View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Illinois Digital Environment for Access to Learning and Scholarship Repository

Optimal Information Retrieval with Complex Utility Functions

Tao Tao, ChengXiang Zhai
Department of Computer Science
University of Illinois at Urbana Champaign
Urbana, IL 61801

E-mail: {taotao,czhai}@cs.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 that may involve 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 [17, 16, 15], classic probabilistic models [13, 21, 20, 4], and the recently developed language modeling approaches [12, 5, 9, 2, 19, 6, 26, 8, 25, 27, 7] are among the major existing retrieval models, which have all been shown to perform well empirically. However, they all only address simple queries, which involve only a single simple utility function based on topic relevance with the assumption that all a user cares about is whether a document is *topically* relevant to the query.

Recently, the risk minimization framework [23, 28] attempts to go beyond "topic relevance only" retrieval by supporting 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 [24]. 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 almost always cares about multiple factors and the retrieval preferences are often quite complicated, involving different bounds on different factors. Consider, 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, the retrieval system has to face multiple factors(aspects) and cope with flexible preferences on each utility function. 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 approximation 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 a typical objective utility function. A user would 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 all the objective utility functions subject to the constraints set by 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 – anti-monotonicity and additivity, and propose an efficient algorithm for retrieval with complex utility functions that can work for any utility functions satisfying anti-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 Problem Formulation

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. Note DB is not necessarily a single database; the items can be in different 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 when the user is a child. 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 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. That is, $\forall \psi \in \Psi_u \; \psi : Q \times 2^{DB} \rightarrow R$, 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.

As discussed in the previous section, we can distinguish two types of utility functions – constraint utility functions and objective utility functions – based on how the user's retrieval preferences are formulated based on a utility function. A constraint utility function is one on which the user would impose a (lower) 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 [14, 23], 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 [23]. 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 π for presenting D to the user[23]. This implies that, in general, retrieval involves a two-step process with the first step choosing a subset of items D and the second presenting D appropriately. Suppose ranking is the only presentation strategy under consideration. This two-step process essentially says that we first select a subset of items and then rank them appropriately.

In general, we have *different* criteria for selecting the item subset and for ranking. This distinction is not important in traditional retrieval approaches, because we only deal 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.

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 [14, 23]. In this paper, we focus on the item selection

problem, which is important and non-trivial for multiple utility functions.

2.3 Item selection as 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 serve as the optimization objective function. Formally, such a multi-objective optimization problem is to find a subset X of DB to

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

, where $h, h_i \in \Psi_u$ $i = 1, ..., m$ and $h_i \neq h_j$ if $i \neq j$.

This is a standard optimization formulation with a single objective function as objective part. In our framework, we also allow users to put several complex utility functions in the objective part. However, due to the potential conflict of multiple objective functions, a final selection, among multiple optimal solutions, must be inevitably left to the user. Since it is generally unrealistic (even meaningless) to return all solutions a user in a retrieval task, we instead assume that the user can combine all the objective functions to form one single objective function for optimization. In this way, the optimization problem involves one single objective function but *multiple* constraints.

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. Third, within the SELECT clause, we introduce two additional clauses "MAXIMIZE' and "SUBJECTTO" to define retrieval preferences in terms of 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 an example query.

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 RANK clause indicates that the selected documents should be ranked by length.

