates to 0.

Node 5:

- Asks Node 6 for the list of documents that match the subquery *marxist* and for its disjoint_val. It receives the list [(D_1, .6)] and disjoint_val = 0.
- Returns the list [(D_1, 1 − .6)] = [(D_1, .4)].
- Returns disjoint_val = 1 − (disjoint_val returned by node 6) = 1 − 0 = 1.

Node 6:

- Returns the inverted list of *marxist*: $[(D_1, .6)]$.
- Returns disjoint_val = 0 since any document not matching the query *marxist* evaluates to 0.

The formal description of this algorithm is found in Figure 4.6.

4.4 Experimental Data

Experimental retrieval runs were performed to compare the actual running times of the straightforward and the new p-norm retrieval algorithms. The syntactically generated queries were used in these experiments with the following parameter settings that were found to be best in Chapter 3, namely:

1. *Fox weights* were used to weight the document collection.
2. Clauses were weighted using *sum-weights-modified* method with 75% increase.
3. The **OR** operators were assigned p-value 3.0.
4. The non-uniform method was used to assign p-values to **AND** operators, where the selective **AND**s were assigned p-value 3.0.

These runs will be referred to as the *optimal* runs. An additional set of runs was performed with *Fox weights* and clause weights as above, but with p-value 1.0 assigned to all operators.

Table 4.1 shows the time required for these retrieval runs on a SUN-4. The amount of retrieval time required by the new algorithm is around 40% less than that of the straightforward algorithm in all three collections. From the Table it can also be observed that, not surprisingly, the amount of retrieval times for the

Output the list returned by p_eval(Q) sorted by similarity value.
p_eval (Q) {
    if (Q is a term) {
      result_inv = inverted list for this term;
      disjoint_val = 0;
      return (result_inv, disjoint_val)
    }
    if (Q = **NOT** Q_1) {
      (result_inv$_1$, disjoint_val$_1$) = p_eval(Q_1);
      for each pair (D,sim) in result_inv$_1$
          add (D,1 − sim) to result_inv
      return (result_inv, 1 - disjoint_val$_1$)
    }
    /* $Q = (Q_1, wt_1)OP(Q_2, wt_2)OP \cdots OP(Q_m, wt_m)$ */
    denominator = $wt_1^p + wt_2^p + \cdots + wt_m^p$;
    numerator = 0;
    for $i = 1$ to m
      (result_inv$_i$, disjoint_val$_i$) = p_eval(Q_i);
    /* Docs are processed and removed from lists in sorted order */
    while inverted lists result_inv$_i$ not empty {
      Let D = document with minimum id in result_inv$_i$ lists;
      for $i = 1$ to m
          let sim$_i$ = weight of D in result_inv$_i$ if D is in result_inv$_i$
              or disjoint_val$_i$ otherwise
          if OP = **AND**^{p} add to numerator $(1 - \text{sim}_i)^p wt_i^p$
          if OP = **OR**^{p} add to numerator $\text{sim}_i^p wt_i^p$
      Remove D from inverted lists;
      Add D to result_inv with weight:
          $1 - \sqrt[p]{\frac{\text{numerator}}{\text{denominator}}}$, if OP = **AND**^{p}
          $\sqrt[p]{\frac{\text{numerator}}{\text{denominator}}}$, if OP = **OR**^{p}
    }
    if OP = **AND**^{p}, disjoint_val = $1 - \sqrt[p]{\frac{\sum_{i=1}^{m}(1-\text{disjoint\_val}_i)^p wt_i^p}{\text{denominator}}}$;

    if OP = **OR**^{p}, disjoint_val = $\sqrt[p]{\frac{\sum_{i=1}^{m}\text{disjoint\_val}_i^p wt_i^p}{\text{denominator}}}$;
    return (result_inv, disjoint_val)
}

Figure 4.6: New Algorithm

Table 4.1: Straightforward Algorithm vs. New Algorithm

Collection	P-values	Straightforward Algorithm	New Algorithm	% Improvement
MEDLARS	Optimal	10.8 sec.	6.6 sec.	38.9%
	p=1	6.9 sec.	4.2 sec.	39.1%
CACM	Optimal	48.9 sec.	28.6 sec.	41.5%
	p=1	32.7 sec.	19.9 sec.	39.1%
INSPEC	Optimal	669.5 sec.	343.9 sec.	48.6%
	p=1	465.8 sec.	239.0 sec.	48.7%

runs with p-values of 1.0 are much shorter than the retrieval times of the *optimal* runs, which need to perform many floating point computations for the required exponentiations.

4.5 Asymptotic Analysis

This section presents an asymptotic analysis of the computational time of the straightforward and the newly proposed algorithm. Even though the experimental data has already demonstrated that the new algorithm is faster for the SMART collections, an asymptotic analysis of these algorithms is important in order to know what to expect when very large collections are used. The following two lemmas formally describe the computational requirements of the two algorithms.

Lemma 1 *The time complexity of the straightforward algorithm is*

$$O(|Nondisj(Q)| \cdot \#\ of\ tree\ nodes)$$

where Nondisj(Q) is the set of documents having a term in common with query Q.

Proof:

Reading in the inverted lists for the query terms takes

$$O(|\text{Nondisj}(Q)| \cdot \#\text{ of terms of } Q)$$

since each list can be at most $|\text{Nondisj}(Q)|$ long.
The while loop is executed $|\text{Nondisj}(Q)|$ times, and the cost for each iteration of the loop is:

1. $O(\#$ of terms in $Q)$ to find the minimum document id.
2. $O(\#$ of tree nodes) for doing p_eval since this routine calls itself recursively these many times.
3. $O(\#$ of terms in $Q)$ to remove document from the inverted lists

This totals $O(\#$ of terms in $Q + \#$ of tree nodes), which is equal to

$$O(\#\text{ of tree nodes}).$$

So, the total cost for the while loop is

$$O(|\text{Nondisj}(Q)| \cdot \#\text{ of tree nodes})$$

Therefore, the entire algorithm has time complexity:

$$O(|\text{Nondisj}(Q)| \cdot \#\text{ of tree nodes})$$

QED

Lemma 2 *The time complexity of the new algorithm is*

$$O(\sum_{node_i \in TREE\_NODES} b_i \cdot |Nondisj(node_i)|)$$

where:

$$
\begin{aligned}
TREE\_NODES &= \textit{set of tree nodes of the query tree,} \\
Nondisj(node_i) &= \textit{set of documents with at least one term from the} \\
&\quad \textit{subtree with root } node_i\textit{, and} \\
b_i &= \textit{number of children of } node_i \textit{ (i.e. \# of operands),} \\
&\quad \textit{or 1 if } node_i \textit{ is a leaf.}
\end{aligned}
$$

Proof: (By induction on the number of tree nodes)

Basis Let number of tree nodes = 1.

Queries represented by only one tree node consist of just a query term.

$$
\begin{aligned}
\text{So, the time to run p\_eval}(Q) &= \text{time to read inverted list for the term} \\
&= |\text{Nondisj}(term)|
\end{aligned}
$$

Therefore, the lemma holds.

Induction Suppose we know that the lemma holds for trees with $< c$ nodes.

Let the tree for query Q have c nodes.

Suppose that $Q = (Q_1 \textbf{ OR } Q_2 \textbf{ OR } \cdots \textbf{ OR } Q_m)$.

The time to process Q is:

1. time to process each of $Q_1 \ldots Q_m$ plus
2. $O(m)$ for computing the denominator plus
3. time to execute the while loop, which can be calculated as follows:
   - the while loop is executed $|\text{Nondisj}(root)|$ times
   - the cost for each time around the loop is:
     (a) $O(m)$ for finding minimum document id
     (b) $O(m)$ for computing numerator

(c) $O(m)$ for removing document from inverted lists

- So, the total cost for the while loop is

$$O(m \cdot |\text{Nondisj}(root)|)$$

Since each Q_i has $< c$ nodes, we know that lemma holds for them by the inductive hypothesis. Therefore, the time to process Q is:

$$O(\sum_{node_i \in \bigcup_{j=1}^{m} TREE\_NODES \text{ of } Q_j} b_i \cdot |\text{Nondisj}(node_i)|) + O(m \cdot \text{Nondisj}(root))$$

Note that the $TREE\_NODES$ of Q are:

$$TREE\_NODES_Q = \bigcup_{j=1}^{m} TREE\_NODES \text{ of } Q_j \cup \{root\}$$

Since m is the number of children of the root, the above time complexity becomes:

$$O(\sum_{node_i \in TREE\_NODES_Q} b_i \cdot |\text{Nondisj}(node_i)|)$$

So, the lemma holds for this case. For the cases of $Q =$ **NOT** Q_1 and $Q = Q_1$ **AND** $\cdots$ **AND** Q_m similar arguments hold.

QED

The savings of the new algorithm comes from not having to process each document at every tree node. The larger the difference between |Nondisj(Q)| and a typical value of |Nondisj($node_i$)|, the greater the speed-up that this algorithm will yield. Note that, in the worst case, |Nondisj(Q)| = |Nondisj($node_i$)| for all i, and the time complexity from Lemma 2 becomes

$$O(|\text{Nondisj}(Q)| \cdot \sum_{node_i \in TREE\_NODES_Q} b_i),$$

which is equal to

$$O(|\text{Nondisj}(Q)| \cdot \#\text{ of tree nodes})$$

as $\sum_{node_i} b_i$ counts each tree node at most twice. So there is no speed-up if we make the unrealistic assumption that all the terms in the query always co-occur.

It should be noted that the longer the query, the more likelihood there is of having a large difference between |Nondisj(Q)| and the average value of |Nondisj($node_i$)|. For example, the set of documents having a term in common with a query of 20 terms is much greater than the set of documents having a given term, and the more terms the query has, the greater the expected difference between the two sets. In a relevance feedback setting[3] in which the query length can be fairly large, the new algorithm would prove to be even more valuable.

An important property that must be pointed out is that the running time of the new algorithm also grows linearly with the size of the collection. For instance, if the collection grows to be twice its original size, typically so will the sets Nondisj($node_i$), thereby doubling the total running time.

The memory requirements of the new algorithm can be very large, and thus needs to be dealt with carefully. The reason for the large memory requirement is:

1. The inverted lists can get very long. Note that when processing a node associated with an operator, the lists of the children are merged. Thus, the length of the lists continually grows as they are propagated up the tree.

