Kim, N., French, J., & Brown, D. (1993). Boolean Query Reformulation with the Query Tree Classifier. 4th ASIS SIG/CR Classification Research Workshop, 123-138. doi:10.7152/acro.v4i1.12615

Boolean Query Reformulation with the Query Tree Classifier

Nam-Ho Kim
Department of Systems Engineering
University of Virginia
Charlottesville, VA 22903

James C. French
Department of Computer Science
University of Virginia
Charlottesville, VA 22903
french@juliet.cs.virginia.edu

Donald E. Brown
Department of Systems Engineering
University of Virginia
Charlottesville, VA 22903
deb@virginia.edu

One of the difficulties in using the current Boolean-based information retrieval systems is that it is hard for a user, especially a novice, to formulate an effective Boolean query. Query reformulation can be even more difficult and complex than formulation since the user can have difficulty in incorporating the new information gained from the previous search into his/her next query. In this research, query reformulation is viewed as a classification problem (i.e., classifying documents as either relevant or nonrelevant), and a new reformulation algorithm is proposed which builds a tree-structured classifier (named the query tree) at each reformulation from a set of feedback documents retrieved from the previous search. The query tree can be easily transformed into a Boolean query. The query tree and two of the most important current query reformulation algorithms were compared on benchmark test sets (CACM, CISI, and Medlars). The query tree showed significant improvements over the current algorithms in most experiments. We attribute this improved performance to the ability of the query tree algorithm to select good search terms and to represent the relationships among search terms into a tree structure.

1. INTRODUCTION

Modern computing technologies have had a great impact on the ways information is handled (e.g., from papers to electronic files), and brought many benefits to today's complicated information management. By digitizing information, we are able to store massive amounts of information in computer disks. For example, a single compact disc can hold 20 volumes of an encyclopedia with sound, and thousands of pictures. In addition, the current proliferation of networks and on-line databases make it possible to disseminate and retrieve information in fast and easy ways. Information updates are simpler and cheaper in an electronic form than on paper. One might think that electronic access would provide all the necessary information from a large database in a short time with little effort. However, consider retrieving relevant articles from a collection of millions of documents. A system with low performance can easily overload the user with unnecessary information or lose potentially valuable information. What most users want is quality output, not quantity. The fact is that finding the right information gets harder as more information becomes available, and the big question is how the user can retrieve only the information he requested.

ISSN: 2324-9773

Kim, N., French, J., & Brown, D. (1993). Boolean Query Reformulation with the Query Tree Classifier. 4th ASIS SIG/CR Classification Research Workshop, 123-138. doi:10.7152/acro.v4i1.12615

The most fundamental information retrieval (IR) model is the Boolean retrieval model. The query is a Boolean expression and the search is based on exact matching. Information is generally stored in inverted files. The advantages of this model are simple structure, easy implementation, and fast retrieval. Its inverted file structure allows set operations to be carried out easily. Critics of the Boolean retrieval model [Bookstein, 1985; Araya, 1990] point out that several disadvantages exist in this model. First, it is hard to control the size of the retrieved document set. Second, there is no partial matching between a query and documents. If there is a term in a document which disagrees with a query, it will not be retrieved even though every other terms satisfy the query. Finally, since there is no term weight, the retrieved documents have equal importance which means no ranking capability. To overcome these limitations, other advanced IR models have been developed such as vector space, probabilistic, cluster-based models, etc. But the major operational IR systems are still based on the Boolean retrieval model. There are a number of reasons for this situation [Radecki, 1982]: (1) the Boolean system is easy and efficient to implement in an online system; (2) there are doubts as to whether other systems would work in operational IR systems (most experiments are done on experimental systems); and (3) it may not be economical to include new technologies in existing systems. One way to overcome this roadblock is to work alongside the existing Boolean-based systems.

2. PROBLEMS

The main purpose of an information retrieval system is to deliver user-requested information quickly, accurately, and easily. Current IR systems are often criticized for delivering only a portion of what is requested along with nonrelevant information. Improvements in retrieval performance (i.e., ability to retrieve only the relevant information) can be achieved from more effective query formulation. The quality of a query fed to an IR system has a direct impact on the success of the search outcomes, i.e., garbage in garbage out. In fact one of the most important but frustrating tasks in information retrieval is query formulation. Most IR system users, especially novices, experience some degree of difficulty in transforming their information needs into the required query language (e.g., Boolean expression) of an IR system. In addition, there is very little help provided by current IR systems. Users usually employ a trial-and-error search and often have to rely on an intermediary (e.g., librarian) in searching for the right information. As more computers and networks become available, experts would be less likely to be present in every site and expensive to employ. There is a definite need for an IR system which cannot only help users formulate an effective query but also increase retrieval performance.

Information search is an iterative process (in another word, a reformulation process). In current IR systems, a user is responsible for constructing and providing the search with the system. The reformulation process can be more difficult and complex than the formulation process. From examining the previously retrieved documents, the user has greater difficulty in incorporating the new information into his next query. It is more natural and easier for a user to identify or present the relevance of information than to formulate a good query. Because the process is based on a trial-and-error instead of systematic approaches, it is time-consuming and prone to errors. Variabilities and inconsistencies of indexing schemes in different IR systems can also pose other hurdles to the typical user.

