

Tuning Before Feedback: Combining Ranking Discovery and Blind Feedback for Robust Retrieval

Weiguo Fan
Ming Luo
Digital Library Research Lab
Virginia Tech
{wfan, lming}@vt.edu

Li Wang
School of Information
University of Michigan, Ann Arbor
wang@umich.edu

Wensi Xi
Edward A. Fox
Digital Library Research Lab
Virginia Tech
{xwensi, fox}@vt.edu

ABSTRACT

Both ranking functions and user queries are very important factors affecting a search engine's performance. Prior research has looked at how to improve ad-hoc retrieval performance for existing queries while tuning the ranking function, or modify and expand user queries using a fixed ranking scheme using blind feedback. However, almost no research has looked at how to combine ranking function tuning and blind feedback together to improve ad-hoc retrieval performance. In this paper, we look at the performance improvement for ad-hoc retrieval from a more integrated point of view by combining the merits of both techniques. In particular, we argue that the ranking function should be tuned first, using user-provided queries, before applying the blind feedback technique. The intuition is that highly-tuned ranking offers more high quality documents at the top of the hit list, thus offers a stronger baseline for blind feedback. We verify this integrated model in a large scale heterogeneous collection and the experimental results show that combining ranking function tuning and blind feedback can improve search performance by almost 30% over the baseline Okapi system.

Categories and Subject Descriptors

H.3.3 [INFORMATION STORAGE AND RETRIEVAL]: Information Search and Retrieval – *Query formulation, Relevance feedback, Retrieval models, Search process, Selection process*

General Terms

Algorithms, Measurement, Performance, Experimentation

Keywords

Intelligent Information Retrieval, Blind feedback, Query Expansion, Search Engine, Ranking Function, Genetic Programming, Information Retrieval

1. INTRODUCTION

Text resources in digital format are quickly increasing with the rapid development of the IT industry, leading to ever growing

repositories of information to support not only corporate interests but also the public at large. However, many challenges face those who aim to make full use of that information. Information retrieval (IR) systems or search engines are designed to address such needs. Given a user query representing a user's information need, an IR system will search through its index of documents and return an ordered list of documents according to their match with the user query.

For years, IR researchers have conducted extensive experiments to improve the search performance of an IR system or a search engine. It is known in the IR literature that there are many factors that could affect an IR system's performance: user query specificity and accuracy, document indexing, ranking function, user's familiarity with an IR system, etc. Most of the recent work on performance tuning center on query improvement through blind feedback [4][17][22], and ranking function tuning using machine / statistical learning techniques [7] [8] [9] [10] [11] [13] [23] [29].

Blind feedback is a widely used technique to improve retrieval effectiveness. In blind feedback, a small number of documents are first retrieved according to the user's query and these documents are assumed to be relevant. Words in those documents as well as words in the original query are used for relevance feedback (more details later). The expanded query will replace the original short user query and retrieve a new set of documents. Generally, blind feedback can improve search performance by 5% to 10% [4][17][22][25].

Another line of performance improvement is to tune or optimize the ranking functions used in the document query matching process. Various machine learning or statistical learning techniques, ranking fusion techniques have been used in the past for this purpose [2][7][8][9][10][11][13][21][23][29].

Since both blind feedback and ranking function tuning are effective in performance improvement, it is natural to consider the effects if we combine them together. We hypothesize that the ranking function tuning stage should be followed by the blind feedback stage. The intuition is that *a well-tuned high-performing ranking function should provide more relevant documents at the top of the search hit list, and thus enable more effective blind feedback.*

Thus our research goal in this paper is to perform extensive large scale experiments to test our hypothesis, along with its implications, as stated in the following research questions:

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SIGIR '04, July 25–29, 2004, Sheffield, UK.

Copyright 2004 ACM 1-58113-000-0/00/0004...\$5.00.

1. Does blind feedback work well on fine-tuned ranking functions as compared to traditional ranking functions such as Okapi BM25?
2. Does the type of queries (very short vs. very long queries) have any impact on the combination approach?
3. Can the ranking function discovered in combination with blind feedback extrapolate well for new unseen queries?

In order to answer these questions, we choose ARRANGER – a Genetic Programming-based discovery engine [9][10] to perform the ranking function tuning. This engine has been shown to be very effective in discovering new ranking functions that perform better than many well-known ranking schemes such as Okapi BM 25, Pivoted TFIDF, etc. We then apply various blind feedback techniques to the newly discovered ranking functions and test the effectiveness of our combination approach.

