

# **Discovery of Context-Specific Ranking Functions for Effective Information Retrieval Using Genetic Programming**

**Weiguo Fan<sup>1</sup>**

Department of Accounting and Information Systems  
Virginia Polytechnic Institute and State University

**Michael D. Gordon**

Department of Computer and Information Systems  
University of Michigan

**Praveen Pathak**

Department of Decision Science and Information Systems  
University of Florida

**Keywords:** Intelligent Information Retrieval, Personalization, Search Engine, Term Weighting, Ranking Function, Text Mining, Genetic Programming, Contextual Information Retrieval, Information Routing, Information Retrieval

---

<sup>1</sup>Corresponding author. Address: 3007 Pamplin Hall, Blacksburg, VA 24061-0101, USA.  
Email: wfan@vt.edu, Tel: 540-231-6588, Fax: 540-231-2511

## Abstract

The Internet and corporate Intranets have brought a lot of information. People usually resort to search engines to find required information. However, these systems tend to use only one fixed ranking strategy regardless of the contexts. This poses serious performance problems when characteristics of different users, queries, and text collections are taken into account. In this paper, we argue that the ranking strategy should be context specific, and we propose a new systematic method that can automatically generate ranking strategies for different contexts based on *Genetic Programming* (GP). The new method was tested on TREC data and the results are very promising.

## 1. Introduction

The Internet has brought far more information than anybody can absorb. Similarly organizations store a large amount of information in manuals, procedures, documentation, expert knowledge, e-mail archives, news sources, and technical reports. Such a large amount of information serves as a huge information repository for organizations. However, it also makes finding relevant information from it extremely difficult. How to help users find their required information is the central task of any information retrieval (IR) system or search engine. However precision and recall, the two most commonly used performance measures, of commonly used search engines are usually very low [2].

Retrieval performance of an IR system can be affected by many factors: the ambiguity of query terms, unfamiliarity with system features, as well as factors relating to document representation [6]. Many approaches have been proposed to address these issues. For example, query expansion techniques based on a user's relevance feedback have been used to discover a user's real information need [3]. Similarly document descriptions have been modified [1]. Another very important factor that is often overlooked by most researchers is the ranking/matching function. It is this ranking function that we focus most of our discussion on.

A ranking function is used to order documents in terms of their predicted relevance to a particular query. It is very difficult to design such a ranking function that can be successful for

every query, user or document collection (which we will call contexts). In this paper, we argue in favor of a method that systematically adapts a ranking function and tailors it to different users' needs (i.e. in different contexts). In particular, we will use Genetic Programming (GP) [5], an inductive learning technique, for the adaptation purpose and compare our results against two well known retrieval systems.

Our paper is organized as follows: in section 2, we review related work on ranking functions and discuss our basic idea of adaptive ranking function. Section 3 describes our proposed ranking function discovery framework using Genetic Programming. Section 4 presents our research question. Section 5 summarizes the experimental setup for our evaluation study. We demonstrate the effectiveness of our method in Section 6 and conclude the paper in Section 7.

## 2. Term weighting strategy and ranking function

Among the many models, proposed in the IR literature, to index and represent documents as well as queries, the vector space model is the most widely used one. We will focus our discussion solely on this model because of its simplicity, intuitive geometric representation and its strong performance in TREC conference evaluations [4]. In this model, each query and document is represented as a vector of terms. Suppose there are total  $t$  index terms in an entire collection, a given document  $D$  and query  $Q$  can be represented as follows:

$$D=(w_{d1},w_{d2},w_{d3},\dots,w_{dt})$$

$$Q=(w_{q1},w_{q2},w_{q3},\dots,w_{qt})$$

where  $w_{di}$ ,  $w_{qi}$  ( $i=1$  to  $t$ ) are term weights assigned to different terms for the document and query respectively. These weights are commonly measured by their statistical properties or statistical features like *Term Frequency* (TF), which measures how many times a term has appeared in a document or query, and *Inverse Document Frequency* (IDF), which can be calculated by  $\log(N/DF)$ , where  $N$  is the total number of documents in the text collection and  $DF$  measures the number of documents in which a term has appeared in a document collection. More statistical features used in term weighting can be found in [11]. The similarity between the document and query vectors, as often measured by the Cosine function, is commonly used to represent the

retrieval status value (RSV) for that document. Documents are ordered by decreasing values of this measure. Since the calculation of the Cosine measure can be computationally very expensive, it is often replaced by the inner dot product, which is represented as follows:

$$Similarity(Q, D) = \sum_{i=1}^t w_{qi} w_{di} \quad (1)$$

It is to be noted that term weighting strategies have a great impact on the similarity calculation using Equation (1) and thus have a great impact on the final search and routing performance. Any change in the term weighting strategy will essentially change the ranking function itself, which is the underlying premise of our research. From now on, we will use these two terms *term weighting strategy* and *ranking function* interchangeably.