2. When merging the lists returned by the children of an operator, all of the lists need to be kept in memory at the same time. Thus, the broader the query the more memory that is required.

Since the expected length of the inverted lists is proportional to the collection size, an unreasonable amount of memory would be required by this algorithm in very large collections. However, this drawback can easily be handled by processing small portions of the collection and then combining the results. For example,

[3]Relevance Feedback is a process in which queries are reformulated for subsequent searches based on the user's evaluation of the previously retrieved documents [Sal89].

suppose that the collection contained 1,000,000 documents, and suppose that given the amount of memory available, the largest acceptable set of documents that can be handled is 200,000. The document collection can then be split into 5 sets, each containing 200,000 documents:

S_1 = consisting of documents 1 through 200,000

S_2 = consisting of documents 200,001 through 400,000

S_3 = consisting of documents 400,001 through 600,000

S_4 = consisting of documents 600,001 through 800,000

S_5 = consisting of documents 800,001 through 1,000,000

A set of inverted lists would then be created for each set S_i. To process a query, the following process is performed:

- For each i, run the new algorithm on set S_i, yielding a list L_i which is sorted by similarity value.
- Merge lists L_1, L_2, L_3, L_4, and L_5. This can be done in linear time.

The expected memory requirements for this process would be one fifth of the requirements for running the new algorithm on the entire collection of 1,000,000 documents.

Determining what the largest acceptable set of documents that can be processed is system dependent, and is based on:

- The expected breadth of the query trees.
- The expected length of the inverted lists.

4.6 Boolean vs. P-norm Retrieval

In order to determine the feasibility of the new p-norm retrieval algorithm, its running time is compared against that of a pure Boolean retrieval. The algorithm for performing Boolean retrieval is described formally in Figure 4.7. This

Table 4.2: New Algorithm vs. Boolean Algorithm

Collection	P-values	New Algorithm	Boolean Algorithm
MEDLARS	Optimal	6.6 sec.	1.9 sec.
	p=1	4.2 sec.	
CACM	Optimal	28.6 sec.	7.3 sec.
	p=1	19.9 sec.	
INSPEC	Optimal	343.9 sec.	98.1 sec.
	p=1	239.0 sec.	

algorithm was described in Chapter 1 as the processing of the operators **AND**, **OR**, and **NOT** as set intersection, set union, and set difference, respectively.

Table 4.2 shows the difference in retrieval time between the two systems. Pure Boolean retrieval can be seen to be much faster than p-norm retrieval. Even when all p-values are 1.0, pure Boolean retrieval is much faster. Thus, the floating point computations only partially account for the greater computational time requirements of the p-norm model. The other factor contributing to the p-norm model's extra time requirements is the length of the lists that are manipulated by the algorithm. Note that whereas the lists of the p-norm algorithm always grow as they are propagated up the tree, the lists of the pure Boolean algorithm shrink whenever an **AND** operator is processed due to the list intersection that is performed.

4.7 Conclusion

Although the new p-norm retrieval algorithm is considerably faster than the straightforward algorithm, it is still much slower than pure Boolean retrieval. The two factors contributing to the extra computational time are the floating point computations and the size of the lists that need to be manipulated in the

```
Let Q be a Boolean query with terms q_1, ... q_n.

for i = 1 to n
    get inv_i = inverted list for term q_i
Let union_list = union of inverted lists inv_i for i = 1, ..., n
result_inv_list = bool_eval(Q)
return result_inv_list

bool_eval(Q)
{
    if (Q is a term) {
      result_inv = inverted list for this term
      return( result_inv )
    }
    if (Q = NOT Q_1) {
      inv_1 = bool_eval(Q_1)
      for each document D in union_list
          if D ∉ inv_1 add it to result_inv
      return( result_inv )
    }
    if (Q = Q_1 AND ··· AND Q_m) {
      for (i = 1, ..., m)
          inv_i = bool_eval(Q_i)
      result_inv = ⋂_{i=1}^{m} inv_i
      return( result_inv )
    }
    if (Q = Q_1 OR ··· OR Q_m) {
      for (i = 1, ..., m)
          inv_i = bool_eval(Q_i)
      result_inv = ⋃_{i=1}^{m} inv_i
      return( result_inv )
    }
}
```

Figure 4.7: Pure Boolean Algorithm

p-norm retrieval process. Thus, the goal of the next chapter is to reduce the p-norm retrieval time further by reducing the time that these two factors require in an attempt to make the p-norm retrieval processing time comparable to that of pure Boolean retrieval.

Chapter 5

Improving P-norm Retrieval Efficiency through Approximations

In Chapter 4, a new algorithm was presented for performing p-norm retrieval in less time than the straightforward algorithm formerly implemented in the SMART system. However, the new algorithm was also seen to be slower than the pure Boolean retrieval algorithm. Because most commercial information retrieval systems use pure Boolean retrieval, it seems important to try to reduce the time for p-norm retrieval to be comparable to that for pure Boolean retrieval. This chapter introduces a new information retrieval model, the *infinity-one model*, with a much more efficient retrieval process, and with a retrieval effectiveness approximating that of the p-norm model. List pruning methods for further efficiency improvements are also presented.

5.1 Infinity-One Model

The p-norm operator OR^p varies from a neutral operator $\mathrm{OR}^{1.0}$ to a pure disjunction OR^{∞} as p grows from 1 to ∞. Hence it seems possible to approximate

$\mathbf{OR}^p$ by a linear combination of $\mathbf{OR}^{1.0}$ and $\mathbf{OR}^{\infty}$. This possibility is the basis for the *infinity-one model.*

The operators of the infinity-one model, $\mathbf{OR}^{\alpha}_{\infty,1}$ and $\mathbf{AND}^{\alpha}_{\infty,1}$, are defined in terms of p-norm operators as follows:

$$\begin{aligned} Sim(D, Q_1\ \mathbf{OR}^{\alpha}_{\infty,1}\ Q_2) &= \alpha \cdot Sim(D, Q_1 \mathbf{OR}^{1.0} Q_2) + \\ & \quad (1-\alpha) \cdot Sim(D, Q_1 \mathbf{OR}^{\infty} Q_2) \end{aligned}$$

and

$$\begin{aligned} Sim(D, Q_1\ \mathbf{AND}^{\alpha}_{\infty,1}\ Q_2) &= \alpha \cdot Sim(D, Q_1 \mathbf{AND}^{1.0} Q_2) + \\ & \quad (1-\alpha) \cdot Sim(D, Q_1 \mathbf{AND}^{\infty} Q_2) \end{aligned}$$

Because the p-norm operators $\mathbf{OR}^{\infty}$, $\mathbf{OR}^{1.0}$ (= $\mathbf{AND}^{1.0}$), and $\mathbf{AND}^{\infty}$ can all be evaluated without exponentiation, it follows that infinity-one operators are much less expensive to evaluate than general p-norm operators, which make heavy use of exponentiation.

The parameter α, which varies between 0 and 1.0, specifies how strictly an infinity-one operator is to be evaluated. Clearly an α-value of 1.0 corresponds to a p-value of 1.0, and an α-value of 0 corresponds to a p-value of ∞. In general, as the p-value increases from 1.0 to ∞, the corresponding α-value decreases from 1.0 to 0. Figure 5.1 shows various level curves of $\mathbf{OR}^{\alpha}_{\infty,1}$ passing through a point (x, x). From the figure, it can be seen that all the level curves for the infinity-one model are made up of two line segments meeting at the diagonal, i.e. the line $x = y$. The angle between the two line segments is 90° when $\alpha = 0$, and it increases to 180° when $\alpha = 1$. The strictness with which the operator is evaluated is determined by the slope of these line segments.

5.1.1 Determining α from a Given P-value

Due to the curved nature of the p-norm level curves for p-values other than 1 and ∞, the infinity-one model can only approximate these level curves. Two methods,

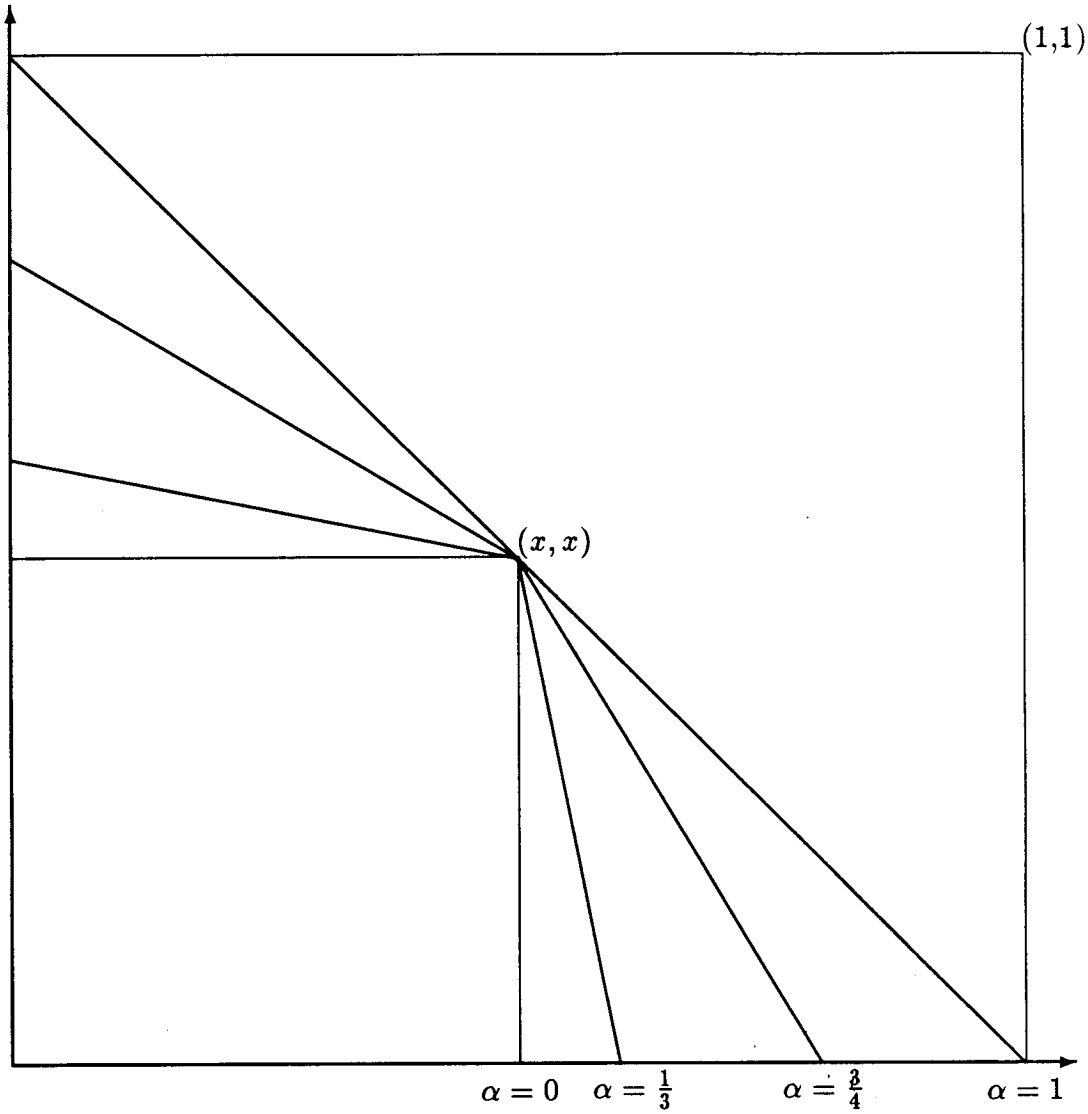

Figure 5.1: Level Curves of $\mathbf{OR}^{\alpha}_{\infty,1}$ Passing Through a Point (x, x)

