Optimal Information Retrieval with Complex Utility Functions

Tao Tao, and ChengXiang Zhai
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL 61801
Email: {taotao,czhai}@uiuc.edu

Abstract. Existing retrieval models all attempt to optimize one *single* utility function, which is often based on the topical relevance of a document with respect to a query. In real applications, retrieval involves more complex utility functions with preferences on several different dimensions. In this paper, we present a general optimization framework for retrieval with complex utility functions. A query language is designed according to this framework to enable users to submit complex queries. We propose an efficient algorithm for retrieval with complex utility functions based on the a-priori algorithm. As a case study, we apply our algorithm to a complex utility retrieval problem in distributed IR. Experiment results show that our algorithm allows for flexible tradeoff between multiple retrieval criteria. Finally, we study the efficiency issue of our algorithm on simulated data.

1 Introduction

The task of information retrieval is to retrieve documents that are relevant to a query submitted by a user. The notion of *relevance* is central to the development of retrieval methods. Different retrieval methods differ mainly in the way of modeling relevance. Vector space models (Salton et al., 1975; Salton and McGill, 1983; Salton, 1989), classic probabilistic models (Robertson and Sparck Jones, 1976; van Rijbergen, 1977; Turtle and Croft, 1991; Fuhr, 1992), and the recently developed language modeling approaches (Ponte and Croft, 1998; Hiemstra and Kraaij, 1998; Miller et al., 1999; Berger and Lafferty, 1999; Song and Croft, 1999; Lafferty and Zhai, 2001; Zhai and Lafferty, 2001b; Lavrenko and Croft, 2001; Zhai and Lafferty, 2001a; Zhai and Lafferty, 2002; Lafferty and Zhai, 2003) are among the major existing retrieval models, and have all been shown to perform well empirically. However, they all only support simple queries that involve one single simple utility function based on topic relevance as if whether a document is *topically* relevant to the query is all a user cares about.

Recently, the risk minimization framework (Zhai, 2002; Zhai and Lafferty, 2003) attempts to go beyond "topical relevance only" retrieval by optimizing a utility function that can potentially contain other retrieval factors such as redundancy and readability. However, the retrieval criterion remains to optimize one *single* utility function. For example, in subtopic retrieval, while both redundancy and topic relevance are part of the utility function, they have to be combined to form a single retrieval utility function (Zhai et al., 2003). As a result, there is no way for a user to specify such complex preferences as "retrieving as many relevant documents as possible but limiting the redundancy level to be under certain fixed threshold".

This is a serious limitation, as in any real retrieval applications, a user would almost alway care about multiple factors and the retrieval preferences are often quite complicated, involving different bounds on different factors. For example, a user who wants to find information in a distributed peer-to-peer environment often cares about the response time and communication cost besides relevance. A user using a "pay-per-search" system clearly cares about the money cost as well as relevance. Thus, in general, a retrieval system has to handle multiple factors (aspects) and cope with flexible preferences on multiple utility functions. We thus need to study the problem of **information retrieval with complex utility functions**. Existing retrieval frameworks are insufficient for handling such a complex utility retrieval problem, since they do not have a mechanism to handle multiple utility functions.

Retrieval with complex utility functions poses several special challenges that do not exist or do not matter in a traditional retrieval model: First, how can we define complex utility functions, and further enable a user to submit a query involving preferences on multiple utility functions? Second, a user's preferences on different utility functions may be inconsistent. How can we resolve any conflict? Finally, finding a solution for retrieval with complex utility functions is NP-hard for any non-trivial cases. How can we develop efficient algorithms or even efficient approximate algorithms?

In this paper, we address these challenges. We propose a *cost optimization framework* for such a *complex utility function retrieval* problem. A query language is designed according to this framework to enable a user to submit complex queries. The basic idea of this framework is to formulate the problem as a constrained optimization problem. We distinguish two types of retrieval preferences: (1) The user's main interests. This part is what users expect to maximize. We call such utility functions "objective utility functions". For example, "relevance" is almost always an objective utility function since a user would always like to find as much relevant information as possible. (2) User's cost, *i.e.* a user's tolerance on some aspects of utilities. For example, a user who can tolerate up to 30 seconds delay in the response time can be regarded as having an upper bound of 30 for the response time utility function. We call such utility functions "constraint utility functions".

Our framework treats the retrieval problem as an optimization problem with a "maximization part" and a "constraint part", corresponding to the two different types of retrieval preferences, respectively. Our goal is to select a subset of documents (or any other information items) that can maximize the objective utility function subject to the bounds on all the constraint utility functions.

Based on this optimization framework, we propose a general query language to allow a user to pose queries to specify preferences on complex retrieval utility functions. We study two special properties of the utility functions – monotonicity and additivity, and propose an efficient algorithm for retrieval with complex utility functions that can work for any utility functions satisfying monotonicity. As a case study, we apply our algorithm to a complex utility retrieval problem in distributed IR. Experiment results show that our algorithm allows for flexible tradeoff between multiple retrieval criteria. We study the efficiency issue of our algorithm on simulated data. The optimization framework is more general than any existing retrieval framework in that it allows for retrieval with multiple utility preferences. The proposed retrieval algorithm can be potentially applied to many different retrieval applications where complex utility functions are involved.

2 A Utility Optimization Framework

In this section, we present the utility optimization framework. In this framework, we use utility functions to model all the factors that a user is concerned about, and formulate the problem of information retrieval with complex utility functions as an optimization problem based on the utility functions. The framework naturally suggests a general query language that a user can use to express complex information need.