ISSN: 2324-9773

One of the major research areas in information retrieval is how to improve the Boolean-based IR system's abilities (referred as *effectiveness* in information retrieval) to retrieve relevant information and to reject nonrelevant information through the use of the relevance judgments of the previously retrieved documents in query reformulation (this technique is often referred as relevance feedback). Another area of major research is how to ease the user's role of query formulation through automating the process of query formulation. A number of researchers [Vernimb, 1977; Dillon and Desper, 1980; Dillon et al., 1983; Salton et al., 1984, 1985; Frants and Shapiro, 1991b] have been working on automating the process of Boolean query reformulation using the relevance feedback. The techniques proposed by these researchers share two major steps: the selection of good search terms and formulation of a query in disjunctive normal form. The main differences are in the ways to perform these two tasks. For query formulation, the initial inputs to the system are received from the user in the form of either a list of search terms or a couple of relevant documents. If the initial inputs to the algorithm are documents, index terms which are assigned to the documents are selected and ordered by the term importance which measures the degree of term contribution to the search. Search terms with the highest term importance are selected and formulated in disjunctive normal form (the term selection and formulation procedures depend on an algorithm). If it is a reformulation process, a set of the previously retrieved documents with their relevance are fed back to the system; the term importances of the index terms are updated and reordered; and the query is reformulated as in the formulation process. For details, see [Salton *et al.*, 1984].

The main goal of this research is to develop a Boolean query reformulation method which improves the effectiveness of the Boolean-based IR systems, and automates the reformulation process. To achieve the goal, a new algorithm for the automatic reformulation of Boolean query is developed from the perspective of classification. Query formulation is in fact a classification problem, that is, to classify the documents as either relevant or nonrelevant. A classification technique, based on CART (Classification And Regression Trees [Breiman, *et al.*, 1984]), was used to formulate a Boolean query from a classification tree (named the *query tree*) capable of retrieving relevant documents from the database. The two key factors in Boolean query formulation are identification of good search terms and their relationships, which are represented by the Boolean operators. In the query tree, promising terms are selected by a misclassification cost function, and term relationships are represented by a tree structure.

3. THE QUERY TREE CLASSIFIER

Query formulation is in fact a classification problem; a query can be regarded as a classifier which assigns documents in the database to the proper classes. In classification, there can be a number of classes; for example, documents can be classified as highly relevant, relevant, somewhat relevant, or nonrelevant. For this research, two classes of documents are considered, relevant and nonrelevant.

The *query tree* is a tree classifier based on CART, capable of assigning documents to either relevant or nonrelevant categories. An example is illustrated in Figure 1. The nodes in a query tree are categorized into two types. The leaves, nodes with no descendants are called *terminal nodes* (in

ISSN: 2324-9773

Kim, N., French, J., & Brown, D. (1993). Boolean Query Reformulation with the Query Tree Classifier. 4th ASIS SIG/CR Classification Research Workshop, 123-138. doi:10.7152/acro.v4i1.12615

short, T-nodes), e.g., Node 4, 6, 8, 9, 10, and 11; the other nodes are called *nonterminal nodes* (in short, NT-nodes), e.g., Node 1, 2, 3, 5, and 7. Let X denote a query tree consisting of a set of T-nodes (denoted as X_T) and a set of NT-nodes (denoted as X_N). For each node $x \varepsilon X_T$, a class is assigned as either relevant or nonrelevant. For each node $x \in X_N$, a criterion is set up to guide input documents either to the right or left child node. Let d be a document in the database D and $t_j = 1$ denote that the j^{th} index term is assigned to d. The criterion 1 in Figure 1 could be of the following form:

for $\forall\, d \in D$, if $t_j = 1$, then send d to Node 2, else to Node 3.

Once a query tree is constructed, documents are classified as follows: each document $d \varepsilon D$ is assigned to the root node. According to the criterion in the root node, d is sent either to the right or left child node. By the next node's criterion, d is sent to one of its child nodes. This process continues until the document d reaches one of the T-nodes. Then, d is classified as either relevant or nonrelevant by the class of the reached terminal node.

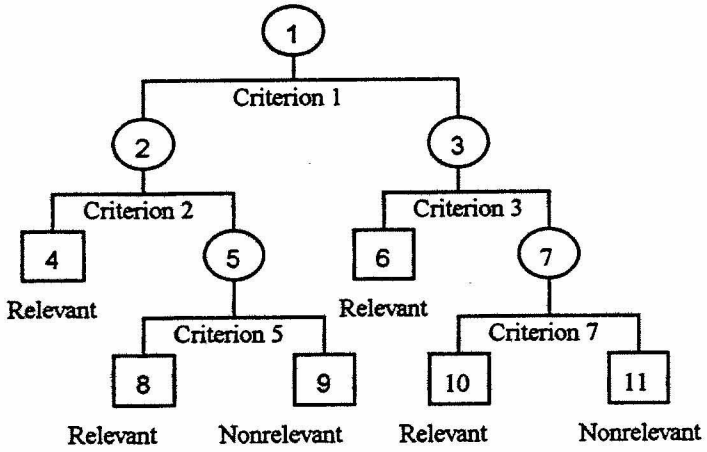


Figure1 An Example of CART

3.1. Construction of the Query Tree

To construct a query tree, a sample of documents with known relevance is required. Let D_I be a set of initial inputs from the database D where D_I consists of two classes: relevant (D_{Ir}) and nonrelevant (D_{In}). Construction of a tree classifier (query tree) consists of the three operations [Breiman, *et al.*, 1984]: (1) at each node starting from the root node, "Are we going to split the node?" — determination of the terminal nodes; (2) if the node is split, "How are we going to split the data (e.g., documents) into the child nodes?" — selection of splitting criteria for the

ISSN: 2324-9773

nonterminal nodes; and (3) if the node is not split, "What class does the node belong to?" — assignments of the terminal nodes to a class.