Our paper is organized as follows. In Section 2, we review related work on ranking function optimization, especially the ARRANGER ranking discovery engine used in our experiments. We also review the related work on blind feedback techniques in this section. In Section 3, we outline our two-stage combination model and give the intuition behind it. In Section 4, we describe the settings of our experiments, including data collection characteristics. The experimental results on both old queries (the regression experiment) and new queries (the extrapolation experiment) are presented and discussed in Section 5. Section 6 concludes this paper and points out future work.

2. BACKGROUND

2.1 Ranking Function Optimization

There have been several efforts on ranking function optimization in IR literature. Some of the earliest work could be traced to N. Fuhr et al. [11], [12] using probabilistic models as machine learning approaches. The concept of relevance description used in [11], [12] is very similar to the common weighting evidence such as *tf*, *idf*, etc. In [11] and [12], the ranking function (called retrieval function in [11], [12]) is either a polynomial regression function [11], or logistic regression/loglinear function [12]. Similar ideas using logistic regression for ranking function design and optimization have also been explored in [13].

Mixture of experts approach, or ranking fusion, forms another front of research on ranking function optimization, in which a set of ranking functions are combined either numerically through linear combination [2][29][23], or simple majority vote [21].

2.2 Ranking Function Discovery Using ARRANGER

ARRANGER, Automatic Rendering of RANKing functions by GEnetic programming, is a discovery engine developed by Fan et al. [9][10]. The core of this engine is a machine learning technique called Genetic Programming (GP). Genetic Programming, an extension of Genetic Algorithms (GA), is an artificial intelligence technique inspired by Darwin’s theory of evolution. “Computer programs that evolve in ways that resemble natural selection can solve complex problems even their creators do not fully understand” [16]. Genetic Programming has been widely used and proved to be effective in solving optimization problems, such as financial forecasting, engineering design, data

mining, and operations management [1][19]. GP makes it possible to solve complex problems for which conventional methods can not find an answer easily. We next review basic knowledge about GP in order to explain how ARRANGER works.

In GP, a large number of individuals, called a population, are maintained at each generation. An individual represents a tentative solution for the target problem. All these solutions form a space, say, Σ . Individuals can be stored using complex data structures, such as a tree, a linked list, or a stack. A tree is the most popular form to store and represent individuals. Figure 1 shows an individual representing a ranking function. A fitness function ($f(\cdot): \Sigma \rightarrow \mathbb{R}$) is also needed in Genetic Programming. A fitness function takes the solution space, Σ , as its domain and returns a real number. Hence tentative solutions, represented by individuals, can be evaluated and ordered according to their return values. The return value of a fitness function must appropriately measure how well an individual, which represents a solution, can solve the target problem.

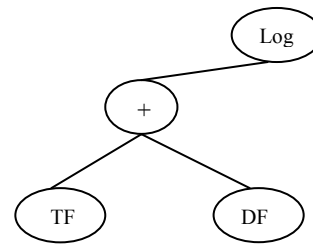


Figure 1. A simple ranking function

Genetic Programming searches for an “optimal” solution by evolving the population generation after generation. Individuals in a new generation are produced based on those in the previous one. Three genetic operators are usually used to produce the new generation. They are Reproduction, Crossover, and Mutation. The reproduction operator directly copies or, in a more appropriate term, clones some individuals into the next generation. The probability for an individual to be selected for Reproduction should be proportional to its fitness. Therefore the better a solution solves the problem, the higher probability it has to enter the next generation. While Reproduction keeps the best individuals in the population, Crossover and Mutation introduce transformation and so provide variations to enter into the new generation. The crossover operator randomly picks two groups of individuals, selects the best (according to the fitness) individual in each of the two groups as parent, exchanges a randomly selected gene fragment of each parent and produces two “children”. Thus, a “child” may obtain the best fragments of its excellent parents and so may surpass them, providing a better solution to the problem. Since parents are selected from a “competition”, good individuals are more likely to be used to generate offspring. The mutation operator randomly changes a gene code, which could be a function or a parameter in our ranking function discovery task, of an individual. Using these genetic operators, a new generation is produced. The new generation keeps individuals with the best fitness in the last generation and takes in “fresher air”, providing creative solutions to the target problem. Better solutions are obtained either by inheriting and reorganizing old ones or by lucky mutation, simulating Darwinian Evolution. As we can see,

Genetic Programming takes a so-called *stochastic search* approach, intelligently, extensively, and “randomly” searching for the optimal point in the entire solution space. It is less likely to be trapped in local optima, which is the major problem of many other search algorithms. It provides sound solutions to many difficult problems, for which people have not found a theoretical or practical breakthrough.

2.2.1 Motivation for using genetic programming in ranking function discovery

The choice of using GP for ranking function discovery was motivated by three factors: a) the large size of the search space; b) the characteristics of the objective function; and c) modeling and representation advantage.