2.1 Utility functions

Suppose $DB = \{b_1, ..., b_N\}$ is a set of N information items. b_i is an information item, which can be a document, a passage, a sentence, or even a term. DB is not necessarily a single database; it could be a union of several distributed databases.

In a typical retrieval task, a user U would pose a query Q to express the information need, and the system would answer the query by returning a subset of items from DB, $D_Q = \{d_1, ..., d_n\}$, where $d_i \in DB$. The goal is to return only *relevant items* w.r.t. the query , *i.e.,* items that can help satisfy the user's information need. Which D_Q counts as the *best* answer is inherently subjective, and clearly depends on the user's retrieval preferences, which, in general, may involve many different utility factors. For example, topical relevance is always an important factor – presumably the main goal of the user is to find relevant information. This requirement is often described through a keyword query. Most existing retrieval systems only deal with topical relevance and return retrieval results based solely on a short keyword query. In reality, a user is almost always also concerned about other factors, such as the communication cost in a network environment and the readability of the returned items in the case of a child user. For each such factor or aspect that matters for the user, we assume there exists a utility function. Let $\Psi_u = \{\psi_1, \psi_2, ..., \psi_k\}$ be all the utility functions interesting to a user U. Each utility function ψ_i measures a candidate subset of information items from one

perspective. Formally, it takes the query Q and a candidate subset of items D_Q as input and generates a real value: $\forall \psi \in \Psi_u$, $\psi : Q \times 2^{DB} \rightarrow R$, where 2^{DB} is the power set of DB. It is reasonable to assume that these utility functions are independent of each other, since if two utility functions depend on each other, we can simply combine them and obtain a new utility function.

We distinguish two types of utility functions – constraint utility functions and objective utility functions – based on how the user's retrieval preferences are formulated on a utility function. A constraint utility function is one on which the user would impose a (upper) bound constraint, thus a candidate set of items need only to satisfy the bound constraint. An objective utility function is one that needs to be maximized; we thus prefer a candidate item set that gives a maximum value for such a utility function. Note that we do not intend to make any distinction on the utility function itself; in both cases, when applied to a candidate set, the utility function would give a real value. The distinction is more on the role a utility function plays in the optimization problem. This will be further discussed later.

In general, the goal of retrieval is to find a subset of items D_Q that can maximize the values of all the utility functions. The relation between selecting information items and ranking them is a subtle issue worth discussing. On the surface they may appear as two different retrieval strategies. However, they are actually closely related and complement with each other well. First, while ranking information items has been well justified as an optimal retrieval strategy for a single utility function (Robertson, 1977; Zhai, 2002), it in fact implies a "prior step" of item selection, in which we simply select *all* the items. Second, in the case of multiple utility functions, it is necessary to explicitly address the item selection step, since we have additional retrieval constraints. In the following section, we discuss this general two-step retrieval process in some detail.

2.2 The two-step information retrieval process

A retrieval problem can be treated most generally as a decision problem (Zhai, 2002). Depending on how we define the decision space, we may end up with different retrieval strategies. One general formulation involves choosing a subset of items D and a presentation strategy for presenting D to the user (Zhai, 2002). This implies that, theoretically, retrieval involves a two-step process with the first step choosing a subset of items D and the second presenting D appropriately. Given a set of selected items, ranking them is generally better than presenting them randomly. Indeed, given a single utility function, which can be a combination of several aspect utility functions, ranking can be shown to be the optimal presentation strategy under certain assumptions (Robertson, 1977; Zhai, 2002). Therefore ranking tends to be the only presentation strategy.

In general, we may have *different* criteria for selecting the item subset and for ranking. This distinction is not important in traditional retrieval approaches, because it only deals with one single utility function, which essentially serves as the same criterion for both item selection and item ranking. In the case of multiple utility functions, however, different utility functions may be involved in these two steps. For example, in distributed information retrieval, the communication cost is very important for item selection, but once the items are selected, the communication cost is almost irrelevant when ranking the selected items. Using distinct utility functions for selection and ranking occurs frequently in database search. For example, a user can use "employee.ID=manager.ID" for record selection and rank the records selected based on names.

It is important to note that the separation of selection and ranking is mainly for convenience of analysis; in a real algorithm, they could be integrated together. Indeed, a ranking strategy may help selecting information items more efficiently, as will be discussed in the algorithm part.

In this paper, we focus on the item selection problem, which is important and non-trivial for multiple utility functions[1].

2.3 Item selection as constrained optimization

Given a set of utility functions, we can formulate the problem of retrieval with complex utility functions as a *constrained optimization* problem. The constraint utility functions along with the user's bounding preferences serve naturally as the constraints, while the objective utility function serves as the optimization objective function. Formally, we use a standard optimization formulation to express our need of selecting a subset X from DB:

[1] Ranking is typically based on a single utility function, and is essentially similar to what most existing retrieval systems are doing.

$$
\begin{aligned}
\textbf{maximize}: \quad & h(X) \\
\textbf{subject to}: \quad & h_1(X) \leq c_1 \\
& h_2(X) \leq c_2 \\
& \ldots \\
& h_m(X) \leq c_m
\end{aligned}
$$

where $h \in Psi_u$ is the objective utility function, $h_i \in \Psi_u \; i = 1, ..., m$ are constraint utility functions, and c_i's are the cost bounds.