Note that the proposed query language is 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)
```

2.5 Execution of the query

In order to answer a query in our query language, we need to first perform item selection and then rank the selected items. Since ranking is typically based on a single utility function, it is essentially similar to what most existing retrieval systems are doing. We thus focus on the item set selection part. The central question is how we can solve the selection optimization problem efficiently.

Without making further assumptions, it appears that the only possible way to find an exact solution to the selection optimization problem is through exhaustively enumerating all possible information item subsets based on the constraint utility functions and then selecting the best based on the objective function. Clearly, this naive algorithm is too time consuming and infeasible in reality. It is thus necessary to consider approximation algorithms or special cases of utility functions for which there exist most efficient algorithms.

3 Properties of utility functions

In this section, we study some special properties of utility functions that can be exploited to develop efficient algorithms for solving the retrieval optimization problem. We examine special cases of utility functions, and clearly define the properties that the utility functions must satisfy to enable us to design an efficient algorithm to take advantage of these properties. Without loss of generality, we assume all utility functions have non-negative values for both any item set.

3.1 Anti-monotonicity

A utility function $\psi \in \Psi$ is said to satisfy the anti-monotonicity property if for all $X, Y \subseteq DB$, $\psi(X \cup Y) \geq \psi(Y)$. Due to symmetry, $\psi(X \cup Y) \geq \psi(X)$ holds as well. The property is called anti-monotonicity because the function values are monotonically increasing when more items are added into the set. Thus, whenever a document set cannot satisfy a constraint, its super set would not satisfy the constraint either. Although it cannot change the NP-hard property of our problem, anti-monotonicity enables us to to prune the intermediate searching space substantially.

In our framework, multiple constraint utility functions are possible. All these constraints can be considered as providing some criteria for pruning the search space. If all constraint utility functions satisfy the anti-monotonicity property, we say this problem satisfies anti-monotonicity property. Fortunately, quite most of utility functions can be defined to have anti-monotonicity. For example, the relevance measure can be defined: to add one more items into a item set, the whole relevance will increase. Like relevance, communication cost, redundancy cost can all be defined by similar ways. This definition leads us to consider additivity property discussed in the next section. Certainly, there are some properties not satisfying anti-monotonicity property, like "average operation" etc. For these properties, we do not explore in this paper. A future study is highly needed.

3.2 Additivity property

In this section, we study the additivity property of utility functions, i.e., whether a utility function, operating over a set, can be decomposed into several "sub-functions" operating on individual information items. Specifically, we consider the case when a utility function $\psi \in \Psi$ can be written as a summation of different degree interaction utility functions. A function is called i^{th} **degree interaction utility functions** if it captures the interaction between all the i items in an item set of size i. ψ then can be rewritten as a summation of different i^{th} degree interaction utility functions:

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

, where $f_i(X) = \sum_{d1,...,d_i \in X} f_i(d_1, ..., d_i)$ accepts i items as its input values. In other words, an aspect is composed of the utility of different degrees of interactions. For example, suppose the user has to pay for each item, then the total cost for a set of items is a sum of the cost associated with different degrees of interactions. The simplest one would be the cost of each individual item; the user pays some amount of money for each information item. Suppose the user can have some discount if buying more than one item together, then the discount due to buying more than one item together would be an example of the utility associated with high-degree interaction. One can imagine that the more items a user buys, the more additional discount the user can obtain, which corresponds to having a separate i-degree interaction for each i.