The main idea of the query tree is to identify and capture characteristics of the input data and to group them into a same class (e.g., what makes this input data belong to a certain class?). The node split is stopped when the data collected in the node becomes homogeneous. The impurity of node x, $\psi(x)$ is defined as

$$\psi(x) = \min_j [\Pr(j|x)] \quad \text{where } j = \text{"relevant" or "nonrelevant".}$$

The impurity function $\Psi(x)$ is maximum when Pr("relevant" | x) = Pr("nonrelevant" | x), and minimum when one of the proportions is zero; its range is $0 \le \psi(x) \le 0.5$ and it is symmetric along the line Pr("relevant" | x) = 0.5 (e.g., Pr("relevant" | x) = 0.7 and Pr("relevant" | x) = 0.3 have the same $\psi(x) = 0.3$). The determination of the T-node (the first operation in the tree classifier construction) can be achieved by setting a small threshold value δ on the impurity $\psi(x)$ such as,

If $\psi(x) < \delta$ then stop the splitting and set the node x as terminal,
else continue the splitting.

A good criterion for splitting a node would makes its child nodes "purer" than their parent node. For our example, it would be the criterion which divides the relevant and nonrelevant documents perfectly. A reasonable splitting criterion would provide the child nodes with the smallest impurity levels. The goodness of the split can be measured by the decrease in impurity level from node x to its child nodes [Breiman, *et al.* 1984]. That is,

$$\Delta\psi(x,\sigma) = \psi(x) - P_L\psi(x_L) - P_R\psi(x_R)$$

where σ is a splitting criterion, and P_L and P_R are the proportion of document sent from the node x to the left child node x_L and the right child node x_R, respectively. In the query tree, when a node is determined to be a NT-node (i.e., decided to be split), one of possible splitting criterion would be to select an index term t* in T_I which provides the highest decrease in impurity, i.e., $t^* = \max(\Delta\psi(x, t))$ for all $t \in T_I$. After a term t^* is selected, the splitting criterion at node x in the query tree is as follows. For each document d in the node x,

if the term t^* is present in d then send d to the left child,
else send d to the right child.

When the node x is determined as a T-node, the next step is to assign a class to the node x (the third operation in the query tree construction). A simple way to determine a class of a node is to compare the class proportions at node x such as

if Pr(*relevant* | x) > Pr(*nonrelevant* | x)
then the class of the node x is "relevant"
else the class of the node x is "nonrelevant".

ISSN: 2324-9773

Kim, N., French, J., & Brown, D. (1993). Boolean Query Reformulation with the Query Tree Classifier. 4th ASIS SIG/CR Classification Research Workshop, 123-138. doi:10.7152/acro.v4i1.12615

The query tree construction procedure is summarized in Figure 2. It is clearer to describe the query tree algorithm with an example. Suppose that the sample of documents (4 relevant and 5 nonrelevant) as in Figure 3 is used as an input to the algorithm. Their relevance judgments are hypothetically created for a user's information need, "articles on expert system applications in the area of chemistry and physics." The constructed query tree is also depicted in Figure 3.

Suppose that the threshold δ is set to be 0.1. The initial 9 documents are assigned to the root node (x_1) and x_1 is pushed into a stack called *N_stack* (Step 1.1). The index terms assigned to the initial nine documents are collected and stored in T_I (Step 1.2). Since *N_stack* is not empty, the root node is then popped from *N_stack* and set to be *C_node* (Step 2.1). The impurity level of the root node (i.e., $\psi(C\_node) = 4/9$) is higher than δ. In Step 2.2. the index term "exp. sys." is selected, which provides the highest $\Delta\psi(C\_node, t) = 2/9$. The splitting criterion *Criterion*(x_1, "exp.sys.") is to send documents with the term "exp.sys." to the left child, otherwise to the right child. The left (x_2) and right (x_3) child nodes are created and the documents 1, 2, 3, 4, 7, and 8 are assigned to x_2, while the rest are assigned to x_3 (Step 2.2.3). Then x_2 and x_3 are pushed into *N_stack* for further consideration (Step 2.2.4). This is the end of one iteration in the WHILE loop.