2.4 Query language

A single keyword query is no longer sufficient to specify the information need when multiple utility functions are concerned. In this section, we propose a general query language that would allow a user to express complex information need. The proposed query language is based on the two-step retrieval process and the optimization problem setup.

First, we define SELECT and RANKBY clauses. They correspond to the item selection and ranking in the two-step retrieval process. Second, within the SELECT clause, we introduce an ABOUT clause to specify the keywords, and two additional clauses MAXIMIZE and SUBJECTTO to define retrieval preferences in the optimization formalism. Thus, a complete query would include both SELECT and RANKBY, and in the SELECT clause, there would be three parts – ABOUT, MAXIMIZE and SUBJECTTO. The following is a query example.

Suppose a user is interested in finding documents about information retrieval. The following is a possible query that involves three utility factors – relevance, response time, and redundancy.

SELECT	document	X
	ABOUT	information retrieval
	MAXIMIZE	relevance(X)
	SUBJECTTO	responsetime(X) < 30
		redundancy(X) < 5
RANKBY	length(X)	

In this query, the SELECT clause contains a keyword part followed by a MAXIMIZE clause and a SUBJECTTO clause to indicate that the goal is to maximize the amount of relevant information under two constraints – the response time is within 30 seconds and the level of redundancy is less than 5%. The RANKBY clause indicates that the selected documents should be ranked by length.

Note that the proposed query language can be regarded as an extension of the regular keyword query language. Indeed, a simple keyword query "information retrieval" is equivalent to the following query in our query language:

SELECT	document	X
	ABOUT	information retrieval
	MAXIMIZE	relevance(X)
RANKBY	relevance(X)	

3 Properties Study of utility functions

The central question for item selection is how we can select items efficiently. Without making further assumptions, it appears that the only possible way to find an exact solution is through exhaustively enumerating all possible choices from the power set of information items based on the constraint utility functions and then selecting the best one based on the objective function. Clearly, this naive algorithm is too time consuming and infeasible for any practically interesting retrieval problem. It is thus necessary to consider the characteristics of utility functions, and to develop more efficient algorithms.

Without losing generality, we assume all utility functions are non-negative.

3.1 Monotonicity

A utility function $\psi \in \Psi$ is said to satisfy the monotonicity property if for all $X, Y \subseteq DB$, $\psi(X \cup Y) \geq \psi(Y)$. ($\psi(X \cup Y) \geq \psi(X)$ holds as well because of symmetry.) The property is called monotonicity because the function values are monotonically increasing whenever more items are added into the original set.
It is interesting but perhaps not surprising that most reasonable utility functions in a retrieval problem satisfy this monotonicity property. Taking communication cost for example, if a new item is added into a selected item set, it will definitely increase the communication cost because the retrieval system has to transfer one more item to a user via the network.
Monotonicity is a useful property. Given a utility function ψ and a constraint bound c, the constraint part in our optimization framework is expressed as $\psi(X) \leq c$. If ψ has monotonicity property, it basically means that whenever an item set cannot satisfy this constraint, its super set would not satisfy the constraint either.

Example 1 *There are three items: b_1, b_2, b_3. We know $\psi(\{b_1, b_2\}) \leq \psi(\{b_1, b_2, b_3\})$ because of monotonicity of function ψ. Therefore, from $\psi(\{b_1, b_2\}) > c$, we can infer $\psi(\{b_1, b_2, b_3\}) > c$ without computing the function value.*

3.2 Decomposition and additivity property

In most retrieval applications, utility functions are often decomposable, i.e., the utility function value over a large item set is a composition of values of the function over smaller item sets. For example, the communication cost utility function is additive – the cost of a large set of items is the sum of the cost of each individual item in the set. Redundancy measure is another example. It is usually defined on a pair of items. The redundancy value of a set can be defined based on pairwise redundancy values (Zhai et al., 2003; Zhai, 2002). These observations drive us to study the decomposition and additivity properties of utility functions.
Formally, a utility function $\psi \in \Psi$ is *decomposable* if it can be decomposed as a summation of *interaction utility functions* of different degrees. A function is called an i^{th} *degree interaction utility function* if it captures the interaction between the items in an item set of size i. A decomposable utility function ψ can thus be written as

$$\psi(X) = \sum_{i=1}^{N} f_i(X)$$

where $f_i(X) = \pi\{f_i(d_1, ..., d_i) | d_1, .., d_i \in X\}$. $f_i(d_1, ..., d_i)$ accepts i items as its input values, and output the utility values. π is an operator to generate the function value for a large item set from values of smaller sets. For example, π can be a summation operator, thus $f_i(X) = \sum_{d_1,...,d_i \in X} f_i(d_1, ..., d_i)$. If π is a maximum operator, the function $f_i(X)$ becomes $f_i(X) = \max_{d_1,..,d_i \in X} f_i(d_1, ..., d_i)$. The selection of operator π is important because not all operators satisfy the monotonicity property requirement(section 3.1). *summation*, *max*, and *min* are examples of monotonic operators, but the *average* operator is not. In this paper, we only consider monotonic operators.
Although in general, a utility function to be decomposed into a summation of interaction functions of multiple degrees, a particular utility function usually only involves a single degree of interaction. For example, money cost usually involves only the 1-degree interactions while redundancy can be defined on 2-degree interactions. Of course, some utility functions may involve multiple-degree interactions. Take money cost for example again. The user pays some amount of money for each information item. This is 1-degree of interaction. However, if the user receives some discount when buying more than one item together, then the discount must be modeled through higher-degree interactions.
The most common utility functions appear to only involve the first and second degree interactions. The first degree interaction makes the classic independence assumption: items are independent of each other. Most utility functions fall into this category. A typical second degree interaction is redundancy.