the *half method* and the *quarter method*, are given below for determining a value for α whose corresponding level curves approximate those of a given p-value. In the next section the effectiveness obtained by using the α values produced by these methods is compared with the effectiveness of the p-norm model.

The *half method* approximates the level curve of $\mathbf{OR}^p$ passing through a point (s, s) with two line segments meeting at the point (s, s).[1] The line segment above the diagonal intersects the $\mathbf{OR}^p$ level curve at point (s, s) and at the point on the curve corresponding to the x-coordinate $\frac{s}{2}$. The reflection of this line segment with respect to the diagonal is used to approximate the lower portion of the level curve. Figure 5.2 shows what this looks like.

The value of the y-coordinate, when $x = \frac{s}{2}$ can be determined by solving the following equation:

$$s = \sqrt[p]{\frac{(\frac{s}{2})^p + y^p}{2}}.$$

In other words, y is the value for which the point ($\frac{s}{2}$,y) obtains similarity s. The following simplifying assumptions have been made here:

- the weights are all 1.0, and
- the number of operands in the clause is 2.

Through straightforward algebraic manipulations, the above equation becomes

$$y = s \cdot \sqrt[p]{2 - \frac{1}{2^p}}.$$

It is easy to show that

$$\alpha \cdot Sim\left((s, s), x\ \mathbf{OR}^{1.0}\ y\right) + (1 - \alpha) \cdot Sim\left((s, s), x\ \mathbf{OR}^{\infty}\ y\right) = s$$

So, in order for the points (s, s) and $\left(\frac{s}{2}, s \cdot \sqrt[p]{2 - \frac{1}{2^p}}\right)$ to lie in the same level-curve in the infinity-one model, one needs to determine a value of α for which

[1] Such a level curve corresponds to similarity s because of the idempotence property which is satisfied by the p-norm model. Such properties are discussed in Chapter 6.

the similarity at point $\left(\frac{s}{2}, s \cdot \sqrt[p]{2-\frac{1}{2^p}}\right)$ is also s. In other words, α must satisfy the following:

$$\begin{aligned} s &= \alpha \cdot Sim\left(\left(\frac{s}{2}, s \cdot \sqrt[p]{2-\frac{1}{2^p}}\right), x \ \mathbf{OR}^{1.0} \ y\right) \\ &+ (1-\alpha) \cdot Sim\left(\left(\frac{s}{2}, s \cdot \sqrt[p]{2-\frac{1}{2^p}}\right), x \ \mathbf{OR}^{\infty} \ y\right) \\ &= \alpha \cdot avg\left(\frac{s}{2}, s \cdot \sqrt[p]{2-\frac{1}{2^p}}\right) + (1-\alpha) \cdot max\left(\frac{s}{2}, s \cdot \sqrt[p]{2-\frac{1}{2^p}}\right) \end{aligned}$$

By dividing through by s, the above equation becomes

$$1 = \alpha \cdot avg\left(\frac{1}{2}, \sqrt[p]{2-\frac{1}{2^p}}\right) + (1-\alpha) \cdot max\left(\frac{1}{2}, \sqrt[p]{2-\frac{1}{2^p}}\right) \tag{5.1}$$

After solving for α, one gets

$$\alpha = \frac{2 - 2\sqrt[p]{2-\frac{1}{2^p}}}{\frac{1}{2} - \sqrt[p]{2-\frac{1}{2^p}}}.$$

Note that α does not depend on s. So all the $\mathbf{OR}^p$ level curves are approximated with the same α.

The same value for α approximates the level curves of $\mathbf{AND}^p$ in the same manner as the curves of $\mathbf{OR}^p$, for consider the level curve of $\mathbf{AND}^p$ corresponding to similarity s. Again, this level curve passes through the point (s, s). The line segment used to approximate the $\mathbf{AND}^p$ level curve intersects the curve at point (s, s) and at the point whose x-coordinate is halfway between s and 1, i.e. $\frac{s}{2} + \frac{1}{2}$. The reflection approximates the other portion of the level curve. See Figure 5.3.

The *quarter method* is identical to the *half method*, except that the line segment used in the approximation intersects the $\mathbf{OR}^p$ level curve at the point $\frac{s}{4}$ instead of $\frac{s}{2}$. See Figure 5.4.

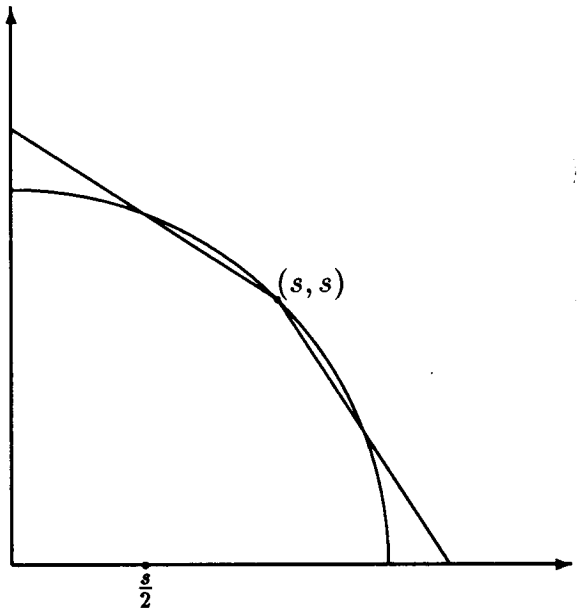

Figure 5.2: Approximating the $\mathbf{OR}^{2.0}$ Level Curve with the Half Method

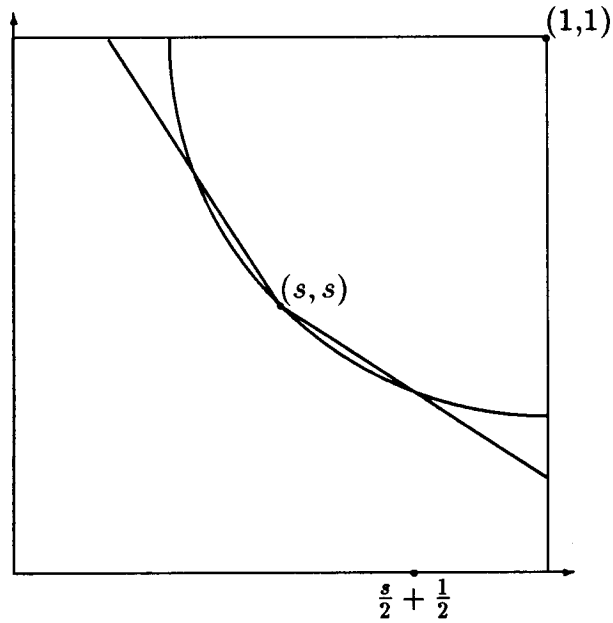

Figure 5.3: Approximating the $\mathbf{AND}^{2.0}$ Level Curve with Half Method

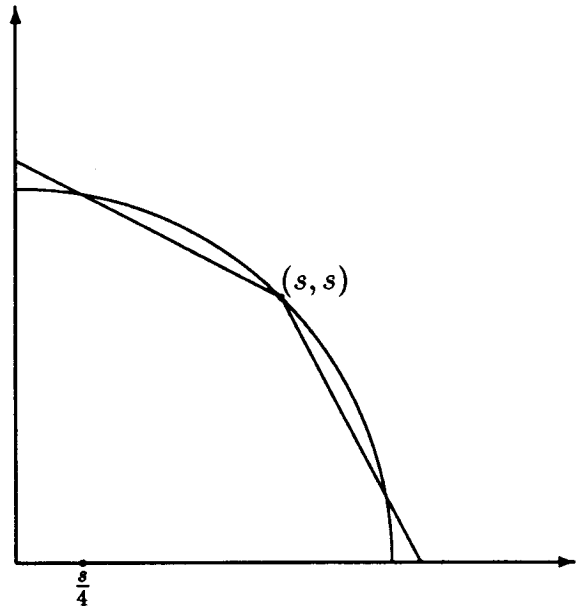

Figure 5.4: Approximating the $OR^{2.0}$ Level Curve with the Quarter Method

5.1.2 Effectiveness & Efficiency

The syntactically generated p-norm queries with the parameter settings that were found best in Chapter 3 are used in the comparisons between the infinity-one model and the p-norm model. The only p-values used in these queries were 1.0 and 3.0. Clearly, the α corresponding to p-value 1.0 is 1.0. The values of α for p-value 3.0 produced by the *half method* and the *quarter method* are .64 and .51, respectively. Table 5.1 shows the effectiveness obtained by approximating the level curves for p-values 1.0 and 3.0 with these values of α. The approximation produced by the *quarter method* is slightly better than the *half method*. However, some deterioration in precision is present as compared with the p-norm model — the largest being in the MEDLARS where there is a decline in the precision of the top 10 documents of 4.5%.

Major assumptions were made in the *half method* and *quarter method*, namely that:

1. the number of operands was always 2, and

2. the weights of the operands were always 1.0.