- First, a ranking function can essentially be represented in a tree structure (Figure 1). For our trees we have used ten terminals (Table 1), four functions, trees of depth no more than ten, and real constants which can vary between 0 and 1. Langdon et al. [20] shows that for these parameters the search space for a tree is very large and the problem is essentially a needle-in-a-haystack problem. Hence other search mechanisms like random search and exhaustive search would take inordinate time [20]. GP has been shown to perform well under such conditions.
- Second, most performance measures in IR are discrete in nature (for example, relevance). GP does not require that objective functions be continuous in nature as long as it can distinguish good solutions from bad ones [19].
- Third, a tree data structure is used in our GP implementation. We found that GP is extremely suitable for the ranking function discovery task as it can automatically combine the various weighting features to approximate a user’s ranking preference. An example of representing one ranking function using a tree structure is shown in Figure 1. The tree-based representation allows for ease of parsing and implementation.

2.2.2 Outline of our GP-based ranking function discovery system – ARRANGER

In this section, we give a brief introduction to the ARRANGER engine. Please refer to [9][10] for details and validation.

A ranking function consists of three parts: variables, constants, and operations (which connect the first two parts), as shown in Figure 1. Hence we need to identify all the potential variables that are used in the ranking function by ARRANGER. Some examples for these variables are the term weighting factors shown in the first column of Table 1. The second column of Table 1 gives the meaning of these variables.

There are two different types of variables, scalar and vector. Some of these predefined variables are summaries calculated for the whole collection or a specific document, such as tf_{max} , N , tf_Avg_Col , etc. These variables belong to the category of scalar variable. Constants are defined to be scalar only. The remaining variables have a vector nature, such as tf_doc and tf_query . We defined that when such variables appear in a ranking function, they represent vectors, instead of single numbers. For example, if a query has n words in it, tf_doc could be represented by $(x_1,$

$x_2, \dots, x_n)$, where x_i ($i = 1, 2, \dots, n$) is the term frequency (tf) of the query’s i th word in the document.

Table 1. Definitions for variables used for ranking discovery by ARRANGER

tf	Query term frequency in the document (vector)
tf_query	Query term frequency in the query (vector)
tf_max	The maximum term frequency in a document (scalar)
Length	Document length in the number of words (scalar)
Length_avg	Average document length in the number of words (scalar)
N	Number of documents in the collection (scalar)
tf_avg	Average term frequency in the current document (scalar)
tf_avg_Col	Average term frequency for all the documents in the collection (scalar)
df_max_Col	Maximum document frequency for a word in the collection (scalar)
df	Document frequency for the query words (vector)

Based on pre-selected variables and constants, we define two types of functions (operations), single-parameter functions (denoted by $\sigma(\cdot)$) and two-parameter functions (denoted by \circ). Single-parameter functions include $\log(\cdot)$ and $\sqrt{\cdot}$. Two-parameter functions include $+$, $-$, $*$, $/$. Some functions, such as $\log(\cdot)$, $\sqrt{\cdot}$ and $/$, need to be protected, since the domain of these functions is not the whole real number space. As a variable could be a scalar or a vector, the functions must take that into consideration. For one-parameter functions, we define $\sigma(x) = y$ and $\sigma(x_1, x_2, \dots, x_n) = (\sigma(x_1), \sigma(x_2), \dots, \sigma(x_n))$, where x , y and x_i represent scalar variables and (x_1, x_2, \dots, x_n) is used to represent vectors. For two-parameter functions, we define $x \circ y = z$, $x \circ (x_1, x_2, \dots, x_n) = (x \circ x_1, x \circ x_2, \dots, x \circ x_n)$ and $(x_1, x_2, \dots, x_n) \circ (y_1, y_2, \dots, y_n) = (x_1 \circ y_1, x_2 \circ y_2, \dots, x_n \circ y_n)$, where x , y , z , x_i and y_i represent scalars and (x_1, x_2, \dots, x_n) represents a vector. Following our definitions for variables and functions, when a vector variable appears in the ranking function, the final result also is a vector, but a scalar usually is needed to measure the similarity between a document and a query. Hence, we further define that the return value of a ranking function is the summation of all the elements when a vector is finally returned by that function. By following these rules, the ARRANGER can work on discovering ranking functions. Note that when we plug in the newly-discovered functions into our search engine, the same rules must be followed.