4 Efficient complex utility retrieval algorithms

In this section, we study algorithms for complex utility retrieval, i.e., algorithms for finding the solution to the constrained optimization problem. We first formulate the optimization problem as a 0-1 integer programming

problem. We then present two relatively efficient algorithms for solving the optimization problem. One is an exact algorithm based on depth-first search with branch-cutting. The other is a greedy algorithm that finds an approximate solution. Without losing generality, we use summation as the operator π throughout the rest of this paper. It can be replaced by any other operator that satisfies monotonicity.

4.1 Item selection as 0-1 integer programming

When the utility functions are decomposable, we can further refine the optimization problem as a classic 0-1 integer programming problem.
Let us first introduce a variable x_i for each item b_i in DB. The value of x_i is either 0 or 1. $x_i = 1$ if b_i is selected, *i.e.* $b_i \in X$; otherwise, $x_i = 0$. Given a function h, assuming it involves k degree interaction, it can be expressed by:

$$h(X) = \sum_{i_1, i_2, .., i_k} h(b_{i_1}, b_{i_2}, ..., b_{i_k}) x_{i_1} x_{i_2} ... x_{i_k}.$$

The product $x_{i_1} x_{i_2} ... x_{i_k} = 1$ iff every $x_{i_j} = 1$ $(j = 1...k)$. This means that the interaction of items $b_{i_1} ... b_{i_k}$ is "turned on" only if all these items are selected.
Clearly, selecting the optimal subset is now equivalent to finding optimal values for b_i's. The following example illustrates the integer programming formulation more clearly.

Example 2 *Suppose the objective function is the relevance measure r, which is a 1-degree interactive utility function, and we have two constraint utility functions: communication cost m and redundancy cost y, which are 1-degree and 2-degree interactive utilities with bounds c_1 and c_2, respectively. The integer programming formulation is:*

$$\begin{aligned} \textbf{\textit{maximize}}: \quad & \sum_{i=1}^{N} r(b_i) x_i \\ \textbf{\textit{subject to}}: \quad & \sum_{i=1}^{N} m(b_i) x_i \leq c_1 \\ & \sum_{i,j=1}^{N} y(b_i, b_j) x_i x_j \leq c_2 \end{aligned}$$

N is the size of DB.

We thus see that the item selection problem can be formulated as a 0-1 integer programming if all utility functions can be decomposed into a certain degree of interaction functions.
Unfortunately, 0-1 integer programming is proved NP-complete, and no exiting polynomial time algorithms is able to solve it. We thus explore alternative ways to solve the optimization problem. Next we present two relatively efficient algorithms: one is depth first searching with branch cutting, which solves the problem exactly. It runs reasonably fast in practice although its worse-case time complexity is still exponential. The other algorithm is a greedy algorithm, which solves the problem approximately, but with a much faster speed.

4.2 Depth first search with branch cutting

This algorithm is motivated from a well-known data mining algorithm called a-priori algorithm (Agrawal and Srikant, 1994). The main difference is that we perform a depth-first search, rather than a breadth-first search as adopted by the a-priori algorithm. The depth-first strategy is more suitable for our problem as will be discussed further. Both algorithms are based on the assumption that all utility functions satisfy monotonicity.

4.2.1 Sorting of objective functions

In section 2, we formulate the complex retrieval problem as an optimization problem. There are two parts: an objective part and a constraint part. Even if the complex retrieval problem introduces several constraints, the

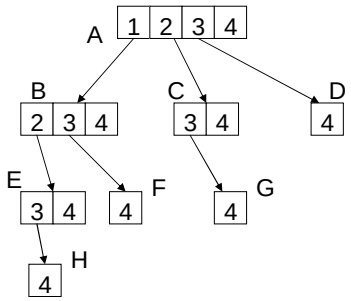

Figure 1. A complete array tree.

primary task is still to maximize the objective function, which is often the relevance measure between a query and retrieved documents. In other words, we are mainly interested in highly relevant items. This observation drives us to consider starting our searching from the top relevant documents. Before selecting documents, we first perform the relevance retrieval procedure to retrieve and rank documents in the descending order of relevance scores. Doing this has two benefits:

1. It allows us to start searching from highly relevant documents. In a later part of this section, we will show that it would also allow us to stop the searching process earlier before finishing every searching branch.

2. It allows us to remove non-promising items from DB heuristically in the beginning. As we mentioned earlier, the user's primary interest is in the objective part. They are unlikely to have interest in a very low-ranked item (e.g., no. 10000 relevant item) even if it can contribute to our final optimal solution. Therefore, another approximation strategy can be to consider only the top k ranked documents, where k is reasonably large, for example 1000.

4.2.2 Data structure: array tree

An array tree is a data structure used in our depth first search. It is motivated by the hash tree in (Agrawal and Srikant, 1994). The array tree is similar to the hash tree but is simpler because we do not need hash search operations as performed in (Agrawal and Srikant, 1994).