Table 5.1: Infinity-One Model vs. P-norm Model

Collection		P-norm	Half method	% change	Quarter method	% change
MEDLARS	Avg. Prec.	.6089	.5939	-2.5%	.5938	-2.5%
	Prec. Top 10	.6733	.6467	-4.0%	.6433	-4.5%
CACM	Avg. Prec.	.3381	.3355	-0.8%	.3356	-0.7%
	Prec. Top 10	.3412	.3294	-3.5%	.3314	-2.9%
INSPEC	Avg. Prec.	.2731	.2696	-1.3%	.2704	-1.0%
	Prec. Top 10	.4260	.4182	-1.8%	.4208	-1.2%

These assumptions made it possible to have the values of α pre-computed for all allowable p-values before the retrieval is performed. In this manner, all of the complex computations required to determine α can be done ahead of time. However, there are often more operands than 2, and the weights of the operands certainly vary a great deal. Thus, the values of α produced by these methods can be quite different from the values that would be produced if the actual weights and number of operands were taken into account.

Another approach to approximating the p-norm effectiveness with the infinity-one model is to find values of α that generally do well in approximating $\mathbf{AND}^{3.0}$ and $\mathbf{OR}^{3.0}$ over all queries. Table 5.2 shows the retrieval output produced by a set of runs in the infinity-one model with varying values of α for approximating these operators with p-value of 3.0, where the top number of each entry represents average precision and the lower number represents precision in the top 10 documents retrieved. It can be seen that in the three collections, setting $\alpha = .75$ to approximate the $\mathbf{AND}^{3.0}$ operators and setting $\alpha = .25$ to approximate the $\mathbf{OR}^{3.0}$ operators comes very close to producing the exact average precision and top 10 precision of the p-norm model.

In table 5.3, it can be seen that the infinity-one model retrieval with $\alpha = .75$

Table 5.2: Varying α in the Infinity-One Model

Method	MED	% Change	CACM	% Change	INSPEC	% Change
P-norm	.6089	-	.3381	-	.2731	-
	.6733	-	.3412	-	.4260	-
$\alpha = .75$ for $\mathbf{AND}^{3.0}$	.5979	-1.8%	.3361	-0.6%	.2719	-0.4%
$\alpha = .50$ for $\mathbf{OR}^{3.0}$	.6533	-3.0%	.3294	-3.5%	.4221	-0.0%
$\alpha = .75$ for $\mathbf{AND}^{3.0}$	.6038	-0.8%	.3402	+0.6%	.2748	+0.6%
$\alpha = .25$ for $\mathbf{OR}^{3.0}$	.6700	-0.5%	.3412	+0.0%	.4247	-0.3%
$\alpha = .50$ for $\mathbf{AND}^{3.0}$	.5935	-2.5%	.3357	-0.7%	.2707	-0.9%
$\alpha = .50$ for $\mathbf{OR}^{3.0}$	.6433	-4.5%	.3314	-2.9%	.4195	-1.5%
$\alpha = .25$ for $\mathbf{AND}^{3.0}$	.5932	-2.6%	.3377	-0.1%	.2707	-0.9%
$\alpha = .25$ for $\mathbf{OR}^{3.0}$	.6500	-3.5%	.3353	-1.7%	.4169	-2.1%

approximating $\mathbf{AND}^{3.0}$ and $\alpha = .25$ approximating $\mathbf{OR}^{3.0}$ is much faster than the p-norm retrieval with the fast algorithm presented in Chapter 4. The algorithm used in performing the infinity-one model's retrieval is exactly like the fast p-norm algorithm, with the exception of the formula that is used to combine the inverted lists that are returned by the recursive processing of the operands. The retrieval time with the infinity-one model is about 60% less than that of the p-norm model, but it is still significantly slower than the Boolean retrieval.

5.2 List Pruning

A user of an information retrieval system is often interested in obtaining only a small number of references about the topic of interest. The question considered in this section is whether retrieval efficiency can be improved in a setting where only a small number of documents is desired. The approach taken in the two

Table 5.3: Infinity-one Efficiency vs. P-norm and Boolean Efficiency

Collection	P-norm Retrieval	Boolean Retrieval	Infinity-One Retrieval
MEDLARS	6.6 sec.	1.9 sec.	2.6 sec.
CACM	28.6 sec.	7.3 sec.	11.6 sec.
INSPEC	343.9 sec.	98.1 sec.	144.4 sec.

methods presented below is to prune the the lists that are propagated up the tree by omitting those documents that are not likely to be ranked in the top 10.

5.2.1 Constant Threshold Pruning

Many of the documents that are retrieved in response to a query have a very small similarity value. This is because a retrieved document is typically indexed only by a small percentage of the query terms. In fact, many of the documents retrieved have only one term in common with the query. These documents are not likely to be ranked in the top 10. Thus, the ability to recognize these documents and omit them when processing a query node can reduce the amount of required processing time significantly.

Constant threshold pruning is a simple modification of the new algorithm presented in Chapter 4. The modification simply consists of pruning from the list returned by each query node all documents whose similarities fail to exceed some threshold, say .01.

The assumption that is made in the constant threshold method is that a document needs to do very well in some component of the query if it is to be ranked in the top. Note that a document eliminated from a node's list can still be reintroduced by some other node. However, once a document is omitted from a node's list, the similarity between the document and the subquery associated

with the node is taken to be the disjoint-value of the node.[2]

5.2.2 Variable Threshold Pruning

Variable threshold pruning also requires that all documents placed in the list returned by a query node exceed a certain threshold. The threshold used starts out at some constant value, as in constant threshold pruning. However, after a sizable number of documents, say 300, have been inserted into the list, the threshold is updated to be the average similarity of the documents placed in the list so far. As more and more documents are processed, a continuously better estimate of the typical similarity between a matching document and the subquery associated with the node can be obtained. Thus, a fairly low initial threshold of .01 is used, and later updated, thereby requiring subsequent documents that are processed to be as good as the typical documents already on the list. The intuition behind this more stringent requirement is that a document ranked in the top 10 generally has an above average similarity with some component of the query.

5.2.3 Effectiveness and Efficiency of List Pruning Methods

Table 5.4 shows the retrieval times required by infinity-one retrieval with and without list pruning, and table 5.5 shows the effectiveness of infinity-one retrieval with and without list pruning. The precision of the top 10 documents retrieved is essentially the same whether or not list pruning is used. However, there is a large variation in efficiency between these runs. The runs with variable threshold pruning were the fastest. The retrieval times for this method ranged from 11% faster to 51% faster than for retrievals without list pruning. Variable threshold pruning is also significantly faster than constant threshold pruning. Since many

[2]As a result, in the presence of NOTs, the pruning must be modified slightly. Such details are ignored here for simplicity.

Table 5.4: Efficiency of List Pruning Methods

Collection	Infinity-One Retrieval	Constant Threshold Pruning	% Change	Variable Threshold Pruning	% Change
MEDLARS	2.6 sec.	2.3 sec.	-11.5%	2.3 sec.	-11.5%
CACM	11.6 sec.	9.5 sec.	-18.1%	7.7 sec.	-33.6%
INSPEC	144.4 sec.	104.3 sec.	-27.8%	70.6 sec.	-51.1%

Table 5.5: Effectiveness of List Pruning Methods

Collection		Infinity-One Retrieval	Constant Threshold Pruning	% Change	Variable Threshold Pruning	% Change
MEDLARS	Avg	.6038	.6040	+0.0%	.6015	-0.4%
	Top 10	.6700	.6700	+0.0%	.6733	+0.5%
CACM	Avg	.3402	.3403	+0.0%	.3361	-1.2%
	Top 10	.3412	.3431	+0.6%	.3373	-1.1%
INSPEC	Avg	.2748	.2741	-0.3%	.2599	-5.4%
	Top 10	.4247	.4234	-0.3%	.4221	-0.6%

documents are discarded by list pruning, it is not surprising that the average precision at recall points of .25, .50, and .75 decreases. However, this is of no importance in a setting in which only the top 10 documents are desired.

When variable threshold pruning infinity-one retrievals are compared against pure Boolean retrievals, one finds that they are comparable in speed and can sometimes be even faster. Table 5.6 shows this comparison.

Table 5.6: Infinity-One with Variable Threshold Pruning vs. Pure Boolean

Collection	Boolean Retrieval	Infinity-One Retrieval
MEDLARS	1.9 sec.	2.3 sec.
CACM	7.3 sec.	7.7 sec.
INSPEC	98.1 sec.	70.6 sec.

5.3 Summary

This chapter has shown that the infinity-one model is approximately as effective as the p-norm model, but much less expensive in retrieval time. Furthermore, by using variable threshold pruning in settings where only a small number of documents are desired in response to a query, the retrieval time has been reduced to an amount comparable to that of pure Boolean retrieval without lowering the precision of the top ranked documents.

Chapter 6

Waller-Kraft Wish List and the P-norm Model

There has been a great deal of debate about the desirable properties for an extended Boolean information retrieval model. This chapter concentrates on the well-known criteria in the *Waller-Kraft wish list* [WK79] and evaluates the p-norm model with respect to these criteria.

It is shown that the p-norm model satisfies most of the criteria on the wish list. Of the criteria that are not satisfied, it is argued that only the Boolean self-consistency property is actually desirable. The failure of Boolean self-consistency is shown to be unavoidable by formalizing the notion of *weak* Boolean operators and then demonstrating that no system having such weak operators can satisfy Boolean self-consistency. In particular, the p-norm model is shown to satisfy all the Boolean algebra properties that can be satisfied by any system containing weak as well as strict Boolean operators.

6.1 Formalization of Extended Boolean Systems

The descriptions of the Waller-Kraft wish list found in the literature typically formalize the effect of query term weights (and query clause weights) in extended Boolean systems in terms of *partial retrieval status values* (partial RSV)[1] [BK81a]. The similarity between a query and a document is sometimes called the *retrieval status value* (or RSV). When a query consists of several components, such as

$$\text{(A AND B) OR (C AND D)},$$

one may consider the contribution made by each of the components (i.e. subqueries) to the final similarity value between a document and the entire query. For example, one could analyze how much the clause (A AND B) contributes to the similarity for the above query. The amount of contribution of a subquery is referred to as its partial RSV. In the presence of weighted subqueries, the partial RSV is described in [BK81a,CK87,CK89] as a function

$$g : [0,1] \times [0,1] \rightarrow [0,1].$$

The partial RSV for a document d based only on term t can be described as:

$$g(F(d,t), a(t)),$$

where

$$F : D \times T \rightarrow [0,1]$$

$$a : T \rightarrow [0,1].$$

D is the set of documents, T is the set of index terms, F assigns weights to index terms in a document, and a assigns weights to query terms. The partial RSV function g typically used in weighted Boolean systems is scalar multiplication.