Often we only have the first and second order interactions. The first order interaction makes the classic independence assumption: documents are independent of each other. Most utility functions fall into this category. A typical second order interaction is redundancy. If the problem involves second or higher order interactions, it can be proved NP-hard by reduction to the classic independent set problem. High order interactions are not naturally available. We do not address it in this paper.

It is interesting and encouraging that a utility function with additivity property and non-negative function values can be proved to satisfy anti-monotonicity property mentioned in Section 3.1. Since most utility functions indeed can be defined to have such an additive property, we design algorithms by the assist of the additive property.

4 An efficient complex utility retrieval algorithm

In this section, we study the complex utility retrieval with anti-monotonicity property. Anti-monotonicity property was first studied in data mining applications [11]. Although itself cannot solve the exponential combination problem(In other words, the problem is still NP-hard), the anti-monotonic property can help us to prune the search space quickly, which makes an algorithm applicable empirically. To deal with anti-monotonicity constraints, a well known data mining algorithm—A-priori algorithm was developed in [1], which is the basis of our algorithm. In this section, we will present our algorithm by first describing a-priori algorithm in some detail.

4.1 A-priori property and hash tree

We first describe the a-priori algorithm, which is first studied in the data mining community to solve frequent item sets problem: In a transaction database, people expect to find a set of items co-occurring in the transaction database frequently, *i.e.* the frequency of co-occurring times is larger than a pre-defined value called *support* [1]. A-priori property says that a set satisfies the *support* constraint only if all its subsets satisfy this *support* constraint. A-priori property is essentially an anti-monotonic property. A-priori algorithm, therefore, can be used to discover frequent item sets in transaction database. An example will make it more clear.

Transaction	Items
10	0 1 2 3
20	2 3
30	0 2 3 4
40	0 2 3
50	0 2 4

Table 1. Transaction database.

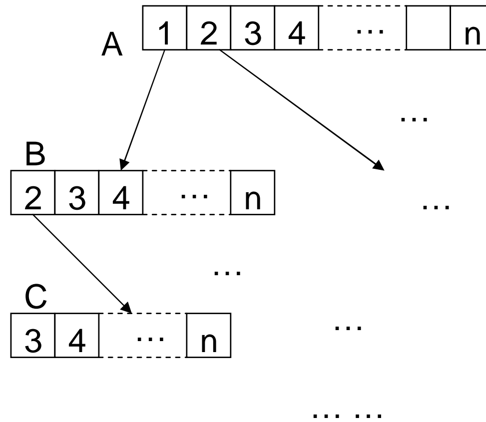

Figure 1. Hash tree.

In Table 1, there are 5 transactions; each of them has several items. If we expect support$= 3$, only tri-item set satisfying this support is $\{0, 2, 3\}$. It is easy to check all its subsets, $\{0\}$, $\{1\}$, $\{2\}$, $\{0, 2\}$, $\{0, 3\}$, $\{2, 3\}$, satisfy the support constraint.

Thus, it is reasonable to generate small sets first, and a large set is generated before checking all its subsets. If any subset is not qualified, no larger set cannot be qualified, and therefore all the supersets will not even be considered. In this way, the search space can be efficiently pruned. In its implementation, the hash tree, a data structure, is used to achieve the efficiency.

Figure 1 illustrates a hash tree, where A,B, and C are used to identify three hash tables. A hash tree is a tree, and each of its nodes is a hash table. Assuming there are n items, from 1 to n, each hash table has n possible keys, as shown in A at the root level. Each cell of a hash table has a link to a subtree(*e.g.* Item 1 in A linking to a hash table B). Similarly, B also has several subtrees. One of them is C linked under key 2. A path, following the effective links from an item at root level to an item in a certain node, represents an item set. For example, item 1 in hash table A, item 2 in hash table B, and item 3 in hash table C represents the item set $\{1, 2, 3\}$ because B is the subtree of item 1, and C is subtree of 2. Thus, to search whether a set is in this tree is just to follow the links. A set has no multiple appearances of the same items; therefore 1 is not in B, and 1, 2 not in C.

A hash tree can be constructed level by level gradually through a-priori algorithm. Due to the a-priori property, the candidate items in a children node can only be generated from its parents. For example, to generate items in C, all candidate items are from B. If, *e.g.*, 5 is not in B, it is will not in C.

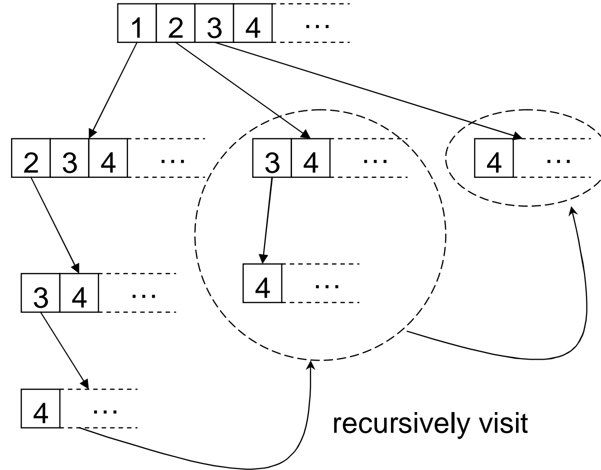

Figure 2. Visit as needs in hash tree.

4.2 Depth-first search and early stop

In this section, we study the characteristics of our problem, and develop our algorithm based on a-priori algorithm.