Figure 1 illustrates a complete 4-item array tree; the IDs of the 4 items are 1 to 4, respectively. There are 8 nodes marked from A to H. Each of them is attached to an array, including a single item ID in each bucket. Every bucket also has a link pointing to its child node. The root node, A, includes all IDs of the array; other nodes are generated from their parent nodes by fixing some "prefix ID string", and only include the remaining IDs from the parent node. For example, node B is generated from node A by assuming 1 is selected. Therefore its IDs can only be from 2 to 4. Similarly, node C has only IDs 3 and 4 because its parent node is 2.

Although nodes H, F, G, and D look identical, they in fact represent choosing different subsets. From this example, we can see that a depth first traversal of our array tree can enumerate all elements in the power set of items in the root node. At any point in the depth first search, the path from a root bucket to a bucket of the current node represents a set enumeration. For example, the first bucket of G stands for $\{2, 3, 4\}$, and the second bucket of C stands for $\{2,4\}$.

It is worthwhile to mention that we do not actually construct a complete tree and traverse it afterward. Instead, every node is dynamically created during the traversal, and deleted after visiting all of its subtrees.

A brute-force depth first search would essentially result in an exhaustive search, and not save any time. Fortunately, we can apply several heuristics to cut unnecessary branches and make the algorithm efficient in practice.

4.2.3 Branch cutting

The monotonicity property shows the following: whenever an item set violates a constraint, all of its supersets must violate the constraint as well. Therefore, we do not need to exhaustively visit every branch in the array tree.

For example, in Figure 1, if we know set $\{1, 2\}$ violates a certain constraint, we would know that all its supersets must violate the same constraint. This further leads us to observe that we do not need to visit nodes E and H at all because they represent all supersets of $\{1, 2\}$. In this way, there are only nodes A,B,C,D,F, and G left from the complete tree to visit. By the same principle, other branches can be pruned as well.

In practice, after creating each node, we perform constraint checking to ensure that the constraint bounds be satisfied for each bucket. Any violated items will be removed from the node. Following the example above, if we find that the 2 in node B (i.e., item set $\{1, 2\}$) violates a constraint, we would remove 2 from B immediately. Since the children nodes are generated from their parent node, E and H would thus not be generated.

4.2.4 Early stop conditions

Our primary task is to find a set to maximize the objective part under several constraints. Therefore, we do not need to proceed once we know that the optimal solution has already been visited in the middle of traversing the tree. In this section, we study these early stopping heuristics. We assume the documents are arranged in the descending order of the objective function (section 4.2.1). We further assume that $k = |X|$, i.e., the number of information items selected from DB is predefined. For example $k = 10$.

Claim 1 *By depth first search, the first k-item set, qualified for all constraints, is* not *necessary the optimal one.*

It is easy to find a negative example to prove this claim. For example, $k = 3$, item set $\{1, 2, 5\}$ should be visited before $\{1, 3, 4\}$ in the depth-first search path. If the objective values for item 1 to item 5 are 10, 9, 8, 7 and 5, respectively. Item set $\{1, 3, 4\}$ would have an objective value of 25, while $\{1, 2, 5\}$ would have 24. This claim tells us that the first solution is not necessarily the optimal one. The question is then, do we need to complete the depth-first search to find the best solution? The answer is "no"! For example, in Figure 1, if $k = 2$ and $\{1, 2\}$ satisfies all constraint bounds, we know immediately that this is the best solution because no other two-item set can have a larger objective value than this one due to the descending order of objective values. We thus have the following claim:

Claim 2 *If either of the following conditions is satisfied, we can stop the depth-first search on the current branch or even stop the whole searching process without missing the best solution provided that we have already discovered several k-item sets satisfying all constraints. Specifically, let M be the largest objective value of these sets. There are two conditions under which we can stop the current search branch:*

- *Items left in the current nodes are not enough to construct k items.*
- *Although there are enough items, the best objective value of any item set is at most M.*

The two conditions are intuitive, and are illustrated in Figure 1 as follows.

First condition: If we aim at a 3-item set, we would not need to create node D because we can never obtain any item set with more than two items when starting from bucket 3 in the root node. More generally, this condition can be checked as follows. Given bucket b in node T, if T's depth (in the tree) plus the number of buckets after b in T is less than k, then we do not need to extend b or any bucket after b. In the example above, $k = 3$, T is A, and b is bucket 3 in A. A's depth is 1, and the number of buckets after b is 1. Since $1 + 1 < 3$, we can finish visiting A.

Second condition: If we aim at a 2-item set and find that item 2 in node B survives constraint bound checking, then we can be sure that $\{1, 2\}$ is the best solution since no other item set found later would further improve the objective function value. More generally, assuming the current best objective function value is M, we are in bucket b of node T, the accumulative objective function value for the path from the root to b is m, b's

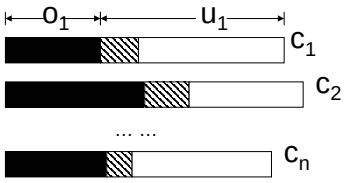

Figure 2. An illustration of the greedy algorithm.

objective function value is o, and T's depth is d, if $m + (k - d) * o < M$, we can prune all the left branches from T.

Note that this algorithm is an "any time" algorithm. Indeed, whenever the algorithm needs to terminate, it can just return the best solution it has found so far. It thus allows for a flexible tradeoff between the quality of solution and the amount of time used for searching. The more time is spent in searching, the better the approximate solution is. This tradeoff will be further discussed in Section 5.