ARRANGER works as follows: First, the best ranking functions learned from the training set are stored and the rest are discarded. Then those functions are tested on the validation set. According to their performance, the functions which do not have consistent performance on both data sets are screened out. Only the most robust and consistent functions are selected. They form the ranking function candidate pool. Since an appropriate stopping rule is hard to find for the Genetic Programming approach, over-training is inevitable unless protecting rules are set. By running the ranking functions on an independent holdout data set (the so-called residual collection method), however, over-trained functions are filtered out.

2.3 Blind Feedback

Blind feedback, also called pseudo-relevance feedback or automatic query expansion, is a technique that automatically adds more terms to a user's query to enhance the performance of search engines. It is a widely-used and effective technique, especially for very short queries. In blind feedback, a small number of documents are first retrieved according to a user's query and these documents are assumed to be relevant. Words in those documents as well as words in the original query are sorted according to a weighting function. An expanded query is generated by selecting some words from this list.

Two popular forms of blind feedback techniques are shown in Equations (1) and (2). As we can see from these formulas, they are essentially modified Rocchio relevance feedback formula [26] and Ide Dec-Hi formula [17], respectively.

$$Q_{i+1} = \alpha Q_i + \beta \sum_{j=1}^{|R|} \frac{R_j}{|R_j|} \quad (1)$$

$$Q_{i+1} = Q_i + \sum_{j=1}^{|R|} R_j \quad (2)$$

R_j is the set of retrieved documents. Q_i is the old query. Q_{i+1} is the new query.

Besides the above-mentioned two most popular approaches based on the Vector Space model, there are also other blind feedback techniques designed based on probability or information theory. For example, a information theory based measure called relative entropy, or Kullback-Leibler Divergence (KLD), has been used to select terms for blind feedback in [3]. Other traditional probabilistic query expansion formulas such as RSV [24], CHI, and DRC [6] also can be modified (replacing the non-relevant documents statistics with the collection level statistics since we do not know which documents are non-relevant) for blind feedback. [18] uses noun phrases instead of simple single terms for automatic blind feedback in web document retrieval.

As shown later in the experiments section, we also experiment with various blind feedback techniques and find Rocchio or Ide Dec-Hi to be the most effective ones in our settings.

3. COMBINING RANKING TUNING AND FEEDBACK – AN INTEGRATED MODEL

In this paper, we propose a two-stage integrated model for ad hoc retrieval performance improvement. As shown in Figure 2, this model involves the ranking function tuning stage followed by the blind feedback stage. The ranking tuning stage uses historical user queries with relevance information to tune the ranking function. The end product of this process is a highly optimized ranking function, designed to work well for these training queries, that also may work well for new queries (our third research question).

The intuition behind this model is that *tuning the ranking function provides a stronger baseline for blind feedback*. In other words, a well-tuned high-performing ranking function should provide more

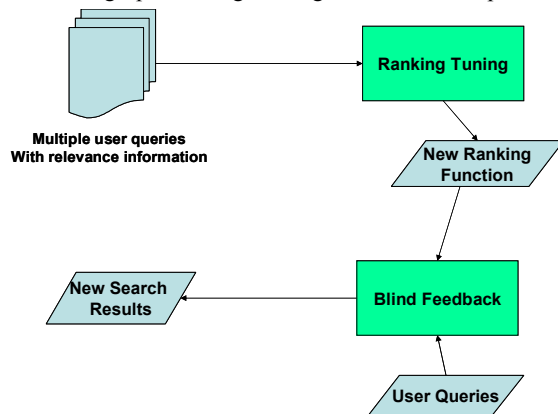


Figure 2: The integrated model for performance improvement

relevant documents at the top of the search hit list, thus a stronger baseline than a low-performing ranking function. When combined with the blind feedback, the overall performance gains by the new ranking function should be higher than the low-performing ranking function.

The model, shown in Figure 2, combines the merits of both blind feedback and ranking function tuning. Recent empirical research on ranking function tuning shows that it can generally improve the ad hoc retrieval performance by 10% to 30% [7][8][9]. Similarly, blind feedback can generally improve performance by 5% to 15% [4][5][25]. Since blind feedback is supposed to work for all kinds of ranking functions [4][5][25] based on the feedback framework as shown in Equations (1) and (2), it is safe to infer that these two techniques are essentially independent of each other. This means that the performance gain of applying both techniques is the sum of applying them individually. We test this conjecture in the experiment.

4. EXPERIMENT SETTING

4.1 Collections

We use the document collections from the ad hoc track of TREC 6, 7, and 8. There are four collections in total. Some statistics about these collections are shown in Table 2.