Research [13] shows that term weighting strategies using different combinations of statistical weighting features perform differently for the same query and the same document collection, and that no strategy works consistently well across all queries and text collections. Overall, then, there is no way to choose, *a priori*, a particular term weighting strategy. Nonetheless, any time a user issues a query to a particular document collection, there is, by definition, a best strategy.

In our work, we seek to find this best (or near best) strategy for the user. We propose using Genetic Programming to automatically determine - or approximate - such a strategy.

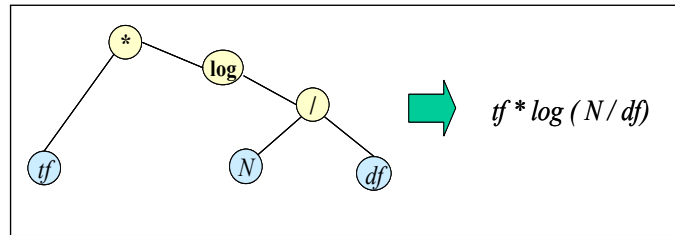
### **3. A framework for ranking function discovery by Genetic Programming**

We propose a GP-based method to the study of the problem of context-specific ranking function discovery by automatic term weighting. Our choice of using GP 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 eleven terminals (Table 1), four functions, trees of depth no more than ten, and real constants that vary between 0 and 1. Langdon et al. [7] show 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 [7]. GP has been shown [7] to perform well under such conditions.

Second, most performance measures in IR are discrete in nature (for example precision, recall, and in our case  $P_{avg}$  as explained later). GP does not require that objective functions be continuous in nature as long as it can differentiate good solutions from bad ones [5].

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. As shown in Figure 1, a typical ranking function formula as shown on the right side of the arrow is internally represented using a tree data structure as shown on the left side of the arrow. The tree-based representation allows for ease of parsing and implementation. We apply the ranking function formula to each term within a document.



**Figure 1 A sample ranking function represented in tree structure**

The detailed description of our Genetic Programming based method is as follows. We start with an introduction to the implementation details of the ranking function discovery framework using GP and conclude with an overview of the entire discovery framework.

### *Terminals*

We selected terminals (weighting features) after examining various term weighting and ranking formulas as in [11, 13]. The terminals used are listed in Table 1.

### *Functions*

Functions combine terminals and/or sub-trees to produce longer expressions. The following functions were used in our implementation:

$+$ ,  $\times$ ,  $/$ ,  $\log$

<i>Terminals</i>	<i>Statistical Meaning</i>
tf	Same as TF: how many times the term appeared in a document
tf_max	The maximum tf for a document
tf_avg	The average tf for a document
tf_doc_max	The maximum tf in the entire document collection
df	Same as DF: the number of unique documents the term appeared
df_max	The maximum df for the entire collection
N	The total number of documents in the entire text collection
length	The length of a document
length_avg	The average length of a documents in the entire collection
R	The real constant number randomly generated by the GP system
n	The number of unique terms in a document

**Table 1 Terminals used in our GP system**

### *Initial Population Generation*

The population is a set of individuals that represent document term weighting formulas (ranking trees). (For query terms weights, we use tf - which almost always produces a weight of 1 since most of these query terms appear only once in the query). The ranking trees in our initial generation were constrained to have a maximum depth of 10 levels and were generated by the Ramped Half-and-Half method [5] as this method has been found to generate a good initial sample of trees [5].

### *Fitness Functions*

We use *Average Precision* ( $P_{avg}$ : defined in Table 2 in section 5.2) in the top  $DCV^2$  documents as the fitness function. It is the most commonly used measure in text retrieval evaluations [4, 15].

### *The GP Operators*

The GP operators used in our implementation are *reproduction* and *crossover*.

- *Reproduction*: Here we rank the whole population according to their fitness values. The top ( $rate\_r * P$ ) trees in the current generation become members of the next generation without undergoing crossover, where  $rate\_r$  is the reproduction rate, and  $P$  is the population size.

---

<sup>2</sup>DCV, stands for Document Cutoff Value. It is the total number of documents returned to the user.

- *Crossover*: Crossover is the predominant operation in GP work. Its purpose is to preserve the ‘best genes’ across generations, improve the diversity of the population, and possibly discover better individuals [5]. For crossover, we used tournament selection [5]. It is implemented by first selecting, with replacement, six trees at random from the current generation. The two trees with the highest fitness are paired, and they exchange sub-trees. The exchanged sub-trees are selected at random, with the qualification that the newly generated trees be mathematically well-formed. This process repeats until the next generation has the same number of trees - including those copied directly - as the current generation.

### *Stopping Criterion*

Our preliminary experiments indicated that thirty generations was a sufficient period to generate effective trees. Hence we stopped the adaptation after thirty generations.