As mentioned in section 2, our optimization problem involves an objective function and several constraint functions. The support constraint in a-priori algorithm plays the similar role of our constraint functions. Therefore, an immediate idea for our problem is to apply a-priori algorithm directly. However, the two problems have two major differences: First, there is an objective function in addition to the constraints, in our problem. Second, instead of finding all sets satisfying the constraints, we are looking only for the best single set with the largest objective value. These two differences enable us to design the algorithm more efficiently.

In a-priori applications, there is no difference in the searching order of items. In our case, however, there is an objective function to optimize: We are looking for an item set satisfying the constraint bounds and also with the largest objective. Intuitively, a searching order starting from largest objective value should help. (Here, we are assuming the objective function has first order additive interaction property because the relevance is a major factor in information retrieval.) Therefore, we sort all the documents in descending order with respect to its objective values. From now on, we assume the item 1 has largest objective value. Item 2 has second largest objective value and so on.

We, then, design the algorithm using hash tree by depth-first search instead of breadth-first search as is used by a-priori algorithm. It is of course starting from document 1, and along the descending order of objective values. Depth-first search raises a problem: When encountering a set, all its subsets are not always available. For example, $\{1, 2, 3, 4\}$ has subset $\{2, 3, 4\}$, which is visited after $\{1, 2, 3, 4\}$ in the path of depth-first search. To cope with this problem, we use an "eager-visiting" technique: Whenever we need a set, which is not visited, we visit it immediately. This process is recursive. *e.g.* $\{1, 2, 3, 4\}$ needs $\{2, 3, 4\}$, which needs $\{3, 4\}$ recursively. This process is illustrated in Figure 2. We visit $\{1, 2, 3, 4\}$, but we do not check $\{2, 3, 4\}$ yet. We, then, recursively check $\{2, 3, 4\}$ as shown in the dashes circle, and then $\{3, 4\}$ In this way, we can achieve the identical effect as breadth-first search.

To change breadth-first into depth-first search is the main difference between our algorithm and a-priori algorithm. It looks simple, but has big impact. By doing this, we can take advantage of the characteristics of our problem, and apply early stopping criteria. We have the following two claims for early stopping. (Note: the documents are sorted in descending order of objective function values.)

Claim 1 *By depth first search, the first document set, satisfying all constraints, is* not *necessary the best one.*

It is easy to find a negative example to prove this claim. For example, item set $\{1, 2, 5, 6\}$ should be visited before $\{1, 3, 4, 5\}$ in depth-first search path. By assigning different objective values to the documents, either of those two sets could have larger objective value.

This claim tells us the first constraint-satisfied solution is not the best one for the objective value. Then, do we need to finish the whole depth-first search to find the best solution? The answer is "no"! Assuming we are looking for the document set with size k, we have the second claim.

Claim 2 *If either of the following conditions is satisfied, we can guarantee to stop depth-first search on current brunch or even the whole searching process without in danger of missing best solution. Assume we have already discovered several solutions satisfying all constraints: Let M is the largest objective value of these solutions. There are two conditions under which we can cut current searching brunch:*

- *Items left in current nodes is not enough to construct k items.*
- *Although there are enough items, the best possible items do not have large enough objective values to beat M.*

Both conditions are easy understood. Take the hash tree in Figure 1 for example: if we are visiting 3 in C now, and we are looking for 8 items for final solution. As we mentioned before, all candidate items in the children of 3 in C can only come from C. The first condition tells us we do not need to proceed if there are no enough items(5 items not including item 3 itself) in C. The second condition tells us we do not need to proceed even if there are 5 items left in C, but their objective values are too small to beat the best solution that we have already found.

Given the two claims above, we have our algorithm : a early-stop depth-first searching with "eager-visiting" technique.

It is worthwhile to mention that this algorithm is in fact an "any time" algorithm. Users can choose time-quality tradeoff. An approximate solution needs small amount of time while an optimal solution cost much more time. More time a user pay, better quality he can get. We will show it in the experiment section.