Table 2. Collection statistics

	Size	Number of Documents	Average Doc Length
FT '91-94	564 MB	210,158	433.21 words
FR '94	395 MB	55,630	704.25 words
FBIS	470 MB	130,471	555.96 words
LA '89-90	475 MB	131,896	542.50 words

This set of collections is quite heterogeneous in nature. While Financial Times (FT) and Los Angeles (LA) Times are better tagged with basically one newspaper article per document, many documents in FBIS are very long, with several pieces of segments that are not related to each other. For example, FBIS4-46649 has 144398 words (after parsing) and it is actually a book chapter about “Tactics: Company, Battalion” translated from Russian. Similarly, the length of Federal Register (FR) documents is unevenly distributed. Some documents just have a patent number and some have an extremely long statement of Federal regulation.

The wide variance in the set of collections affords an ideal setting for our experiments since we want to test the robustness of our model in a very heterogeneous environment.

4.2 Data Processing

All of our experiments were run on a two-2.3GHz processor Dell Server running the Linux operating system. We did not make use of the document structure. In the parsing process, we simply removed the non-informative content in the collection and kept only the texts in the TEXT field. These texts were indexed into both forward index and inverted index formats for our experimental purposes after removing stop words and stemming. No phrases were used in our experiments.

For query processing, we indexed three different versions of the topic descriptions. The first version is description queries, which are generated based on the Description field only. The second, short queries, are based on the Title and Description fields. The third, long queries, are extracted based on all three fields, i.e., the topic Title, Description, and Narrative.

5. EXPERIMENTS

We conducted two separate experiments to validate our proposed integrated model. The first experiment is a regression experiment in which we tune the search engine’s ranking for 150 old queries. After the tuning is done, we apply the blind feedback on the new ranking scheme on a separate residual collection (test collection) and compare performance results. This experiment is to test whether the proposed two-stage model works for the same queries in different collections. The second experiment is an extrapolation experiment. We need to find out whether the newly discovered ranking function combined with blind feedback can work well for other unseen new queries – testing the robustness of our integrated model. In this experiment, we use the 50 queries from the latest robust track of TREC 11.

5.1 Experiment 1: Performance Tuning on Old Queries

For the first experiment, we give details of tuning the search engine’s performance using our proposed two-stage model. This tuning is based on 150 old queries available from the ad-hoc retrieval track of TREC 6, 7, and 8.

5.1.1 Ranking Tuning Using ARRANGER

We used ARRANGER to discover “optimal” functions on the Robust Track collection. These ranking functions are trained on various versions of the user queries. We tested the automatically learned functions on three types of queries: description query, short query, and long query as described in Section 4, using a residual collection method. Table 3 shows the results on the residual collection. From this table, we can see that significant improvement is achieved by replacing the Okapi BM25 function with our newly-discovered functions. This proves the effectiveness of ARRANGER ranking function tuning.

Table 3. Performance comparison between Okapi BM25 and ranking functions discovered by ARRANGER on 150 queries

	Desc	Short	Long
Okapi BM25 (baseline)	0.1880	0.2194	0.2375
RF 1	0.2173 (+15.6%)	0.2394 (+9.1%)	0.2620 (+10.3%)
RF 2	0.2079 (+10.6%)	0.2317 (+5.6%)	0.2607 (+9.8%)
RF 3	0.2047 (+ 8.9%)	0.2282 (+4.0%)	0.2590 (+9.1%)
RF 4	0.2036 (+8.3%)	0.2245 (+2.3%)	0.2602 (+9.6%)

Among the four new ranking functions discovered, RF 1 performs the best. It improves upon Okapi BM 25, with description queries, by more than 15%.

5.1.2 Blind Feedback

We apply various blind feedback techniques, based on new functions discovered by our ARRANGER. They are Rocchio [26], Ide Dec-Hi [17], CHI [6], KLD [3], RSV [24], and a variation of KLD [3], which we derived based on probability theory. Those techniques are applied on both description queries and long queries. They yield performance improvement on both types of queries. As we expected, they improve more on description queries than long queries. For each approach, there are several parameters to be adjusted, for example, the number of documents assumed relevant, the number of terms for the expanded query, and parameters in the weighting function. Since it would take an enormous amount of time to try out all combinations, a factorial design was used to look for the “best” parameter settings. We found that using the top 6 documents, with 30 feedback terms, gives the best overall performance.

We then applied sensitivity analysis around these optimal numbers.

Figure 3 shows the number of documents effect on MAP (mean average precision). We used the Rocchio feedback method with 30 feedback terms. As can be seen from Figure 3, using the top 2 documents gives the best performance for GP function 1. However, using the top 2 documents is not ideal for the other functions. In general, using the top 6 documents gives the best performance for all GP ranking functions. Therefore we chose to use the top 6 documents for feedback.