[1] Also referred to as *generalized retrieval values* in [CK87,CK89]

This formalization assumes that the behavior of all weighted Boolean systems can be described in terms of the partial RSV function g. This assumption has led to claims that the term properties and separability condition of the wish list, described in the next section, cannot be simultaneously satisfied [BK81b,BK81a]. However, not all weighted Boolean systems can be described in terms of this type of partial RSV function. In particular, the p-norm model cannot, as will be seen below.

First, let us suggest an alternative formalization of an extended Boolean retrieval system that clarifies the partial RSV approach above.

In an extended Boolean system, the set of possible similarity values between queries and documents is the interval [0,1]. So a query is a map[2]

$$D \rightarrow [0,1].$$

The set Q of queries consists of primitive queries and compound queries. Primitive queries are simply terms; i.e., a term t is a map

$$t : D \rightarrow [0,1].$$

How are compound queries defined? The similarity space [0,1] comes equipped with a variety of operations, and each such operation induces an operation on Q. To see how this works, consider for example the operation

$$\mathbf{OR}{:}[0,1] \times [0,1] \rightarrow [0,1]$$

given by

$$\mathbf{OR}(x,y) = max(x,y).$$

This induces an **OR** operation on queries

$$\mathbf{OR}{:}Q \times Q \rightarrow Q$$

by pointwise application:

[2]Since queries are viewed as maps, the notation $Q_1(d)$ will be used in place of the usual $Sim(d,Q_1)$ throughout this chapter.

$$\mathbf{OR}(Q_1, Q_2)(d) = \mathbf{OR}(Q_1(d), Q_2(d)).$$

Similarly, there are induced **AND** and **NOT** operations on Q.

In summary, the set of queries is generated from

1. terms, and

2. operations induced from operations on [0,1].

The partial RSV approach to query weights can now be described as the use of an operation $wt : [0,1] \times [0,1] \rightarrow [0,1]$ to induce an operation

$$wt : Q \times [0,1] \rightarrow Q.$$

Thus $wt(Q_1, a)(d) = wt(Q_1(d), a)$.

This approach of incorporating query weights into an extended Boolean system, though intuitively appealing, can be seen to run into problems. Consider

$$wt(t_1, 0) \ \mathbf{AND} \ t_2$$
$$wt(t_1, 0) \ \mathbf{OR} \ t_2,$$

where **AND** and **OR** are *min* and *max*, respectively. One expects both of these queries to be equivalent to t_2. Since term t_1 is given weight 0, it should have no effect on the retrieval output. So $wt(t_1, 0)$ must be a constant function returning, say c. So

$$(wt(t_1, 0) \ \mathbf{AND} \ t_2)(d) = min(c, t_2(d))$$

and

$$(wt(t_1, 0) \ \mathbf{OR} \ t_2)(d) = max(c, t_2(d)).$$

Clearly no choice of c gives

$$min(c, t_2(d)) = max(c, t_2(d)) = t_2(d)$$

in general, thus there is no way for query weights to work properly if this approach is used.

In constrast, the approach taken by the p-norm model is to treat query weights as representatives of the relative importance of the operands in a given clause. Thus, the weights are considered to be associated with the operators rather than the subqueries themselves. Formally, this approach can be described as the use of *weighted* **OR** and **AND** operations

$$\textbf{OR}: \ ([0,1] \times [0,1]) \times ([0,1] \times [0,1]) \rightarrow [0,1]$$
$$\textbf{AND}: \ ([0,1] \times [0,1]) \times ([0,1] \times [0,1]) \rightarrow [0,1]$$

to induce operations

$$\textbf{OR}: \ (Q \times [0,1]) \times (Q \times [0,1]) \rightarrow Q$$
$$\textbf{AND}: \ (Q \times [0,1]) \times (Q \times [0,1]) \rightarrow Q$$

Using the terminology of the literature, this approach can be described as one in which given query

$$(Q_1, w_1) \ \textbf{OR} \ (Q_2, w_2)$$

the weights w_1 and w_2 are not used in computing the partial RSV of Q_1 and Q_2. Instead, these weights are used in computing the partial RSV of the entire **OR** clause. With this handling of query weights, the weight w_1 in (Q_1, w_1) is considered to have no meaning unless it is a component of a larger query where it represents how important subquery Q_1 is relative to the other components. Thus the operator that combines Q_1 with other components is the only one that needs to handle the weight assigned to Q_1. This formalization is used in the Waller-Kraft wish list description presented below.

6.2 Waller-Kraft Wish List

The Waller-Kraft wish list [WK79] consists of the following six items:

1. separability

2. generalization

3. Boolean self-consistency

4. term properties

5. expression properties

6. merging properties.

Separability is the ability to evaluate a query by first evaluating the individual terms, then the Boolean combinations of the terms, and so forth. In other words, a complex Boolean query can be evaluated from the evaluations of terms and simpler Boolean queries that it is composed of. This is exactly the way that extended Boolean queries are formalized above; separability is the same as the property that operations on queries are all induced from operations on [0,1]. P-norm queries are evaluated in this way, as is demonstrated by the recursive nature of the algorithms presented in Chapter 4 and Chapter 5. The p-norm model thus satisfies the separability property.

The *generalization* property requires that when term weights are restricted to 0 and 1, the evaluation of queries be identical to that of the pure Boolean model. The p-norm model also satisfies this property since, as described in Chapter 1, the Boolean model is included as a special case when weights are restricted to {0,1} and p-values are set to ∞.

Boolean self-consistency is the property that logically equivalent queries yield identical results. The notion of logical equivalence is based on Boolean algebra properties such as commutativity and associativity. Unfortunately, the p-norm model fails to satisfy this property. A detailed discussion of Boolean self-consistency in systems with *weak* Boolean operators is deferred to Section 6.3.

Term properties describe the effect of weights on query terms. *Expression properties* describe (among other things) the effect of weights on query clauses. Really there is no reason to separate these two cases; they can be unified into a single list of properties that describe the effect of weights on arbitrary subqueries.

The properties are as follows: If (Q_1, w_1) is an argument to a weighted **AND** or **OR** clause C, then

1. $C(d)$ should increase monotonically with $Q_1(d)$.
2. If $w_1 = 0$, then $C(d)$ should be constant with respect to $Q_1(d)$.
3. As w_1 increases, the rate of growth of $C(d)$ versus $Q_1(d)$ should increase monotonically.

The first property says that the similarity between a document and clause C should be higher for those documents that have a higher similarity with Q_1. In other words, the more similar a document is to Q_1, the greater its similarity to C.

The second property simply states that if an operand's weight is zero, then the similarity between this operand and a document should have no effect on the similarity between the document and the entire clause.

The third property states that given two weights for Q_1, say w_1 and $\widehat{w_1}$, where $w_1 > \widehat{w_1}$, the graph of $C(d)$ versus $Q_1(d)$ for w_1 should increase faster than the graph of $C(d)$ versus $Q_1(d)$ for $\widehat{w_1}$.

The p-norm model satisfies these three properties. For consider the p-norm clause

$$C = (Q_1, w_1) \ \mathbf{AND}^p \ (Q_2, w_2).$$

The similarity between a document d and this clause is

$$C(d) = 1 - \sqrt[p]{\frac{w_1^p(1 - s_1)^p + w_2^p(1 - s_2)^p}{w_1^p + w_2^p}},$$

where $s_1 = Q_1(d)$ and $s_2 = Q_2(d)$.

As s_1 increases, $(1 - s_1)^p$ decreases, thereby making the numerator

$$w_1^p(1 - s_1)^p + w_2^p(1 - s_2)^p$$

decrease. Since the entire p-th root component is negated, $C(d)$ increases.

If $w_1 = 0$, then $C(d)$ becomes

$$1 - \sqrt[p]{\frac{w_2^p(1-s_2)^p}{w_2^p}},$$

which is equivalent to s_2 (or $Q_2(d)$). Thus when $w_1 = 0$, subquery Q_1 has no effect on the final similarity value.

The third property is equivalent to the condition that $\frac{\partial C}{\partial s_1}$ increase monotonically with w_1. That is, given two weights for Q_1, say w_1 and $\widehat{w_1}$, where $w_1 > \widehat{w_1}$, the derivative of $\frac{\partial C}{\partial s_1}$ is greater when Q_1 is given weight w_1 than when Q_1 is given weight $\widehat{w_1}$.

The derivative $\frac{\partial C}{\partial s_1}$ when weight w_1 is assigned is

$$-\left(\frac{1}{p}\right) \cdot \frac{\left[w_1^p(1-s_1)^p + w_2^p(1-s_2)^p\right]^{\frac{1}{p}-1}}{\sqrt[p]{w_1^p + w_2^p}} \cdot pw_1^p(1-s_1)^{p-1}(-1)$$

$$\| $$

$$\frac{\left[w_1^p(1-s_1)^p + w_2^p(1-s_2)^p\right]^{\frac{1}{p}-1} \cdot w_1^p(1-s_1)^{p-1}}{\sqrt[p]{w_1^p + w_2^p}},$$

$$\| $$

$$\left[w_1^p(1-s_1)^p + w_2^p(1-s_2)^p\right]^{\frac{1}{p}-1} \left[w_1^{-p}(1-s_1)^{-p}\right]^{\frac{1}{p}-1} w_1 \left(w_1^p + w_2^p\right)^{-\frac{1}{p}}$$

$$\| $$

$$\left[1 + w_1^{-p}(1-s_1)^{-p}w_2^p(1-s_2)^p\right]^{\frac{1}{p}-1} \left[w_1^{-p}\right]^{-\frac{1}{p}} \left(w_1^p + w_2^p\right)^{-\frac{1}{p}}$$

$$\| $$

$$\left[1 + w_1^{-p}(1-s_1)^{-p}w_2^p(1-s_2)^p\right]^{\frac{1}{p}-1} \left[1 + w_1^{-p}w_2^p\right]^{-\frac{1}{p}} \tag{6.1}$$

Similarly, the derivative $\frac{\partial C}{\partial s_1}$ when weight $\widehat{w_1}$ is assigned is

$$\left[1 + \widehat{w_1}^{-p}(1-s_1)^{-p}w_2^p(1-s_2)^p\right]^{\frac{1}{p}-1} \left[1 + \widehat{w_1}^{-p}w_2^p\right]^{-\frac{1}{p}}. \tag{6.2}$$

It can trivially be shown that formula 6.1 is greater than formula 6.2 for the case when $p = 1$. The following two claims show that this is also true when $p > 1$.