5 Experiments

In this section, we design the experiments to examine the effectiveness and efficiency of our algorithm. In general, our algorithm provides a way to do objective and constraints tradeoff while traditional algorithms can only maximize one single objective function(usually, relevance). By sacrificing a little on the objective function part, our method can largely reduce other constraint cost. This tradeoff gives users more flexibility in retrieval. With more emphasis on the objective part, other utility aspects will be compromised, while emphasizing the constraints part would cause a "sacrifice" on the objective function. In section 5.1, we describe an application of our model in distributed IR, and show that our algorithm can support this tradeoff in a flexible way. We then examine the efficiency of our algorithm in section 5.2. All our experiments are done on Linux platform with $2.4G$ cpu speed and $1G$ memory.

5.1 An application in distributed IR

Optimization on complex utility functions in information retrieval has many applications. A good example is the following distributed IR problem. In such a problem, the information (i.e., collection of documents) is distributed among many computers in distinct geographic locations. As in a normal retrieval problem, a user submits a query to describe the information need each. A centralized retrieval server is responsible for interacting with the user, taking the query and returning the results. Retrieval often involves three steps. In the first step (collection

selection), the server selects the most promising sites based on how likely they will contain relevant information to the query. In the second step, it sends the query to these selected sites, each of which has a local search engine that can return a ranked list of documents based on the query. Finally, the central server would merge the results returned from local retrieval engines and present one single ranked list of documents to the user. Existing research in distributed information retrieval has focused on two algorithmic challenges – collection/resource selection and merging/fusion [18, 10, 3]. However, 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. And 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 local retrieval situation.

Therefore, a user is in general interested in all the following three aspects of utilities: (1) topic relevance; (2) communication cost (or response time); and (3) redundancy. The goal is to retrieve as much relevant information as possible while at the same minimizing the communication cost and redundancy. It is thus a good example of retrieval with complex utility functions. In this section, we will study this example in our optimization framework, and test the proposed algorithms with simulated data sets.

We use Associated Press (AP) data set available through TREC [22] as our document database, which has $164,597$ news articles. We use TREC topics 101 - 150 as our experiment queries [22]. The objective function is "relevance" measure, and two constraints are: communication cost and redundancy. We simulate communication cost and redundancy in a distributed environment in the following way. First of all, we use language models [25, 27] to compute relevance value for each query. In general, user's main interest is still at "relevance" part, and therefore they are unlike to have interest in a very-low-ranked document w.r.t relevance measure no matter how efficiency the other two constraint are. Thus, we focus on top ranked documents only for each query without bothering the whole data set. We presume the retrieval is in distributed environment with 10 different sites. We randomly pick 20 documents for each site from top 400 relevant document returned from language models. Thus, 10 sites have 200 documents in total as the our candidates. Random picking could generate redundant documents on different sites. This does happen in reality, where different servers are likely hold same or similar documents. For each server, we generate a random communication coefficient α_i ($i = 1, ..., 10$). The communication cost for each document is defined by its document length multiplying the coefficient of its own server. The communication cost has first order interaction additive property. Redundancy of two documents is defined by the redundant words of two documents. Given document A and B, if term t appears both in A and B, redundancy on this term t is $\min\{c(t, A), c(t, B)\}$, where $c(t, X)$ is the number of t occurs in document X. Redundancy on two documents A and B is $\sum_t min\{c(t, A), c(t, B)\}$. This is not a very accurate redundancy measure, but since it does not affect our algorithm, we just take it. Redundancy of a set of documents is defined by the summation of redundancy of all pairs of documents within this set. This is the second-order interactive utility function. For easy constraint bound setting, we normalize all aspect(relevance, communication, and redundancy) ranging from 1 to 100.