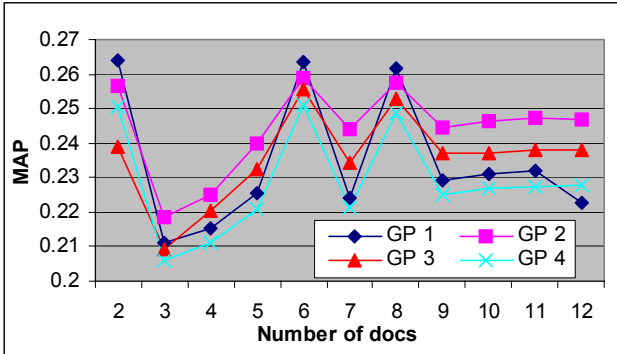


Figure 3. Number of documents effect on GP ranking function feedback. Query terms are fixed at 6.

In terms of the sensitivity analysis on the effect of the number of feedback terms using different weighting schemes, namely Rocchio, KLD and CHI, Figure 4 shows MAP at various levels of terms. Number of documents is fixed at 6 and queries are description queries only. We found that Rocchio feedback performed the best for our discovered ranking functions. The number of terms has very little impact on MAP. The ranking function used in Figure 4 is RF 1. Overall, using 30 terms for query expansion seems optimal.

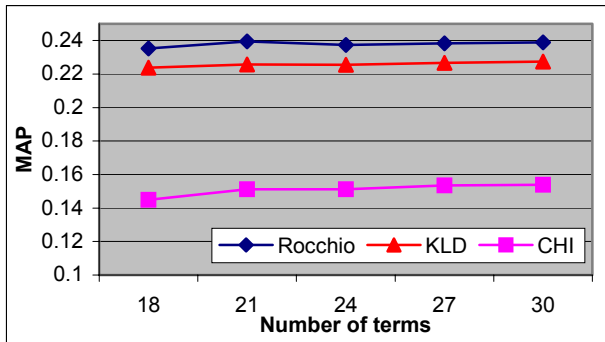


Figure 4. Feedback methods on various feedback terms

Table 4 gives the performance comparisons between RF 1 and Okapi using Rocchio blind feedback method as shown in Equation (1) on the 150 old queries. All results are reported in terms of MAP. Only description queries and short queries results are reported here in Table 4. The performance improvement results on long queries using blind feedback are not reported because of lack of statistical significance.

As can be seen, blind feedback technique works well for both description queries and short queries, with description queries gaining the most.

Compared with the Okapi baseline without blind feedback, blind feedback on Okapi ranking improves by 10.43% for description queries, and 8.71% for short queries. For RF 1 discovered by the ARRANGER engine, blind feedback further improves the performance by 11.46% for description queries and 11.15% for short queries. Blind feedback gains more in RF 1 than Okapi. This clearly answers our first research question that highly-tuned ranking function helps blind feedback since they can provide more relevant documents at the top of the rank list. In other words, ranking function tuning does provide a stronger baseline and helps for blind feedback.

Overall, when combining the two stages of ranking function tuning and blind feedback, we see that we get more than 28% improvement over the Okapi baseline system with no feedback for description queries. For short queries, the improvement is more than 21%. These improvements clearly prove that the combination of ranking function tuning and blind feedback is very beneficial. It also verifies our hypothesis that the effect of ranking function performance tuning is independent of the effect of blind feedback. In other words, these performance gains are additive. The higher performance gain in description queries than short queries does indicate the query effect raised in our second research question.

Table 4. The effect of blind feedback (BF)

Run No.	Desc	Short
Okapi without BF (Baseline)	0.1880	0.2194
Okapi with BF	0.2076 (+10.43%)	0.2385 (+8.71%)
RF 1 without BF	0.2173 (+15.59%)	0.2394 (+9.12%)
RF 1 with BF	0.2422 (+28.83%)	0.2661 (+21.29%)

5.2 Experiment 2: Extrapolation Test

The purpose of this experiment is to see whether the two-stage model using the newly discovered ranking function from the previous experiment can extrapolate well for other user queries. To test this hypothesis, we test the new RF 1 again combined with blind feedback on the 100 testing queries from TREC 11 Robust track. The 100 queries contain 50 old very hard queries from the 150 queries from TREC 6, 7, 8, and 50 new queries. The results for the 100 queries are shown in Table 5. Table 6 lists the results for the 50 new queries only.

If we compare the results in Tables 5 and 6 with those in Experiment 1, we can see that the improvement of RF 1 over Okapi on description queries drops from more than 15% in Experiment 1 to more than 6% in Experiment 2 on the 50 new queries. This is largely due to the variance of user queries. In fact, the 50 new users queries are relatively easier queries compared to previous ones. This makes the ranking function we tuned based on old 150 queries not extrapolate very well on the new 50 queries. Nevertheless, the new ranking function RF 1 still has

performance edge over the baseline Okapi BM 25 by more than 6%.