With two nodes in *N_stack*, x_2 is popped from *N_stack* and its node impurity level is computed. With $\psi(x_2) = 1/3$, it is still higher than δ and another node split is carried out. The previously used

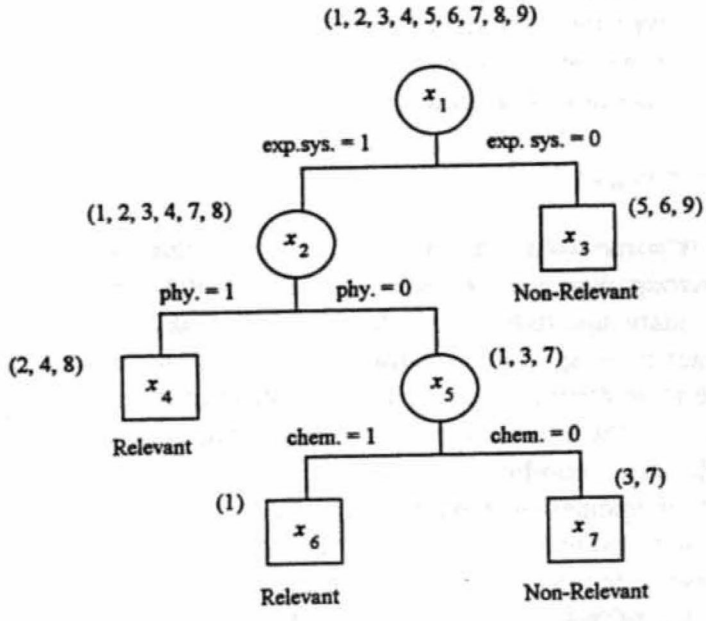
(Initialization)
1.1 Assign initial input documents D_I to the root node and put the root node into stack *N_stack*.
1.2 Find terms assigned to D_I and save them in T_I.

(Tree Construction)
WHILE ($N\_stack \neq \varnothing$) DO
2.1 *C_node* = *Top(N_stack)*
2.2 IF ($\psi(C\_node) < \delta$)
THEN
2.2.1 Set *C_node* as T-node and assign a class to the node.
ELSE
2.2.2 Select a term *t**
where *t** = max $\Delta\psi(C\_node, t)$ and $t \in T_I$
2.2.3 Create the left and right child nodes for *C_node* and assign the data in *C_node* to the child nodes according to *Criterion* (*C_node*, *t**).
2.2.4 Put these child nodes in *N_stack*.
END OF IF
END OF WHILE

Figure 2 Procedure for the Query Tree Construction

ISSN: 2324-9773

index terms are excluded for subsequent splits. This time the term "phy." is chosen with the highest $\Delta\psi(x_2$, "phy.") = 1/6. The documents 2, 4, and 8 which have index term "phy." are sent to the left child node x_4 and documents 1, 3, and 7 to the right child node x_5. Then x_4 and x_5 are pushed into *N_stack*. At the next iteration, the node x_4 is popped from the stack and its $\psi(x_4)$ is 0, so it is assigned as the T-node and the class is "relevant" with all the relevant documents at the node (Step 2.2.1). The next node is popped again from *N_stack* and it is x_5. With the $\psi(x_5) = 1/3$, the node x_5 gets split and this process continues until *N_stack* is empty.



exp. sys.	a	b	c	phy.	d	chem.	.....	RELEVANCE.
#1	1	0	1	0	0	1	1	.... r
#2	1	0	0	0	1	1	1	.... r
#3	1	0	0	1	0	1	0	.... n
#4	1	1	1	0	1	1	0	.... r
#5	0	0	0	0	1	1	1	.... n
#6	0	0	1	0	1	1	0	.... n
#7	1	0	0	0	0	0	0	.... n
#8	1	0	0	0	1	1	0	.... r
#9	0	0	1	0	0	1	1	.... n

Note: the alphabets (e.g., "a" and "b") represent index terms and the numbers (e.g., 1 and 2) represent documents. The abbreviations are exp. sys. = expert systems; physics;chem. = chemistry; r = relevant; and n = nonrelevant. An element "1" means the index term is assigned; otherwise it is not.

Figure 3 An Example of the Query Tree

ISSN: 2324-9773

Kim, N., French, J., & Brown, D. (1993). Boolean Query Reformulation with the Query Tree Classifier. 4th ASIS SIG/CR Classification Research Workshop, 123-138. doi:10.7152/acro.v4i1.12615

The basic concept of selecting a splitting criterion at node x in the query tree is to find an index term with the largest decrease in impurity level (i.e., max $\Delta\psi(x, t)$). Experiments conducted for this study showed that the input documents, especially from the feedback documents during reformulations, very often contain more nonrelevant documents than the relevant ones. That is, at the root node the ratio of the nonrelevant to the relevant is much larger than 1. When this is the case, care must be taken in selecting an index term for node split.

Let $C_{r/n}$ and $C_{n/r}$ be the costs of assigning a document as relevant given that it is nonrelevant, and nonrelevant given that it is relevant, respectively. Let $N_{r/n}(t,x)$ and $N_{n/r}(t,x)$ be the number of documents assigned as relevant given that they are nonrelevant, and the number of documents assigned as nonrelevant given that they are relevant at the node x by the term t, respectively. The total cost $M(t,x)$ of misclassifying documents at node x by the index term t can be defined as

$$M(t,x) = C_{r/n} \cdot N_{r/n}(t,x)) + (C_{n/r} \cdot N_{n/r}(t,x))$$

For the term selection criterion ($t^*=\max\Delta\psi(x, t)$ in Figure 2 Step 2.2.2), there are no costs involved in assigning documents to the wrong class; the cost is assumed to be equal. There are two different types of node splitting policies: static and dynamic. If the costs $C_{r/n}$ and $C_{n/r}$ are set at the beginning and do not vary at each node split, the splitting policy is said to be static; on the other hand, if the costs vary, it is said to be dynamic. In a static node splitting policy, there is a large number of combinations of costs which a policy can choose from (e.g., $C_{r/n}$ =1 $C_{n/r}$ =2; $C_{r/n}$ =3 $C_{n/r}$ =1), and it is hard to decide which combination to use. In this research, a dynamic node splitting policy is proposed, which updates the costs $C_{r/n}$ and $C_{n/r}$ automatically at each node according to the proportion of the relevant documents to the nonrelevant. Let $Nr(x)$ and $Nn(x)$ be a number of relevant and nonrelevant documents at node x, respectively. In a dynamic splitting policy used here, the total cost M_α of misclassifying documents at node x by the index term t is defined as

$$M_\alpha(t,x) = \alpha \cdot N_{r/n}(t,x) + (1 - \alpha) \cdot N_{n/r}(t,x) \text{ where } \alpha = (N_r(x)/(N_r(x) + N_n(x))$$