We set final solution size to 7, *i.e.* we are aiming to select 7 documents from 200 documents. This parameter is indeed the searching depth of hash tree. Table 2 shows our results. The second row is optimal results without any constraints. Other three rows are results with some constraints. In the first column, there are two parameters for constraints bounding: the first number is communication bound, and the second one is redundancy bound. Other numbers are the experimental values. For example, 419.7 in second column, third row is the objective value of optimal solution under 300, 450 constraints. All these numbers are the average over all 50 queries.

From the table, we can easily observe the tradeoff: we restrict the communication cost and redundancy by losing some on relevance: all restricted solutions have smaller values on relevance part, but also have smaller values on communication cost and redundancy. These numbers show the tradeoff.

The table above is all on average. To show the performance on each individual queries, we select first 10 queries (query 101 to 110) with parameters: 300 and 450 to plot the performance in Figure 3. In this histogram, there are 6 types of bars: two relevance, two communication cost, and two redundancy. For each aspect, the first one is non-constraint solutions, and the second one is that with constraints. From this figure, we can clearly see the

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 2. Performance on AP data.

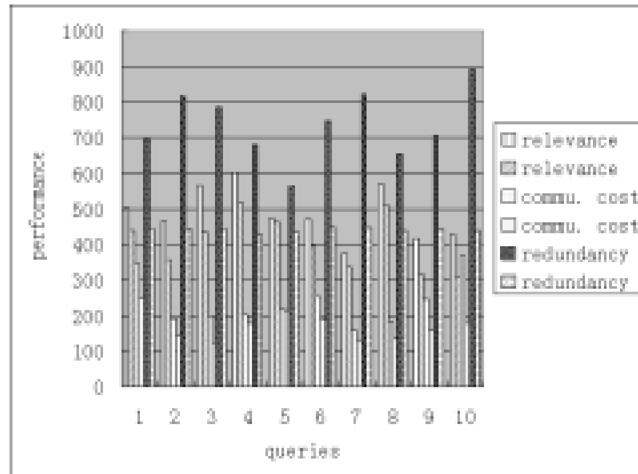

Figure 3. Performance on each individual query.

performance on each individual queries: they are indeed consistent with the average performance over all queries.

5.2 Efficiency

In this section, we evaluate the efficiency of our algorithm on some simulated data. Since the whole problem is NP-hard, it is impossible to have a polynomial time algorithm in general case.

We first evaluate the scalability on searching depth with the working set 200 documents, and two constraint functions. All these values are generated by uniform distribution(1-100) randomly. We use the percentage of the expected values as the constraint bounds setting. For example, if searching depth=5, 5 documents have expected value on a certain constraint is $50 \times 5 = 250$ because each one is from a uniform distribution over 1 to 100 . Then, 30% bounds will be $250 * 0.3 = 75$. Figure 4 shows the scalability on searching depth. The x-axis is the searching depth while the y-axis is running time in log scale. All values are average over 10 experiments with identical parameter setting. The plot shows the exponential property without surprise. However, we also we see our algorithm in fact outperforms much more than brute-force searching. For example, 200 documents and 10 level search. A pure enumeration will cost $200^{10} = 1.024 \times 10^{23}$! computation, but we can finish it on desktop within several seconds. We also implement **pure a-priori algorithm**, which cannot finish all these case in hours when searching depth is larger than 7 and number of documents larger than 400. In this figure, there are four curves corresponding four different constraint bounds setting. Obviously, a tight bound leads to more pruning in the hash tree, and it is usually more fast and efficient.

We then evaluate the scalability over the number of documents. In the experiment, we evaluate them from 200 to 2000 in Figure 5. It shows sub-exponential property. There are two curves in this figure to represent 25% and

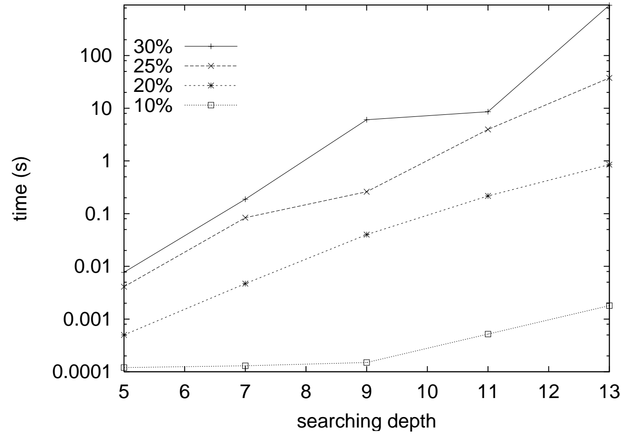

Figure 4. Scalability on the searching depth.

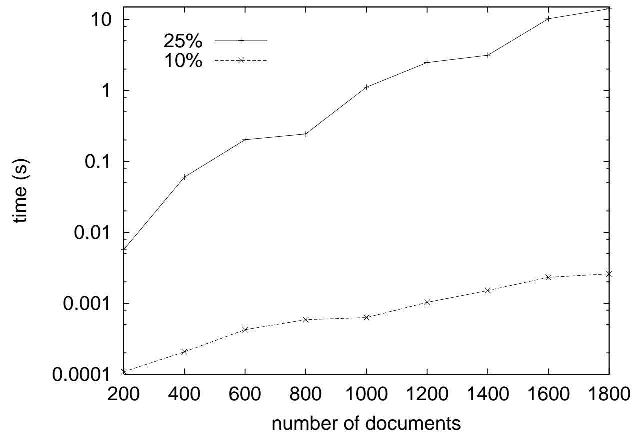

Figure 5. Scalability on the number of documents.

10% bounds setting. Both have searching depth 6. Again, all values are average over 10 experiments with identical parameter setting. Consistent with the figure above, a more strict constraint makes the curve more flat.

The scalability tells us 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 reality. We then examine another property of our algorithm – any-time stop.