Table 5. Performance results on 100 test queries (50 old hard queries + 50 new test queries)

Run No.	Desc	Long
Okapi without BF (Baseline)	0.2107	0.2298
Okapi with BF	0.2322 (+10.20%)	0.2415 (+5.09%)
RF 1 without BF	0.2309 (+9.59%)	0.2522 (+9.75%)
RF 1with BF	0.2545 (+20.79%)	0.2731 (+18.84%)

Table 6. Performance results on 50 new test queries

Run No.	Desc	Long
Okapi without BF (Baseline)	0.3289	0.3454
Okapi with BF	0.3510 (+6.72%)	0.3636 (+5.27%)
RF 1 without BF	0.3489 (+6.08%)	0.3670 (+6.25%)
RF 1with BF	0.3801 (+15.54%)	0.3907 (+13.12%)

The results in Table 5 look better than those in Table 6. This is not a surprise as the 100 queries contain 50 old queries that are part of the training queries in our first experiment. The differences imply that the ranking function discovered in Experiment 1 works better on the old 50 hard queries than the 50 new queries. The differences also indicate that tuning a ranking function that can work well for all queries is not a trivial task. The ranking function discovered is typically very context dependent, which is in consistence with previous ranking function optimization findings [9][27][30].

Another observation that can be made from the results in Tables 5 and 6 is that blind feedback works consistently well. It improves Okapi baseline by more than 6% for description queries and more than 5% for long queries. On the other hand, blind feedback improves on RF 1 by almost 9% and 6.5% for description queries and long queries, respectively. The higher performance gain of blind feedback in RF1 over Okapi is in consistent with the results in Experiment 1 and also proves our initial hypothesis that ranking function tuning helps blind feedback.

The overall performance gain by doing the two-stage model is clearly beneficial. The performance gain of the two stages is almost equivalent to the sum of performance gain of doing each of the two stages individually. This point is consistent with our hypothesis raised earlier and the results in Experiment 1. This answers our third research question that our proposed model still can extrapolate well for new unseen user queries.

6. CONCLUSION

In this paper, we look at the performance improvement for ad-hoc retrieval from a more integrated point of view by combining the merits of ranking function tuning and blind feedback. In particular, we argue that a ranking function should be tuned first using user provided queries before applying the blind feedback technique. Our experimental results confirm our initial hypothesis that highly-tuned ranking offers more high quality documents at the top of the hit list, and thus offers a stronger baseline for blind feedback. We also find that this integrated model works well for very short queries. The performance gain using both techniques is almost the sum of the gains by applying the individual techniques alone. Our results on the 150 old queries in Table 4 show that combining ranking function tuning and blind feedback can improve the search performance by almost 30% over the baseline Okapi system. This is very encouraging considering the fact that most ad hoc search queries are very short and many of these queries are repetitive. Our model also extrapolates well for new users queries.

Note that in the current implementation, we use ARRANGER (a GP-based discovery engine) to tune the ranking function. It is possible that this stage could be replaced by other ranking function tuning techniques, such as ranking fusion or logistic regression. The effect of such implementations will be left for future research.

We also plan to test this integrated model on more document collections such as web collections and news collections, to see whether this integrated model will work well for those contexts.

7. ACKNOWLEDGMENT

This material is based on work supported by the National Science Foundation under Grant Number IIS-0325579, DUE-0136690 and DUE-0333531.

Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

We are also grateful to Ye Zhou, Yui Yang for their programming support.

8. REFERENCES

- [1] Banzhaf, W., Nordin, P., Keller, R. E., and Francone, F. D., *Genetic Programming: An Introduction — on The Automatic Evolution of Computer Programs and Its Applications*. San Francisco, CA, Morgan Kaufmann Publishers, 2000.
- [2] Bartell, B. T., Cottrell, G. W., and Belew, R. K., Automatic combination of multiple ranked retrieval systems, in the *Proceedings of Seventeenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 173–181, 1994.
- [3] Carpineto, C., Mori, R. D., Romano, G., and Bigi, B., An information-theoretic approach to automatic query Expansion, *ACM Trans. on Information Systems*, 19(1), 1-27, 2001.