In this cost function, $C_{r/n}$ and $C_{n/r}$ are replaced by α and (1- α) respectively and adjusted by the number of relevant and nonrelevant documents at node x. This means when there are more nonrelevant documents at the node than relevant ones, the cost of assigning the relevant to the nonrelevant is more than that of assigning the nonrelevant to the relevant (i.e., $\alpha < 0.5$). In the dynamic splitting policy, the cost weighting factor α changes from node to node; on the other hand, in a static splitting policy, the costs $C_{r/n}$ and $C_{n/r}$ are fixed never to change. The term selection rule in the dynamic policy is to select $t^* = \min M_\alpha(t, x)$ for all t in T_I. In this research, the dynamic splitting policy is used instead of a static splitting policy because the dynamic policy is always as good and frequently better than the static policy.

It is often the case that more than one index term has the same minimum total cost M_α (i.e., there are several candidate index terms to choose from). The dynamic splitting policy is extended to handle the multiple case. Another way to select a term for node split is to use term importance factors such as the "prevalence" proposed by Dillon's method and the "relevance weight" by the DNF method. In our study, the function called term contribution is used, that is,

ISSN: 2324-9773

$$\xi(t) = f_r(t)/f(t) \qquad \text{where } 0 \le \xi(t) \le 1$$

The functions $f_r(t)$ and $f(t)$ represent the frequency of the term t in the relevant documents and the database, respectively. The basic idea of $\xi(t)$ is the assumption that the term importance of t is proportional to the frequency of t in the relevant documents and inversely proportional to the total frequency of t in the database. It is similar to the Inverse Document Frequency (IDF) where the numerator of IDF is a frequency of a term in a single document [Salton and McGill, 1983]. Let T^* be a set of index terms with the same minimum $M_\alpha(t, x)$. When there is more than one index term in T^*, the term contribution $\xi(t)$ can be used to decide which index term is to be selected (i.e., to select a term with the highest $\xi(t)$. That is,.

$$\text{If } |T^*| = 1 \quad \text{then } t^* = t \quad \text{where } t \in T^*$$
$$\text{else } t^* = \max_t \xi(t) \quad \text{where } t \in T^*$$

If there is still more than one index term with the same minimum $M_\alpha(t, x)$ and maximum $\xi(t)$, then select randomly one of index terms for node split (During the experiments, this case did not occur). Now we need to update Step 2.2.2 in Figure 2, i.e.,

2.2.2 At node x, collect terms t^* in T^* where $t^* = \min M_\alpha(t, x)$ for all $t \in T_I$
If $|T^*| = 1$ then $t^* = t$ where $t \in T^*$
else $\max_t \xi(t)$ where $t \in T^*$

3.2. Transformation of Query Tree into a Query in DNF

After the query tree is completed, the documents in the database D can be classified by sending them through the root node and see where they end. Examining every document in the database would be very inefficient, especially for a large database. The query tree can be transformed into a Boolean query in DNF. Call a query generated from the query tree the *Tree-derived query*. The transformation procedure is described in Figure 4. First, collect the T-nodes classified as relevant and denote them as X_{T_r}. For every $_x$ in X_{T_r}, starting from the node x, read and collect its parents' splitting index terms recursively up the root node; connect each term in the collected set with the Boolean operator *and*. If the term is equal to zero (e.g. between the node x_2 and x_5 in Figure 3), add the Boolean operator *not* in front of the term. After finishing this process with every relevant T-nodes, connect the *and*ed clauses (i.e., conjuncts) with the Boolean operator *or* (Step 3.2 in Figure 4); this will produce a Boolean query in DNF. From Figure 4, the DNF is

("exp.sys." *and* "phys.") *or* ("exp.sys." *and* (*not* "phys.") *and* "chem.")

This query is equivalent to

("exp.sys." *and* "chem.") *or* ("exp.sys." *and* "phys.").

ISSN: 2324-9773

Kim, N., French, J., & Brown, D. (1993). Boolean Query Reformulation with the Query Tree Classifier. 4th ASIS SIG/CR Classification Research Workshop, 123-138. doi:10.7152/acro.v4i1.12615


```
(Query Transformation)
1. FOR (each node x ∈ query tree) DO
     IF (x∈ X_Tr) THEN
        Set C_node = x
        WHILE (C_node ≠ root node) DO
           Collect a term t in parent(C_node)
           IF t = 1 THEN collect it in And_Term
           IF t = 0 THEN collect "not" + t in And_Term
           Set C_node = parent(C_node)
        END (WHILE)
        Connect terms in And_Term with "and" and store it in Or_Clause
     END (IF)
   END OF FOR
2. Formulate a query in DNF by connecting clauses in Or_Clause
   with "or" operator.
```

Figure 4 Query Generation from a query tree

This is the query generated by the query tree algorithm for the user's information need on "expert system applications in chemistry and physics", which would be normally formulated as the query just generated from the query tree.