4.3 Greedy algorithms

A greedy algorithm first selects a single best item, and then gradually adds new items into this set by choosing a new best item given the currently selected items in each step. The key task is to choose the next best item. In our problem, there are multiple constraint utility functions. Due to monotonicity, a new item always increases the cost when it is added into a set. Thus, the next selected item should preferably be decided based on the most "critical" utility function. A "critical" utility function is one for which the current item set is close to exceeding the cost bound. We use Figure 2 to illustrate this. Assume we have n different utility functions, and their bounds are c_i, $i = 1, ..., n$, respectively. Black area (o_i) stands for the utility "used/occupied" by previously selected items , and other areas (u_i) are the utilities left. $o_i + u_i = c_i$. If a new item is selected, it occupies the striped areas(t_i). Assume its gain in objective function value is b. We want to select an item, which will occupy a minimum percentage of left area, but has maximum objective function gain. Formally, we use the following equation to compute a score for each item and select the next item by maximizing this score.

$$\max\{\frac{b}{\max_i \frac{t_i}{o_i}}\}$$

Clearly, if $\max_i \frac{t_i}{o_i} > 1$ for an item, the item cannot be considered.

5 Experiments

In this section, we present the results of experiments to examine the effectiveness and efficiency of our algorithm.

We first discuss an interesting application of our general framework to distributed IR. We then show the effectiveness of our framework on a distributed document set. As expected, our algorithm allows for a tradeoff between the traditional relevance measure and some new constraints. We also use simulated data to study the efficiency of our algorithms. For comparison, we use the a-priori algorithm as our baseline algorithm. Finally, we compare our depth-first search with the greedy algorithm. All our experiments are done on a Dell Precision workstation with 2.4GHz cpu speed and $1GB$ memory.

5.1 An application in distributed information retrieval

Distributed information retrieval is a natural example of the complex utility retrieval problem. In a distributed environment, the collection of documents are distributed among many servers in distinct geographic locations. Existing research in distributed information retrieval focuses its attention on two algorithmic challenges: collection/resource selection and merging/fusion (Si and Callan, ; Nottelmann and Fuhr, ; Callan

parameters	relevance	comm. cost	redundancy
without constraint	498.0	221.4	754.8
300,450	419.7	157.4	441.9
250,400	411.8	147.3	391.7
200,350	397.4	143.0	344.4

Table 1. Performance on AP data.

et al.,). The retrieval decision has so far been only based on topical relevance. In reality, the communication cost is another important factor that matters to the user. Because of the overhead of fetching documents through the network and different collections can easily duplicate information items, the issue of redundancy also becomes more significant than in a centralized retrieval situation. Therefore, users in distributed information retrieval environment are facing at least all the following three aspects of utilities: (1) topic relevance; (2) communication cost (or response time); and (3) redundancy. A user's goal is to retrieve as much relevant information as possible while minimizing the communication cost and redundancy. Generally speaking, redundancy is not independent on communication cost because its computation relies on information transition between servers. However, redundancy computation is non-relevant to queries, and therefore can be computed off-line. When a real query is submitted by a user, a server does not need to fetch documents from other servers any more. We will assume them to be independent.

5.2 Algorithm effectiveness

We use the Associated Press (AP) data set available through TREC (Voorhees and Harman, 2001) as our document database, which has $164,597$ news articles. We use TREC topics 101 - 150 as our experiment queries (Voorhees and Harman, 2001).

We use the relevance score, computed from a language modeling retrieval approach (Zhai and Lafferty, 2001a; Zhai and Lafferty, 2002), as the objective function. The two constraints are communication cost and redundancy. Communication cost is simulated, and the redundancy between each pair of documents is computed by the redundant words of two documents. Redundancy of a set of documents is defined by the sum of pairwise redundancy of documents within this set. In our problem, the communication cost is of 1-degree interaction, and redundancy is of 2-order interaction. For simplify constraint bound setting, we normalize all aspects (relevance, communication, and redundancy) ranging from 1 to 100.

We carry out experiments to select 7 from a working set of 200 documents, which can be selected using the objective function [2]. Table 1 shows our results. The second row is the optimal results without any constraints. The other three rows are the results with some constraints, as listed in the first column: the first number is the bound for communication cost, and the second is for redundancy. The second, third, and fourth columns show the experiment results. For example, 419.7 in the second column, third row is the objective function value of the optimal solution under the constraints that the communication cost is below 300 and redundancy is below 450. All these numbers are the average over all 50 queries.

From the table, we can easily observe the tradeoff: we achieve low communication cost and redundancy by losing some relevance.

5.3 Algorithm efficiency

Since the item selection problem is in general NP-hard, it is necessary to study the algorithm's efficiency. We evaluate our algorithm over simulated data sets. The baseline method of brute-force enumeration is a very weak baseline, we thus use the a-priori algorithm as our baseline algorithm, which also uses monotonicity to prune the search space, but with breadth-first search instead of depth-first search. This is a much stronger baseline.

We first evaluate the scalability on the searching depth k, i.e., the number of selected documents by fixing the total number of documents to 200. We use one objective function and two constraint functions. All

[2] We assume that any item with a significantly small objective function value is unlikely interesting to the user.