- [4] Carpineto, C., Romano, G., Improving retrieval feedback with multiple term-ranking function combination, *ACM Trans. on Information Systems*, 20(3), 259-290, 2002.
- [5] Chung, Y. M. and Lee, J. Y., Optimization of some factors affecting the performance of query expansion, *Information Processing and Management*, in press, 2004.
- [6] Fan, W., Gordon, M. D. and Pathak, P., Effective profiling of consumer information retrieval needs: a unified framework and empirical comparison, *Decision Support Systems*, in press, 2004.
- [7] Fan, W., Gordon, M. D., Pathak, P., Xi, W. and Fox, E. A., Ranking function optimization for effective web search by Genetic Programming: an empirical study, in the *Proceedings of 37th Hawaii International Conference on System Sciences (HICSS)*, 2004.
- [8] Fan, W., Fox, E. A., Pathak, P. and Wu, H., The effects of fitness functions on genetic programming-based ranking discovery for web search, *Journal of the American Society for Information Science and Technology*, 55(7), 628-636, 2004.
- [9] Fan, W., Gordon, M. D., Pathak, P., A generic ranking function discovery framework by genetic programming for information retrieval, *Information Processing and Management*, in press, 2004.
- [10] Fan, W., Gordon, M. D., and Pathak, P., Discovery of context-specific ranking functions for effective information retrieval by Genetic Programming, *IEEE Transactions on Knowledge and Data Engineering*, 16(4), 523-527, 2004.
- [11] Fuhr, N., and Buckley, C., A probabilistic learning approach for document indexing, *ACM Transactions on Information Systems*, 9(3), 223-248, 1991.
- [12] Fuhr, N., and Pfeifer, U., Probabilistic information retrieval as combination of abstraction, inductive learning and probabilistic assumptions, *ACM Transactions on Information Systems*, 12(1), 92-115, 1994.
- [13] Gey, F. C., Inferring probability of relevance using the method of logistic regression, in the *Proceedings of Seventeenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 222-231, 1994.
- [14] Gordon, M. D., Probabilistic and genetic algorithms for document retrieval. *Communications of ACM*, 31(2), 152-169, 1988.
- [15] Harman, D., Relevance Feedback Revisited, in the *Proc. of the Fifteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, Ed. Nicholas Belkin, Peter Ingwersen and Annelise Mark Pejtersen, SIGIR Forum, June 21-24, 1992.
- [16] Holland, J. H., *Adaptation in Natural and Artificial Systems*, the University of Michigan Press, Ann Arbor, 1975.
- [17] Ide, E., New experiments in relevance feedback, in Gerard Salton, Editor, *The SMART Retrieval System*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1971.
- [18] Khan M. S., Khor S., Enhanced web document retrieval using automatic query expansion, *Journal of the American Society for Information Science and Technology*, 55(1), 29-40, Jan 2004.
- [19] Koza, J. R., *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, Cambridge, MA, USA: MIT Press, 1992.
- [20] Langdon, W. and Poli, R., *Foundations of Genetic Programming*. Springer-Verlag, 2002.
- [21] Lee, J. H., Analyses of multiple evidence combination, in the *Proceedings of Twentieth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 267-276, 1997.
- [22] Lundquist, C., Grossman, D. and Fireder, O., Improving relevance feedback in the vector space model, in the *Proceedings of the CIKM conference (CIKM97)*, (LasVegas Nevada, USA, 97), ACM Press, New York, NY, 16-23, 1997.
- [23] Pathak, P., Gordon, M. D., and W. Fan, Effective information retrieval using genetic algorithms based matching function adaptation, in *Proceedings of the 33rd Hawaii International Conference on System Science (HICSS)*, Hawaii, USA, 2000.
- [24] Robertson, S. E., On term selection for query expansion. *Journal of Documentation*. 46(4), 359-364, 1990.
- [25] Robertson, S. E., Walker, S., Jones S., Hancock-Beaulieu M. M., and Gatford, M., Okapi at TREC-4, in D. K. Harman, editor, *Proceedings of the Fourth Text Retrieval Conference*, 73-97. NIST Special Publication 500-236, 1996.
- [26] Rocchio, Jr., J.J., Relevance feedback in information retrieval, in Gerard Salton, Editor, *The SMART Retrieval System*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1971.
- [27] Salton, G. and Buckley, C., Term weighting approaches in automatic text retrieval, *Information Processing and Management*, 24(5), 513-523, 1988.
- [28] Salton, G. and Buckley, C., Improving retrieval performance by relevance feedback. *Journal of the American Society for Information Science*, 41(4), 288-297, 1990.
- [29] Vogt, C. C., and Cottrell, G. W., Fusion via a linear combination of scores, *Information Retrieval*, 1(3), 151-173, 1999.
- [30] Zobel, J. and Moffat, A., Exploring the similarity space, *SIGIR Forum*, 32(1): 18-34, 1998.