3.3. Query Reformulation

So far we have discussed the query tree algorithm for a single query formulation. This section describes a reformulation process with the query tree. The query tree algorithm is based on the Boolean-based IR system and requires a set of initial inputs (learning sample) consisting of relevant documents. (The algorithm does not require nonrelevant documents.) The initial inputs can be supplied by the user in three forms: a list of few relevant documents, a list of search terms, or a Boolean query. If the inputs are search terms, the system can (1) retrieve documents with a Boolean query formulated as a disjunct of each search term supplied by the user, (2) present them to the user in order of coordination level (the number of terms a document has in common with a query), and (3) get relevance judgments in the retrieved set from the user. If the inputs are a Boolean query, the same process can be implemented, except for the Boolean query formulation.

A real advantage of the query tree algorithm as well as the others, DNF and Dillon's method, is in query reformulation with relevance feedback. Once an initial input is provided, the user is free from formulating a Boolean query, and only needs to examine the retrieved set for its relevance. The algorithm for query reformulation with the query tree is described in Figure 5.

ISSN: 2324-9773

Kim, N., French, J., & Brown, D. (1993). Boolean Query Reformulation with the Query Tree Classifier. 4th ASIS SIG/CR Classification Research Workshop, 123-138. doi:10.7152/acro.v4i1.12615

1. Prepare the initial inputs
2. WHILE (*stopping criteria* are not met) DO
   - 2.1 Construct the query tree (Figure 3-2)
   - 2.2 Transform the query tree into a Boolean query in DNF (Figure 3-4)
   - 2.3 Retrieve and present documents to the user
   - 2.4 Get relevance feedback and update the initial inputs.
3. END (WHILE)

Figure 5 Query Reformulation with the query tree

First, the initial inputs are prepared. Two sets of initial inputs are needed, one for tree growing and the other for tree pruning (Step 1). The algorithm can be interrupted by either itself or the user (Step 2). There can be several stopping criteria during reformulation. One criterion for an algorithm is no relevant documents in the input set. Another criterion is no or minimal changes between documents retrieved from the i^{th} reformulation and those from the $(i+1)^{th}$ reformulation. This implies that either all the relevant documents are retrieved, or the algorithm is not getting new information any more. The user can also stop the reformulation process if he is satisfied with the search outcomes or tired. Step 2.1 and 2.2 (the tree growing and transforming) have been discussed in the previous sections. Documents are retrieved and presented to the user for relevance judgments (Step 2.3). Then the relevance feedback is added to the input sets (Step 2.4). A new tree is grown from the updated input sets and the process is repeated until one of the stopping criteria is met.

4. EXPERIMENTAL RESULTS

To evaluate the retrieval performance (i.e. effectiveness) of the query tree, two of the current query reformulation algorithms were selected: the *disjunctive normal form* (DNF) method by Salton et al. [1984, 1985] and an algorithm (call it Dillon's method) by Dillon et al. [1980, 1983]. These algorithms were then compared on three widely-used test sets (the CACM, the CISI, and the Medlars). In general, a test set consists of a collection of documents with various attributes (titles, authors, abstracts, manually indexed keywords, etc.), a set of queries and their relevance judgments. Queries in the test sets are represented in natural language form. Some of the test sets (e.g., CACM and CISI) also come with a set of Boolean queries constructed from the natural language descriptions. Statistics of the test sets used in the experiments are listed in Table 1. For the experiments in this research, the original query sets are reduced to the sets of queries with more than 5 relevant documents since the algorithms require relevant documents in the initial inputs (see 3 and 4 in Table 1). The reported numbers of index terms are the number of unique terms after a stemming is performed. The parameter settings for the DNF and Dillon's methods are the same as those used in the experiments performed in [Salton et al., 1984].

ISSN: 2324-9773

Kim, N., French, J., & Brown, D. (1993). Boolean Query Reformulation with the Query Tree Classifier. 4th ASIS SIG/CR Classification Research Workshop, 123-138. doi:10.7152/acro.v4i1.12615

Table 1 Test Set Statistics

Description	CACM (Computer Sci.)	CISI (Information Sci.)	Medlars (Medicine)
1. Number of documents	3204	1460	1033
2. Number of index terms (after the stemming)	6166	5593	7276
3. Number of queries	64	111	30
4. Number of queries with more than 5 relevant documents	41	73	30
5. Average number of relevant documents of a query in 4.	19	43	23
6. Max. number of relevant document in a query	51	155	39
7. Average index term frequency in a document.	53	94	118