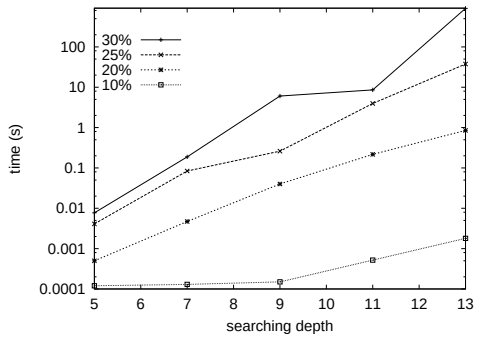

Figure 3. Scalability on the searching depth.

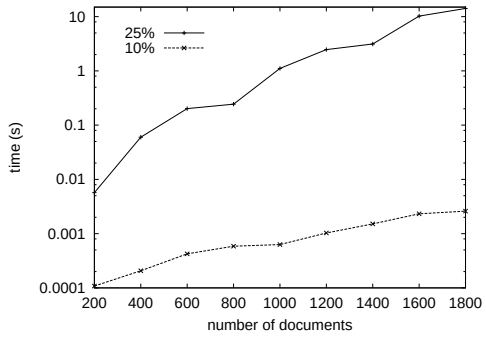

Figure 4. Scalability on the number of documents.

function values are generated by sampling according to a uniform distribution(1-100). We use the percentage of the expected values as the constraint bounds. For example, if searching depth=5, the expected value of 5 documents is $50 \times 5 = 250$. 30% bounds will be $250 * 0.3 = 75$. Figure 3 shows the scalability of our algorithm on searching depth. The x-axis is the searching depth while the y-axis is the running time in log scale. All values are the average over 10 experiments with identical parameter settings. As expected, the plot shows an exponential growth. However, we do observe that our algorithm finishes the experiments on a desktop within just several seconds. On the country, the a-priori algorithm cannot finish the experiments in hours if $k > 7$.

We then evaluate the scalability over the number of documents. We vary the number of documents from 200 to 2000 in Figure 4. It shows sub-exponential property. There are two curves, corresponding to a bound of 25% and 10%, respectively. Both have a searching depth of 6. Again, all values are the average over 10 experiments with identical parameter settings. We also observe that our algorithm outperforms the a-priori algorithm, which again cannot finish in a reasonable time period when the number of documents is larger than 400.

The figures also show that the algorithm cannot completely solve the exponential time complexity problem although it makes the problem more tractable. Thus, an approximate algorithm should be considered for real applications.

5.4 Approximation

In this section, we first evaluate the any-time stopping property of our algorithm. We use 6 simulated sets of data. Table 2 shows a comparison between the running time when the program stops as soon as it has encountered (and thus found) the optimal solution (2^{nd} row), and the running time when the program runs

different data	1	2	3	4	5	6
optimal time	2.53	5.28	0.83	0.48	1.27	0.118
total time	116.98	33.27	175.48	13.52	10.16	11.02
percentage	2.2%	15.9%	0.47%	3.6%	12.5%	1.1%

Table 2. Any time stop.

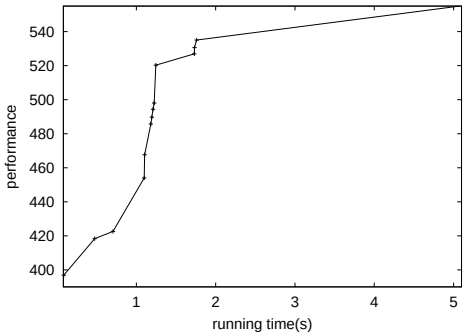

Figure 5. Performance improving over running time.

until it is finished (3^{rd} row). The test is done with 500 documents, searching depth 8, and 25% constraint bounds. The last row shows the percentage of the second row over the third row. It is clear that the program usually encounters the optimal solution much earlier than it finishes. The percentage ranges from around 1% to 15%. Figure 5 shows how the effectiveness (measured by the objective function value) increases as the program runs longer. We only plotted the x-axis up to the time of finding the optimal solution. We see that the effectiveness improves quickly in the beginning, but soon becomes flat although it still increases as the program keeps running.

We compare two approximation strategies using a simulated data set with 500 documents. The query again involves an objective function and two constraint functions. Figure 6 shows a comparison of the two strategies we discussed earlier. Clearly, the greedy algorithm is much faster than the any-time stopping strategy, but it has less room to improve. When running time is reasonably long, any-time stopping can beat the greedy algorithm.

6 Conclusions and future work

In this paper, we define a new retrieval problem with multiple utility functions, and propose a novel framework to address this new retrieval problem. The fundamental difference between our framework and previous ones is that we allow users to express retrieval preferences in *multiple* aspects of utilities. This gives a user much more flexibility and the user's information need can be described more accurately. We formulate the problem as a constraint optimization problem, and design a query language for these complex retrieval tasks.

We study two special properties of the utility functions – anti-monotonicity and additivity, and formulate the optimization problem as a 0-1 integer programming problem. We further propose an efficient algorithm for retrieval with complex utility functions that can work for any utility functions satisfying monotonicity. Although retrieval with complex utilities is generally NP-hard, the proposed algorithm is relatively efficient as shown in our experiments.

As a case study, we apply our algorithm to a complex utility retrieval problem in distributed IR. Experiment results show that our algorithm allows for flexible tradeoff between multiple retrieval criteria. The proposed retrieval algorithm can be potentially applied to many different retrieval applications where complex utility functions are involved.