Claim 1: $\left[1 + w_1^{-p}(1-s_1)^{-p}w_2^p(1-s_2)^p\right]^{\frac{1}{p}-1} > \left[1 + \widehat{w_1}^{-p}(1-s_1)^{-p}w_2^p(1-s_2)^p\right]^{\frac{1}{p}-1}$ for $p > 1$.

Proof: By raising both sides of the inequality to the $\frac{p}{1-p}$ power[3], we get:

$$\left[1 + w_1^{-p}(1-s_1)^{-p}w_2^p(1-s_2)^p\right] < \left[1 + \widehat{w_1}^{-p}(1-s_1)^{-p}w_2^p(1-s_2)^p\right]$$
$$\Updownarrow$$
$$w_1^{-p}(1-s_1)^{-p}w_2^p(1-s_2)^p < \widehat{w_1}^{-p}(1-s_1)^{-p}w_2^p(1-s_2)^p$$
$$\Updownarrow$$
$$w_1^{-p} < \widehat{w_1}^{-p}$$
$$\Updownarrow$$
$$w_1 > \widehat{w_1}$$

QED

Claim 2: $\left[1 + w_1^{-p}w_2^p\right]^{-\frac{1}{p}} > \left[1 + \widehat{w_1}^{-p}w_2^p\right]^{-\frac{1}{p}}$

Proof: By raising both sides of the inequality to the $-p$ power, we get:

$$\left[1 + w_1^{-p}w_2^p\right] < \left[1 + \widehat{w_1}^{-p}w_2^p\right]$$
$$\Updownarrow$$
$$w_1^{-p}w_2^p < \widehat{w_1}^{-p}w_2^p$$
$$\Updownarrow$$
$$w_1^{-p} < \widehat{w_1}^{-p}$$
$$\Updownarrow$$
$$w_1 > \widehat{w_1}$$

QED

A similar argument shows that these three properties hold in **OR** clauses as well.

There are a few expression properties that have not yet been considered. These are:

[3]Note that this is a negative number.

1. If $C = (Q_1, w_1)$ **AND** (term t,wt_2), then $C(d)$ should not increase as wt_2 increases for any document d.

2. If (Q_1, w_1) is an argument to a weighted **AND** or **OR** clause C, then as $Q_1(d)$ increases, the rate of growth of $C(d)$ versus w_1 should increase monotonically.

3. If (**NOT** t,wt_1) is an argument to a weighted **AND** or **OR** clause C, then $C(d)$ should decrease monotonically as $t(d)$ increases.

The explanation given in [WK79, page 241] as a justification of the first property is that

> such a term is connected via an **AND** to restrict the size of the retrieved subset of documents, increasing precision...and decreasing recall.

Increasing the weight wt_2 is taken to mean that the query is more difficult to satisfy, thereby restricting the set of retrieved documents. Since the query is considered more difficult to satisfy, the requirement is made that $C(d)$ should not increase. However, if we consider that the weights express the relative importance of the components of a clause, then increasing weight wt_2 simply makes term t more important relative to the other clause components[4]. Given this interpretation of query weights, it is inappropriate to require that $C(d)$ not increase. For example, consider

$$C_1 = (Q_1, .5) \textbf{ AND } (\text{term } t, .01).$$

Query C_1 states that component Q_1 is much more important than term t. If the weight of term t is increased to 1.0, i.e. $C_2 = (Q_1, .5)$ **AND** (term t,1.0), then

[4]In fact, in the p-norm model the relative values of the weights are what matters, not the absolute values. So, for example

$$(Q_1, .01) \textbf{ AND } (Q_2, .01) \textbf{ AND } (Q_3, .01) = (Q_1, 1.0) \textbf{ AND } (Q_2, 1.0) \textbf{ AND } (Q_3, 1.0).$$

term t is considered to be twice as important as Q_1. Suppose that the document collection contains a document d, such that

$$t(d) = 1 \text{ and } Q_1(d) = 0.$$

Document d does not satisfy query C_1 very well, since it contains only term t, which is not regarded as very important by query C_1. On the other hand, C_2 considers term t to be very important. This implies that $C_2(d)$ is greater than $C_1(d)$. Conversely, consider document d', such that

$$t(d') = 0 \text{ and } Q_1(d') = 1.$$

Document d' does not contain term t, which is an unimportant component of C_1 and a very important component of C_2. This implies that $C_2(d')$ should be less than $C_1(d')$. The effect of increasing weight wt_2 should depend on the similarity values of the individual components of the clause — it can either increase or decrease the evaluation of a document. Since the weight interpretation of the p-norm model is relative importance, this is the effect obtained by increasing wt_2.

The second property can be easily proven by an argument similar to that used to show that the third term property held for the p-norm model.

The evaluation of a negated term is simply one minus the weight of the term in the document. Thus, (**NOT** t)(d) = 1 - $t(d)$. Clearly, as $t(d)$ increases, $(1 - t(d))$ decreases. Therefore, by the first term property, which we have shown is satisfied by the p-norm model, $C(d)$ decreases.

The *merging properties* refer to the behavior of the **AND** and **OR** operators in general when evaluating complex Boolean expressions. The following criteria are given:

1. the evaluation should not increase if an additional expression is connected to the already existing expression via an **AND**

2. the evaluation should not decrease if an additional expression is connected to the already existing expression via an **OR**.

These merging properties can hold only in systems with strict interpretations of the **AND** and **OR** operators. Although these properties may appear reasonable at first glance, they are actually undesirable. For example, suppose that the original expression was Q_1 and that expression Q_2 was connected to Q_1 via an **AND** operator yielding the new query

$$Q = (Q_1, w_1) \textbf{ AND } (Q_2, w_2).$$

Suppose that for document d we have

$$Q_1(d) = 0 \text{ and } Q_2(d) = 1.$$

Since d satisfies Q_2 very well, it seems clear that document d must be better than a document that does not satisfy Q_2 at all. In other words, d must partially satisfy query Q, and therefore, $Q(d)$ should be greater than zero. But if $Q(d) > 0$, then

$$Q(d) > Q_1(d),$$

and this violates the first merging property. A similar argument demonstrates the undesirability of the second merging property. Clearly, the p-norm model does not satisfy these merging properties, but this should be regarded as a good feature rather than a drawback.

6.3 Boolean Algebra and the P-norm Model

In this section, we consider the Boolean algebra properties of extended Boolean information retrieval systems with weak operators. Certain properties, for example associativity, are shown to fail in any such system, and the p-norm model is thus shown to do essentially as well as possible.

In our formalization of extended Boolean systems, all operations on queries are induced from operations on [0,1]. Because of this, our discussion of Boolean

algebra properties can be carried out entirely in terms of the operations on [0,1]. For example, consider commutativity:

$$\textbf{AND is commutative on } Q$$

$$\Updownarrow$$

$$\forall Q_1, Q_2 \; (\; Q_1 \textbf{ AND } Q_2 = Q_2 \textbf{ AND } Q_1)$$

$$\Updownarrow$$

$$\forall Q_1, Q_2 \; \forall d \; ((Q_1 \textbf{ AND } Q_2)(d) = (Q_2 \textbf{ AND } Q_1)(d))$$

$$\Updownarrow$$

$$\forall Q_1, Q_2 \; \forall d \; (\; Q_1(d) \textbf{ AND } Q_2(d) = Q_2(d) \textbf{ AND } Q_1(d) \;)$$

$$\Updownarrow$$

$$\forall s_1, s_2 \; (\; s_1 \textbf{ AND } s_2 = s_2 \textbf{ AND } s_1 \;)$$

$$\Updownarrow$$

$$\textbf{AND is commutative on } [0,1].$$

There are many possible operations

$$\Delta : [0,1] \times [0,1] \rightarrow [0,1]$$

that could be used as **AND** or **OR** operations in an extended Boolean system, but most would be completely unreasonable. At a minimum, Δ should satisfy the following three conditions:

1. $0 \; \Delta \; 0 = 0$ and $1 \; \Delta \; 1 = 1$. (i.e. given query $A \; \Delta \; B$, a document not containing either A or B should be given the lowest possible similarity value. Similarly, a document containing both A and B with the highest possible weight, should be given the highest possible similarity value.)

2. Δ is commutative. (i.e. query $A \; \Delta \; B$ should be equivalent to $B \; \Delta \; A$.)

3. Δ is monotonic on each component. If $x < x'$ then $x \; \Delta \; y \leq x' \; \Delta \; y$, and if $y < y'$ then $x \; \Delta \; y \leq x \; \Delta \; y'$. (i.e. given query $A \; \Delta \; B$ and two documents with the same weight for term B, the document with the higher weight for

term A should be given a similarity value no lower than that of the other document.)

Let any function satisfying these three conditions be referred to as a *combining function.* Examples of combining functions include the *min* and *max* operators used in the fuzzy set model. A *weak* Boolean operator may now be defined as a combining function that is *strictly monotonic* on each component. In other words, the third condition of combining functions is made stronger by requiring that if $x < x'$ then $x \triangle y < x' \triangle y$, and similarly for the second component. Strict monotonicity is a desirable property because it says that when the value of a component increases the value of the entire expression must also increase. This condition overcomes one of the major drawbacks of the fuzzy set model illustrated by the following example.

Example: Consider query $Q = A$ **AND** B and documents

$$d_1 = ((A, .9), (B, .1))$$

$$d_2 = ((A, .2), (B, .1)).$$

The fuzzy set model considers d_1 and d_2 to be equally good since:

$$Q(d_1) = \min(.9,.1) = .1$$

$$Q(d_2) = \min(.2,.1) = .1,$$

which is clearly undesirable. Document d_2 certainly comes closer to satisfying the query than does d_1.

The main result presented in this section simply states that there exists a trade-off between satisfying the Boolean algebra properties and being strictly monotonic. In other words, any extended Boolean system containing weak Boolean operators cannot be a Boolean algebra.