Two of the most widely-used measures for effectiveness are recall and precision. Recall is defined as the ratio of the number of relevant documents retrieved to the total number of relevant documents in the collection. Precision is defined as the ratio of the number of relevant documents retrieved to the total number of documents retrieved from the collection. One of the difficulties using recall and precision is in comparing performance of several systems. Let r and p denote recall and precision, respectively. Suppose that ($r1$, $p1$) and ($r2$, $p2$) are the pairs of recall and precision from System 1 and System 2, respectively. The problem arises when $r1 > r2$ but $p1 < p2$ or vice versa. It is hard to tell which system performs better overall. But the recall and precision are still the most widely-used measures in information retrieval. It is known from experiments [Salton, 1986] that there exists an inverse relationship between recall and precision. This means as recall increases, precision decreases and vice versa. To increase recall, it is necessary to bring more documents to the retrieved set, but at the same time the new nonrelevant documents retrieved are likely to decrease precision. Since two measures with the inverse relationship are hard to compare and evaluate the effectiveness of systems, single measures of performance have been proposed. The E-measure [Van Rijsbergen, 1979] is a composite measure which is a combination of recall and precision in a single expression. That is, $E = (1-(1/(\alpha/p+(1-\alpha)/r)))$ where α is a weighting factor which can be adjusted according to the relative importance of recall and precision. In this research, E-measure is selected to compare the performance of different algorithms. It is a widely-used measure and the weighting factor α can be adjusted to control the importance of recall and precision.

To measure the changes in system effectiveness during reformulations, one formulation (denoted by reformulation 0) and five reformulations had been executed. The total number of initial input documents to the algorithms is five (three relevant and two nonrelevant) and randomly selected from the database. The number of feedback documents at each reformulation was set to 10. When

ISSN: 2324-9773

Kim, N., French, J., & Brown, D. (1993). Boolean Query Reformulation with the Query Tree Classifier. 4th ASIS SIG/CR Classification Research Workshop, 123-138. doi:10.7152/acro.v4i1.12615

feedback documents are sent to the algorithms, the documents used by an algorithm at the previous reformulations are excluded from the current feedback documents. If there are no documents to return, feedback documents are selected from the previously retrieved sets which were not used by the algorithm.

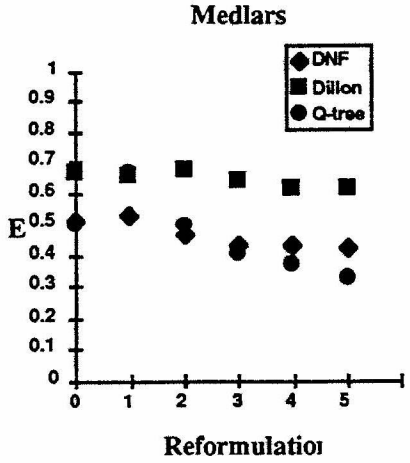


Figure 6 E-measure($\alpha = 0.5$) vs. Reformulation in Medlars

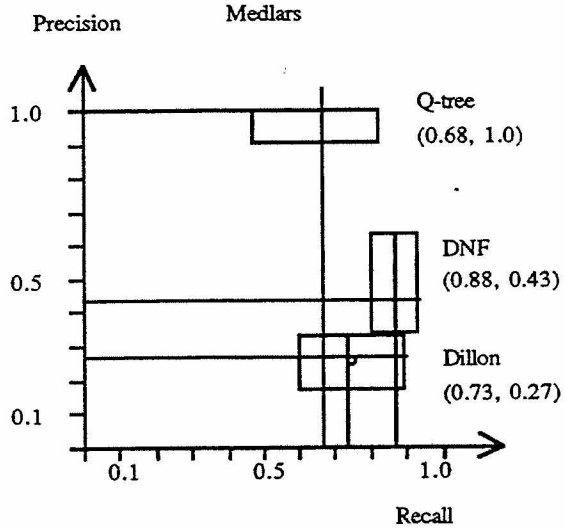


Figure 7 Median Recall and Precision with Interquartile range in Medlars

ISSN: 2324-9773

Kim, N., French, J., & Brown, D. (1993). Boolean Query Reformulation with the Query Tree Classifier. 4th ASIS SIG/CR Classification Research Workshop, 123-138. doi:10.7152/acro.v4i1.12615

To reduce the number of graphs and tables for performance outputs of the algorithms, the weighting factor α of the E-measure is set at 0.5, and statistical analyses (e.g., significance tests) were performed with performance at the 5^{th} reformulation. And only the results of the test set, Medlars are reported here in detail. For more detailed analysis, see [Kim, 1993].

In the Medlars test set, the DNF and query tree methods achieved steady improvement (see Figure 6), but there was not much improvement for Dillon's method during reformulations. The query tree started to outperform the DNF method at the later reformulations (i.e., 3, 4 and 5) and the mean difference got larger. Performance of the query tree at reformulation 1 was worse than that from the previous query, but improved for the rest of the reformulations.

A limitation of a single measure such as the E-measure is that it does not reveal recall and precision. In this research, a pair of median recall and precision is plotted along with its interquartile ranges (Figure 7). The interquartile range contains the middle 50 percent of the scores. The lowest and highest 25 percent of the scores lie outside the interquartile range. When the shape of a frequency distribution is not symmetric, it is preferable to report the median as the average along with a measure of variation, such as the interquartile range [Tashman and Lamborn, 1979]. The main purpose of this graph is for visual inspection, i.e., to show an evaluator the region of recall and precision pairs generated by each algorithm. In the recall and precision graph (Figure 7), the DNF method gained the highest median recall (0.88); and the query tree achieved the highest median precision (1.0). The DNF and Dillon's methods gained higher recall than the query tree method; on the other hand the query method outperformed the other methods in precision.