We simulate 6 sets of data. Table 3 shows a comparison between the time when the program achieves optimal solutions(2nd row), and the time when the program really stops(3rd row). The test is done with 500 documents, searching depth 8, and 25% constraints bounds. The last row shows the percentage of second row over third row. It is clear to see the program gets optimal solution much earlier than it real stops. The percentage is usually is not bigger than 15%. Figure 6 is a concrete case to show how real performance is increasing over the spending time. We only plot x-axis up to the time of finding optimal solution. It shows the performance improves quickly in the beginning, and gets flat sooner although it can improve a little if spending more time.

From Table 3 and Figure 6, we see: First, an approximate algorithm is plausible. Early stop does not hurt the performance too much. Second, a more efficient optimal algorithm, to detect the optimal point earlier, is possible.

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 3. Any time stop.

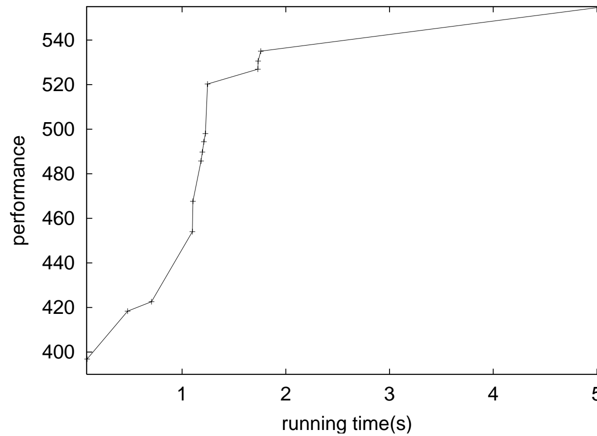

Figure 6. Performance improving over running time.

This leave room for future work.

6 Conclusions and future work

In this paper, we first propose a novel framework to address a new retrieval problem: information retrieval with multiply utility functions. The fundamental difference between our framework and previous ones is that we allow users to consider *multiple* aspects of utilities. This allows much more flexibility in retrieval, and reflects a user's preferences more accurately. We then formulate the problem as a constraint optimization problem, and design a query language to execute these complex retrieval tasks. We study two special properties of the utility functions – anti-monotonicity and additivity, and propose an efficient algorithm for retrieval with complex utility functions that can work for any utility functions satisfying anti-monotonicity. Although retrieval with complex utilities is general NP-hard, the proposed algorithm is relatively efficient for complex utility retrieval over a working set of documents selected based the primary objective utility function.

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 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.

There are several issues for further exploring and extending the work presented here: (1) Algorithm efficiency. 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. In the future, more efficient algorithms have to be explored. Presumably, major effort should be on good approximate algorithms, instead of optimal ones. (2) Utility functions without anti-monotonicity property. In this paper, we address anti-

monotonicity. As we mentioned before, some utility functions, for example average operation, do not satisfy this property. How to cope with such utility functions is still an open question.

References

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In J. B. Bocca, M. Jarke, and C. Zaniolo, editors, *Proc. 20th Int. Conf. Very Large Data Bases, VLDB*, pages 487–499. Morgan Kaufmann, 12–15 1994.

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

[3] J. Callan, F. Crestani, H. Nottelmann, P. Pala, and X. Shou. 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*.

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

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

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

[7] J. Lafferty and C. Zhai. Probabilistic relevance models based on document and query generation. 2003.

[8] V. Lavrenko and B. Croft. Relevance-based language models. In *Proceedings of SIGIR'2001*, Sept 2001.

[9] D. H. Miller, T. Leek, and R. Schwartz. 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, 1999.

[10] H. Nottelmann and N. Fuhr. 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*.

[11] J. Pei, J. Han, and L. V. S. Lakshmanan. Mining frequent item sets with convertible constraints. *ICDE*, pages 433–442, 2001.

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

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

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

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

[16] G. Salton and M. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, 1983.

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

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

[19] F. Song and B. Croft. 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, 1999.

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

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

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

[23] C. Zhai. *Risk Minimization and Language Modeling in Text Retrieval*. PhD thesis, Carnegie Mellon University, 2002.

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

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

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

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

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