There are several directions for further exploring and extending the work presented here: (1) Algorithm effi-

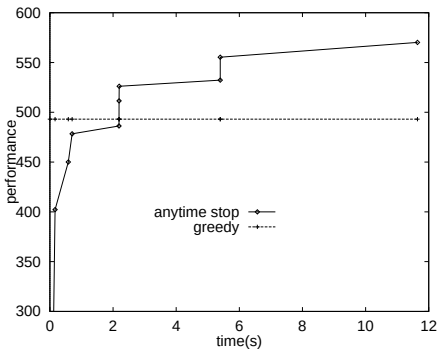

Figure 6. Performance comparison between any time stopping and greedy algorithms.

ciency. Although our algorithm significantly improves the efficiency over brute-force enumeration approach and the pure a-priori algorithm, the running speed is still a concern if the number of documents is too large. More efficient algorithms thus have to be explored. It is especially interesting to explore good approximation algorithms. (2) Utility functions without monotonicity property. In this paper, we have only addressed utility functions with monotonicity, but some utility functions do not satisfy monotonicity. How to cope with such utility functions is an open question.

References

Agrawal, R. and Srikant, R. (1994). Fast algorithms for mining association rules. In Bocca, J. B., Jarke, M., and Zaniolo, C., editors, *Proc. 20th Int. Conf. Very Large Data Bases, VLDB*, pages 487–499. Morgan Kaufmann.

Berger, A. and Lafferty, J. (1999). Information retrieval as statistical translation. In *Proceedings of the 1999 ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 222–229.

Callan, J., Crestani, F., Nottelmann, H., Pala, P., and Shou, X. Resource selection and data fusion in multimedia distributed digital libraries. In *Proceedings of the 2003 ACM SIGIR Conference on Research and Development in Information Retrieval*.

Fuhr, N. (1992). Probabilistic models in information retrieval. *The Computer Journal*, 35(3):243–255.

Hiemstra, D. and Kraaij, W. (1998). Twenty-one at trec-7: Ad-hoc and cross-language track. In *Proc. of Seventh Text REtrieval Conference (TREC-7)*.

Lafferty, J. and Zhai, C. (2001). Document language models, query models, and risk minimization for information retrieval. In *Proceedings of SIGIR'2001*, pages 111–119.

Lafferty, J. and Zhai, C. (2003). Probabilistic relevance models based on document and query generation.

Lavrenko, V. and Croft, B. (2001). Relevance-based language models. In *Proceedings of SIGIR'2001*.

Miller, D. H., Leek, T., and Schwartz, R. (1999). A hidden markov model information retrieval system. In *Proceedings of the 1999 ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 214–221.

Nottelmann, H. and Fuhr, N. Evaluating different methods of estimating retrieval quality for resource selection. In *Proceedings of the 2003 ACM SIGIR Conference on Research and Development in Information Retrieval*.

Ponte, J. and Croft, W. B. (1998). A language modeling approach to information retrieval. In *Proceedings of the ACM SIGIR*, pages 275–281.

Robertson, S. and Sparck Jones, K. (1976). Relevance weighting of search terms. *Journal of the American Society for Information Science*, 27:129–146.

Robertson, S. E. (1977). The probability ranking principle in IR. *Journal of Documentation*, 33(4):294–304.

Salton, G. (1989). *Automatic Text Processing: The Transformation, Analysis and Retrieval of Information by Computer*. Addison-Wesley.

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

Salton, G., Yang, C. S., and Yu, C. T. (1975). A theory of term importance in automatic text analysis. *Journal of the American Society for Information Science*, 26(1):33–44.

Si, L. and Callan, J. Relevant document distribution estimation method for resource selection. In *Proceedings of the 2003 ACM SIGIR Conference on Research and Development in Information Retrieval*.

Song, F. and Croft, B. (1999). A general language model for information retrieval. In *Proceedings of the 1999 ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 279–280.

Turtle, H. and Croft, W. B. (1991). Evaluation of an inference network-based retrieval model. *ACM Transactions on Information Systems*, 9(3):187–222.

van Rijbergen, C. J. (1977). A theoretical basis for theuse of co-occurrence data in information retrieval. *Journal of Documentation*, pages 106–119.

Voorhees, E. and Harman, D., editors (2001). *Proceedings of Text REtrieval Conference (TREC1-9)*. NIST Special Publications. http://trec.nist.gov/pubs.html.

Zhai, C. (2002). *Risk Minimization and Language Modeling in Text Retrieval*. PhD thesis, Carnegie Mellon University.

Zhai, C., Cohen, W. W., and Lafferty, J. (2003). Beyond independent relevance: Methods and evaluation metrics for subtopic retrieval. In *Proceedings of ACM SIGIR'03*, pages 10–17.

Zhai, C. and Lafferty, J. (2001a). Model-based feedback in the KL-divergence retrieval model. In *Tenth International Conference on Information and Knowledge Management (CIKM 2001)*, pages 403–410.

Zhai, C. and Lafferty, J. (2001b). A study of smoothing methods for language models applied to ad hoc information retrieval. In *Proceedings of SIGIR'2001*, pages 334–342.

Zhai, C. and Lafferty, J. (2002). Two-stage language models for information retrieval. In *Proceedings of SIGIR'2002*, pages 49–56.

Zhai, C. and Lafferty, J. (2003). A risk minimization framework for information retrieval. In *Proceedings of the ACM SIGIR'03 Workshop on Mathematical/Formal Methods in Information Retrieval*.