The significance tests (Wilcoxon signed-rank test [Chao, 1974]) on paired E-measures of two algorithms are provided in Table 2. The tests were carried out at three different α levels: 0.33, 0.5, and 0.66. At each level, two pairs of algorithms, (query tree, DNF) and (query tree, Dillon) were tested. For all three cases, the query tree algorithm has a larger number of favored queries than the others do. As the weighting factor α gets higher, which means more weight to precision, the number of favored queries increases. This phenomenon is from high precision in the query tree algorithm. The more weight there is to the precision, the lower the E-measure gets. (A lower E-measure implies improvement.) In comparisons between the query tree and the DNF method, the p values for the Wilcoxon signed-rank were all lower than 0.05 except when $\alpha = 0.33$ (i.e., $p = 0.5071$). And the query tree achieved between 3 to 38% less of the DNF's E-measure. Overall, the query tree performed better than the DNF at reformulation 5. Between the query tree and Dillon's method, the differences were all statistically very significant ($p \leq 0.0003$).

5. CONCLUSIONS

In this research, Boolean query formulation was formalized as a problem of finding a query which minimizes the costs of misclassifying documents to a wrong class (i.e., retrieving nonrelevant documents and not retrieving relevant documents). The query tree algorithm, which is derived from CART (Classification and Regression Trees) was developed to formulate an effective Boolean query in a short time (its query formulation time took 0.27 second on average in Sun SparcStation 2 with 64 MBytes of main memory and a 64 KBytes of cache while the DNF and Dillon's methods took 0.57 and 1.10 seconds, respectively). For other experiments (CACM and

ISSN: 2324-9773

Table 2 Statistical Analysis of Performance of the Algorithms in Medlars
(Q = query tree; D = DNF method; L = Dillon's method)

System pairs	Medlars at the 5th reformulation							
	Mean		Differ.	Improv.	Query favoring			Signif. Test
a b	a	b	b - a	b-a/b	a	b	=	Wilcoxon
$= 0.33$								
T > D	0.36	0.37	0.01	3%	19	11	0	0.5072
T > L	0.36	0.56	0.20	36%	24	6	0	0.0003
$\alpha = 0.50$								
T > D	0.33	0.42	0.09	21%	22	8	0	0.0276
T > L	0.33	0.62	0.29	47%	22	8	0	< 0.0001
$\alpha = 0.66$								
T > D	0.29	0.47	0.18	38%	25	5	0	0.0005
T > L	0.29	0.67	0.38	57%	28	2	0	< 0.0001

CISI), the query tree also outperformed the current algorithms at the 5th reformulation. Overall, the query tree's performance improved steadily during reformulations while the other methods showed small or no improvement. The query tree had always achieved the highest average precision than any other method.

6. REFERENCES

Araya, J. E. (1990), *Interactive Query Formulation and Feedback Experiments in Information Retrieval,* Ph.D. Dissertation, Cornell University.

Bookstein, A. (1985), "Probability and Fuzzy-Set Applications to Information Retrieval," *Annual Review of Information Science and Technology,* vol.20, pp. 117-151.

Breiman, L. et al, (1984), *Classification And Regression Trees*, Wadsworth Inc.

Chao, L. L. (1974), *Statistics Methods and Analyses*, McGraw-Hill.

Dillon, M. and Desper, J. (1980), "The Use of Automatic Relevance Feedback in Boolean Retrieval Systems," *J. of Documentation*, vol.36, no.3, pp. 197-208

Dillon, M., Ulmshneider, J. and Desper, J. (1983), "A Prevalence Formula for Automatic Relevance Feedback in Boolean Systems," *Information Processing & Management*, vol.19, no.1, pp. 27-36.

ISSN: 2324-9773

Kim, N., French, J., & Brown, D. (1993). Boolean Query Reformulation with the Query Tree Classifier. 4th ASIS SIG/CR Classification Research Workshop, 123-138. doi:10.7152/acro.v4i1.12615

Frants, V. I., and Shapiro, J. (1991a), "Control and Feedback in a Documentary Information Retrieval System," J. *of Amer. Soc. Info. Sci.*, vol.42, no.9, pp. 623-634.

Frants, V. I., and Shapiro, J. (1991b), "Algorithm for Automatic Construction of Query Formulations in Boolean Form," *J. of Amer. Soc. Info. Sci.*, vol.42, no.1, pp. 16-26.

Kim, H. Nam, (1993), "A *Classification Approach to the Automatic Reformulation of Boolean Queries in Information Retrieval,"* Ph. D. Dissertation, University of Virginia.

Radecki, T. (1982), "Incorporation of Relevance Feedback into Boolean Retrieval Systems," in *Proceedings of Research and Development in Information Retrieval,* pp.133-150.

Salton, G. and McGill, M. J. (1983), *Introduction to Modern Information Retrieval,* McGraw-Hill, Inc.

Salton, G., Fox, E. A. and Voorhees (1985), "Advanced Feedback Methods in Information Retrieval," *J. of Amer. Soc. Info. Sci.*, vol.36, no.3, pp. 200-210.

Salton, G., Voorhees, E. and Fox, E. A. (1984), "A Comparison of Two Methods for Boolean Query Relevancy Feedback," *Information Processing & Management*, vol.20, no.5/6, pp. 637-651.

Van Rijsbergen (1979), Information Retrieval, 2nd ed., Butterworths, London.

Vernimb, C. (1977), "Automatic Query Adjustment in Document Retrieval," *Info. Proc. & Management.*, vol.13, pp. 339-353.

ISSN: 2324-9773