Recall that a Boolean algebra [AO69] is a set of elements together with three operations: $\triangledown$, $\triangle$, and $\sim$, such that for all elements A, B, and C, the following properties hold:

idempotence: $A \bigtriangledown A = A \bigtriangleup A = A$

commutativity: $A \bigtriangledown B = B \bigtriangledown A$ and

$A \bigtriangleup B = B \bigtriangleup A$

associativity: $A \bigtriangledown (B \bigtriangledown C) = (A \bigtriangledown B) \bigtriangledown C$ and

$A \bigtriangleup (B \bigtriangleup C) = (A \bigtriangleup B) \bigtriangleup C$

distributivity: $A \bigtriangledown (B \bigtriangleup C) = (A \bigtriangledown B) \bigtriangleup (A \bigtriangledown C)$ and

$A \bigtriangleup (B \bigtriangledown C) = (A \bigtriangleup B) \bigtriangledown (A \bigtriangleup C)$

absorption: $A \bigtriangledown (A \bigtriangleup B) = A = A \bigtriangleup (A \bigtriangledown B)$

There are 2 special elements 1 and 0 having the properties:

intersection: $1 \bigtriangleup A = A$

union: $0 \bigtriangledown A = A$

The unary operation $\sim$ has the following properties:

involution: $\sim (\sim A) = A$

relation of 1 and 0: $\sim 1 = 0$

De Morgan's rules: $\sim A \bigtriangledown \sim B = \sim (A \bigtriangleup B)$

complementarity: $\sim A \bigtriangleup A = 0$ and

$\sim A \bigtriangledown A = 1$

Lemma 1: A weak operator $\bigtriangledown$ cannot satisfy associativity.

Proof: By the property of strict monotonicity, we have:

$$(1 \bigtriangledown 0) > (0 \bigtriangledown 0),$$

and since $(0 \bigtriangledown 0) = 0$, we have that

$$(1 \bigtriangledown 0) > 0.$$

Thus, through strict monotonicity on the second component:

$$1 \bigtriangledown (1 \bigtriangledown 0) > 1 \bigtriangledown 0.$$

Since $(1 \bigtriangledown 1) = 1$,

$$1 \bigtriangledown (1 \bigtriangledown 0) > (1 \bigtriangledown 1) \bigtriangledown 0.$$

QED

Lemma 2: Absorption does not hold for any two weak operators $\triangle$, and $\bigtriangledown$.

Proof: By the property of strict monotonicity, we have:

$$(1 \bigtriangledown 0) < (1 \bigtriangledown 1).$$

Thus, through strict monotonicity on the second component:

$$\begin{aligned} 1 \triangle (1 \bigtriangledown 0) &< 1 \triangle (1 \bigtriangledown 1) \\ &= 1 \triangle 1 \qquad \text{since } (1 \bigtriangledown 1) = 1 \\ &= 1 \end{aligned}$$

Therefore, $1 \triangle (1 \bigtriangledown 0) < 1$.

QED

Lemma 3: A weak operator $\triangle$ cannot satisfy the intersection property.

Proof: By strict monotonicity

$$\begin{aligned} 1 \triangle 0 &> 0 \triangle 0 \\ &= 0 \end{aligned}$$

Therefore, $1 \triangle 0 > 0$.

QED

Lemma 4: A weak operator ∇ cannot satisfy the union property.

Proof: By strict monotonicity

$$\begin{aligned} 0 \nabla 1 &< 1 \nabla 1 \\ &= 1 \end{aligned}$$

Therefore, $0 \nabla 1 < 1$.

QED

Lemma 5: A weak operator Δ cannot satisfy the complementarity property.

Proof:

$$\begin{aligned} (\textbf{NOT } 1) \Delta 1 &> (\textbf{NOT } 1) \Delta 0 && \text{strict monotonicity on 2nd component} \\ &\geq 0 \Delta 0 && \text{since (NOT 1) lies in [0,1]} \\ &= 0 \end{aligned}$$

Therefore, $(\textbf{NOT } 1) \Delta 1 > 0$.

$$\begin{aligned} (\textbf{NOT } 0) \Delta 0 &< (\textbf{NOT } 0) \Delta 1 && \text{strict monotonicity on 2nd component} \\ &\leq 1 \Delta 1 && \text{since (NOT 0) lies in [0,1]} \\ &= 1 \end{aligned}$$

Therefore, $(\textbf{NOT } 0) \Delta 0 < 1$.

QED

An attractive feature that an information retrieval system may have is the ability to handle weak operators as well as strict Boolean operators, i.e. the min and max operators. Thus, allowing the user to enter queries such as:

$$(\textit{interfaces}\ \mathbf{AND}^{weak}\ \textit{windows})\ \mathbf{AND}^{strict}\ (\text{author}\ \textit{Jones}).$$

In this query the user was able to express a weak desire that the documents retrieved be about both *interfaces* and *windows*, as well as the requirement that all retrieved documents be written by *Jones*. However, as the following lemma shows, any system containing these two kinds of operators cannot satisfy the distributivity property.

Lemma 6: Any system with weak operators and containing the strict Boolean operators *min* and *max* cannot satisfy distributivity.

Proof: (By contradiction) Let $\triangle$ be a weak conjunction, $\triangledown$ be a strict disjunction (max), and suppose that $\triangledown$ distributes over $\triangle$.

By strict monotonicity of $\triangle$,

$$(0 \triangle 1) > (0 \triangle .5).$$

Thus,

$$\begin{aligned}
0 \triangle 1 &= (0 \triangle 1) \triangledown (0 \triangle .5) && \text{since } \triangledown \text{ is max} \\
&= [0 \triangledown (0 \triangle .5)] \triangle [1 \triangledown (0 \triangle .5)] && \text{by distributivity} \\
&= (0 \triangle .5) \triangle 1 && \text{since } \triangledown \text{ is max} \\
&> (0 \triangle 0) \triangle 1 && \text{by strict monotonicity of } \triangle \\
&= 0 \triangle 1
\end{aligned}$$

Therefore, we have a contradiction.

QED

Note that the statement of Lemma 6 is not very strong in the sense that it does not say anything about distributivity when only weak operators are present. In fact, distributivity can hold among weak operators. Consider a system in which the disjunction $\triangledown$ and the conjunction $\triangle$ are both the average operation. These are clearly weak combining functions since:

1. $0 \triangle 0 = \text{avg}(0,0) = 0$ and $1 \triangle 1 = \text{avg}(1,1) = 1$.

2. $x \triangle y = \text{avg}(x,y) = \text{avg}(y,x) = y \triangle x$.

3. $\triangle$ is strictly monotonic on each component since avg is.

Distributivity holds because

$$\begin{aligned} x \bigtriangledown (y \triangle z) &= x \bigtriangledown (\frac{y+z}{2}) \\ &= \frac{x + \frac{y+z}{2}}{2} \\ &= \frac{2x+y+z}{4} \end{aligned}$$

and

$$\begin{aligned} (x \bigtriangledown y) \triangle (x \bigtriangledown z) &= (\frac{x+y}{2}) \triangle (\frac{x+z}{2}) \\ &= \frac{(\frac{x+y}{2}) + (\frac{x+z}{2})}{2} \\ &= \frac{2x+y+z}{4}. \end{aligned}$$

Lemma 7: The p-norm model satisfies idempotence, commutativity, involution, relation of 1 and 0, and De Morgan's rules.

Proof:

idempotence

$$\begin{aligned} (s, w_1)\ \textbf{AND}^p\ (s, w_2) &= 1 - \sqrt[p]{\frac{w_1^p(1-s)^p + w_2^p(1-s)^p}{w_1^p + w_2^p}} \\ &= 1 - \sqrt[p]{\frac{(w_1^p + w_2^p)(1-s)^p}{w_1^p + w_2^p}} \\ &= s \end{aligned}$$

A similar argument holds for the case when the operator is an **OR**.

commutativity This is trivially true since addition is commutative.

involution **NOT** (**NOT** s) = 1 - (1 - s) = s

Relation of 1 and 0 $\quad$ **NOT** $1 = 1 - 1 = 0$

De Morgan's Rules

$$(\textbf{NOT}\ s_1, w_1)\ \textbf{AND}^p\ (\textbf{NOT}\ s_2, w_2)$$

$$\|$$

$$1 - \sqrt[p]{\frac{w_1^p(1-(1-s_1))^p + w_2^p(1-(1-s_2))^p}{w_1^p + w_2^p}}$$

$$\|$$

$$1 - \sqrt[p]{\frac{w_1^p s_1^p + w_2^p s_2^p}{w_1^p + w_2^p}}$$

$$\|$$

$$\textbf{NOT}((s_1, w_1)\ \textbf{OR}^p\ (s_2, w_2))$$

QED

6.4 Conclusion

An analysis of the p-norm model with respect to the criteria of the Waller-Kraft wish list was presented, showing that the p-norm model satisfies most of the wish list. Upon consideration of the criteria which the p-norm model fails to satisfy, it was found that only the Boolean self-consistency property is actually desirable. However, it was shown that any system attempting to overcome the drawbacks of the fuzzy set model through the use of *weak operators* cannot satisfy basic desirable Boolean algebra properties, such as associativity.

Chapter 7

Conclusion

Theoretical as well as practical issues relating to the p-norm information retrieval model are explored in this thesis in response to the following criticisms that have been made of the p-norm model:

1. The p-norm model does not satisfy all Boolean algebra properties.

2. P-norm retrieval is too slow to be useful.

3. Formulating p-norm queries is difficult for untrained users.

The theoretical investigation includes a study of the desirable properties that an extended Boolean system should have as stated in the Waller-Kraft wish list [WK79], and how the p-norm model relates to them. Boolean self-consistency, i.e. the property stating that logically equivalent queries should produce identical results, is thoroughly studied. A key result of this study is that some basic Boolean algebra properties, such as associativity, cannot be satisfied by any extended Boolean system with *weak* operators. Weak operators, defined as strictly monotonic functions on each component, are clearly desirable because they overcome many of the drawbacks of the fuzzy set model.

This thesis presents a new p-norm retrieval algorithm which evaluates the entire document collection in one recursive traversal of the query tree and compares

it against the straightforward algorithm formerly implemented in the SMART system, which requires a traversal of the query tree for each document that is evaluated. An asymptotic analysis showed that the time complexity of the new algorithm is better, and an experimental analysis shows a reduction in retrieval time of 40%.

Since p-norm retrieval with the new algorithm is still significantly slower than pure Boolean retrieval, further efficiency improvements are introduced by means of approximations. The infinity-one model is introduced and shown to closely approximate the effectiveness of the p-norm model without requiring exponentiation. Furthermore, in a setting in which only a small number of documents in response to a search request are desired, the variable threshold pruning method for reducing the size of the inverted lists manipulated by the infinity-one retrieval algorithm is shown to significantly reduce the retrieval time further without affecting the precision of the top ranked documents. The retrieval time of the infinity-one model together with variable threshold pruning was found to be comparable to pure Boolean retrieval time. In fact, in INSPEC it was found to be 28% faster.

The implication of these efficiency results is that it really doesn't make sense to use pure Boolean systems. With the p-norm model, far better retrieval effectiveness is obtained, along with comparable retrieval efficiency via the infinity-one model with variable threshold pruning. Thus, current information retrieval systems based on pure Boolean retrieval can improve significantly by interpreting Boolean queries entered by users as p-norm queries. This modification can be done very simply since the front-end of the system can remain the same. Users do not even need to be aware of this change. However, users would find the system much easier to use since they would no longer have to worry about too many or too few documents being retrieved, a very common concern when writing Boolean queries.

The possibility of generating p-norm queries automatically from natural language search requests by relying on the syntactic structure of the sentences is also explored in this thesis. With the algorithm presented, the effectiveness obtained is found to be comparable to manually constructed queries and far better than the statistically generated queries obtained from the algorithm presented in [SBF83]. However, the syntactically constructed queries outperform the standard vector model only in the experiments with MEDLARS, a collection with a very technical vocabulary. It was observed that queries with structure tend to be more sensitive to indexing errors than vector queries. To overcome this drawback of structured queries, a combination of the p-norm model and the vector model was proposed. It was found that the vector model has a moderating effect on the p-norm model, which leads to significant improvements in effectiveness in the three collections — the most impressive being in INSPEC where a 21.1% increase in average precision is obtained when compared with the average precision of the standard vector model.

Further work with p-norm queries should further explore the issue of parameter setting. In this thesis, only term frequency information was used to assign term weights, clause weights, and p-values. Co-occurrence information of the terms in a clause could prove to be useful for clause weighting and p-value assignments. The use of a technical dictionary may also be useful for this purpose because the goodness of a clause of single terms can be based on whether the phrase made up of the terms in a clause appears as an entry in the dictionary. The dictionary may also help in expanding the query with synonyms by means of the **OR** operator — this natural means of query expansion is not present in unstructured systems.

Further work should also explore the use of the p-norm model in environments which require both exact matching and partial matching, such as an electronic mail system. In such a system, queries could include specific information, such as

the date of the message or the sender of the message, as well as fuzzy information, expressing the subject of the message. The p-norm model may prove to be most useful in these types of environments.

Bibliography

[AO69] Carl B. Allendoerfer and Cletus O. Oakley. *Principles of Mathematics.* McGraw-Hill, Inc., 1969.

[BK81a] Duncan A. Buell and Donald H. Kraft. A model for a weighted retrieval system. *Journal of the American Society for Information Science*, 32:211–216, 1981.

[BK81b] Duncan A. Buell and Donald H. Kraft. Threshold values and boolean retrieval systems. *Information Processing & Management*, 17:127–136, 1981.

[Boo78] A. Bookstein. On the perils of merging boolean and weighted retrieval systems. *Journal of the American Society for Information Science*, 29:156–158, 1978.

[Boo80] Abraham Bookstein. Fuzzy requests: An approach to weighted boolean searches. *Journal of the American Society for Information Science*, 31:240–247, 1980.

[Boo81] A. Bookstein. A comparison of two systems of weighted boolean retrieval. *Journal of the American Society for Information Science*, 32(4):275–279, July 1981.

[Buc85] Chris Buckley. Implementation of the smart information retrieval system. Technical Report 85-686, Cornell University, May 1985.

[Bue81] Duncan A. Buell. A general model of query processing in information retrieval systems. *Information Processing & Management*, 17:249–262, 1981.

[BVW72] F.H. Barker, D.C. Veal, and B.K. Wyatt. Towards automatic profile construction. *Journal of Documentation*, 28(1):44–55, March 1972.

[CBH85] M.S. Chodorow, R.J. Byrd, and G.E. Heidorn. Extracting semantic hierarchies from a large on-line dictionary. In *Proceedings of the*

23rd Annual Meeting of the Association for Computational Linguistics, pages 299–304, 1985.

[CK87] Steven C. Cater and Donald H. Kraft. Tirs: A topological information retrieval system satisfying the requirements of the waller-kraft wish list. In *Proceedings of the Tenth Annual International ACM-SIGIR Conference on R&D in Information Retrieval*, pages 171–180, 1987.

[CK89] Steven C. Cater and Donald H. Kraft. A generalization and clarification of the waller-kraft wish list. *Information Processing & Management*, 25(1):15–25, 1989.

[DG87] Padmini Das-Gupta. Boolean interpretation of conjunctions for document retrieval. *Journal of the American Society for Information Science*, 38(4):245–254, 1987.

[Fag87] Joel L. Fagan. *Experiments in Automatic Phrase Indexing For Document Retrieval: A Comparison of Syntactic and Non-Syntactic Methods.* Ph.D. dissertation, Cornell University, 1987.

[Fox83] Edward Alan Fox. *Extending the Boolean and Vector Space Models of Information Retrieval with P-norm Queries and Multiple Concept Types.* Ph.D. dissertation, Cornell University, 1983.

[GK83] Hanna Grzelak and Kazimierz Kowalski. Automatic construction of information queries. *Information Processing & Management*, 19(6):381–389, 1983.

[Hei72] George E. Heidorn. *Natural Language Inputs to a Simulation Programming System.* Ph.D. dissertation, Yale University, 1972.

[HH70] David Hsiao and Frank Harary. A formal system for information retrieval from files. *Communications of the ACM*, 13(2):67–73, 1970.

[HJM+82] G.E. Heidorn, K. Jensen, L.A. Miller, R.J. Byrd, and M.S. Chodorow. The epistle text-critiquing system. *IBM Systems Journal*, 21(3):305–326, 1982.

[KB83] Donald H. Kraft and Duncan A. Buell. Fuzzy sets and generalized boolean retrieval systems. *International Journal on Man-Machine Studies*, 19:45–56, 1983.

[Lee88] Whay C. Lee. Experimental comparison of schemes for interpreting boolean queries. Masters dissertation, Virginia Polytechnic Institute and State University, 1988.

[Pai84] C.D. Paice. Soft evaluation of boolean search queries in information retrieval systems. *Information Technology*, 3(1):33–41, January 1984.

[Rad76] Tadeusz Radecki. Mathematical model of information retrieval system based on the concept of fuzzy thesaurus. *Information Processing & Management*, 12:313–318, 1976.

[Rad77] Tadeusz Radecki. Mathematical model of time-effective information retrieval system based on the theory of fuzzy sets. *Information Processing & Management*, 13:109–116, 1977.

[Rad79] Tadeusz Radecki. Fuzzy set theoretical approach to document retrieval. *Information Processing & Management*, 15:247–259, 1979.

[Rad82] Tadeusz Radecki. Reducing the perils of merging boolean and weighted retrieval systems. *Journal of Documentation*, 38(3):207–211, September 1982.

[Rob78] Stephen E. Robertson. On the nature of fuzz: A diatribe. *Journal of the American Society for Information Science*, 29:304–307, 1978.

[Sal71] Gerard Salton. *The SMART Retrieval System: Experiments in Automatic Document Processing.* Prentice-Hall, 1971.

[Sal75] Gerard Salton. *Dynamic Information and Library Processing.* Prentice-Hall, 1975.

[Sal89] Gerard Salton. *Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer.* Addison-Wesley Publishing Company, Inc., 1989.

[SB88] Gerard Salton and Christopher Buckley. Term-weighting approaches in automatic text retrieval. *Information Processing & Management*, 24(5):513–523, 1988.

[SBF83] G. Salton, C. Buckley, and E.A. Fox. Automatic query formulations in information retrieval. *Journal of the American Society for Information Science*, 34(4):262–280, 1983.

[SFW83] G. Salton, E.A. Fox, and H. Wu. Extended boolean information retrieval. *Communications of the ACM*, 26(11):1022–1036, Nov 1983.

[SM83] Gerard Salton and Michael J. McGill. *Introduction to Modern Information Retrieval.* McGraw-Hill, Inc., 1983.

[SP86] Michael A. Shepherd and W.J. Phillips. The profile-query relationship. *Journal of the American Society for Information Science*, 37(3):146–152, 1986.

[SS89] G. Salton and M. Smith. On the application of syntactic methodologies in automatic text analysis. In *Proceedings of the Twelfth Annual SIGIR Conference*, pages 137–150, 1989.

[SV85] Gerard Salton and Ellen Voorhees. Automatic assignment of soft boolean operators. In *Proceedings of the Eighth Annual SIGIR Conference*, pages 54–69, 1985.

[Tah76] Valiollah Tahani. A fuzzy model of document retrieval systems. *Information Processing & Management*, 12:177–187, 1976.

[Tah77] Valiollah Tahani. A conceptual framework for fuzzy query processing—a step toward very intelligent database systems. *Information Processing & Management*, 13:289–303, 1977.

[Usz86] Marc Uszynski. Fuzzy queries with linguistic quantifiers for information retrieval from data bases. Technical Report UCB/CSD 87/333, University of California, July 1986.

[Voo85] Ellen M. Voorhees. *The Effectiveness and Efficiency of Agglomerative Hierarchic Clustering in Document Retrieval.* Ph.D. dissertation, Cornell University, 1985.

[Win83] T. Winograd. *Language as a Cognitive Process.* Addison-Wesley, 1983.

[WK79] W.G. Waller and Donald H. Kraft. A mathematical model of a weighted boolean retrieval system. *Information Processing & Management*, 15:235–245, 1979.

[Wu81] Harry Chih Chien Wu. *On Query Formulation in Information Retrieval.* Ph.D. dissertation, Cornell University, 1981.

[WZW85] S.K.M. Wong, Wojciech Ziarko, and Patrick C.N. Wong. Generalized vector space model in information retrieval. In *Proceedings of the Eighth Annual SIGIR Conference*, pages 18–25, 1985.

[Zad65] L.A. Zadeh. Fuzzy sets. *Information and Control*, 8:338–353, 1965.