### *Overall Ranking Function Discovery Framework*

- 
1. Generate an initial population of random “ranking trees”;
  2. Perform the following sub-steps on *training* documents for thirty generations
    - For each ranking tree, use it to score and rank documents
    - Calculate the fitness of each ranking tree
    - Record the top 10 ranking trees
    - Create new population by:
      - Reproduction
      - Crossover
  3. Apply the recorded (30\*10=300) candidate “ranking trees” on a set of *validation*<sup>3</sup> documents and select the best performing tree
- 

## **4. Research Question**

We seek to answer the following question in this study:

*How effective is the retrieval performance of our context-specific ranking discovery methodology based on GP, in comparison to other well known systems?*

We chose one fixed ranking strategy Okapi BM25 [10] and one learning strategy based on neural network (NN) [9, 12] as our baselines for comparisons. Okapi BM25 was chosen as it has consistently produced good results in TREC evaluations, while the neural network learning

---

<sup>3</sup>The validation data set is used to help select a tree that generalizes well for unseen documents and thus to avoid the effect of over-training.

method was chosen as we wanted to compare the results against another well known learning methodology.

## 5. Experimental Setup

The TREC conference is the primary conference that is focused on the applicability study of traditional as well as new IR techniques on large-scale text collections [4]. Many information retrieval systems and commercial engines have used TREC data as the test bed to evaluate and validate their performance. We will follow this practice as well.

### 5.1 Data

We use the text corpus from Associate Press, which consists of three years (1988-1990) news-wire documents. It covers a broad variety of domains and has 242,918 documents with an average length of 450 words.

To address the overfitting problems in GP implementations we follow a three data-set design [8]. We use AP88 (79919 documents) as the training data, AP89 (84678) as the validation data and AP90 (78321) as the test data. The training data is used to train GP to get potential high performing ranking trees. The validation data is used to help select the best performing tree from the candidate set that can generalize well. The test data is used to report the final predictive performance for the tree selected after validation.

There are 50 different user-provided topics (or queries) used in the TREC 7 experiment. All the 242,918 documents in the AP collection were judged for relevance by experts. Because 15 of the 50 queries have very few relevant documents in both the training and validation set (we use threshold of 4), 35 out of 50 topics were included in our experiments. We leave the examination of the effect of having very few relevant documents on our methodology to future experiments.

We indexed two different versions of these topics. In the *short* version, only the title and description portion of the topics were indexed to simulate the real web search scenario in which the length of search queries is very short. The number of terms contained in these short queries varied from 4 to 18. In the *longer* version, we indexed all the components of the 35 topics (title, description, narrative, and concepts). The number of terms for long queries varied from 17 to 96.

This represents situations where a user is willing to provide a lengthy description about his/her information need for more accurate search.

## 5.2 Performance measures

There are various performance measures defined in TREC evaluations as listed in Table 2.

<i>Measure</i>	<i>Definition</i>
P_avg	The average of precisions every time a new relevant is found, normalized by the total number of relevant documents in an entire collection
R_P	The precision when T_Rel documents are retrieved.
T_Recall	The recall of the top 1000 documents retrieved.

**Table 2 Performance measures and their definitions**

Among these measures, P\_avg is the dominant one used in TREC routing experiment evaluations. The definition of P\_avg reveals that it is a hybrid measure of both recall and precision. Thus it is very suitable for cross-system comparison in terms of overall performance [15]. We use P\_avg as the primary measure for comparison as well.

## 5.3 Experimental design

The preliminary experiments on several queries suggested that we use large population size and different random seeds<sup>4</sup> on the TREC data. Rest of the GP settings follow the suggestions of [5]. Thus our experiment is set up as follows:

<b>Query</b>	35 topics (each short and long)			
<b>Training Set</b>	79,919 documents from AP88			
	DCV	1000	Crossover	0.9
	Population size	200	Reproduction	0.1
	Random seeds	5	Generations	30
<b>Validation Set</b>	84,678 documents from AP89			
<b>Test Set</b>	78,321 documents from AP90			

## 5.4 Experimental procedure

The experiment proceeds as follows: for each topic (both short and long versions), we train the GP system using the AP88 training data set. At the end of each generation of the GP training process, the 10 best trees for that generation are validated on the AP89 data and the results are recorded. After the training and validation, the best tree is selected based on the average of its

<sup>4</sup>Random seed impacts population initialization, which will accordingly affect the final learning results.

performance on both training data and validation set and it is then applied to the test data. Its corresponding performance result in the test data is the predictive performance of this best tree.

We use the three data set design to largely avoid the over-fitting problem. Obviously, the best tree selected using the training and validation data set is expected to perform well in the test data set. However, we need to check how this tree fares against the best tree selected based solely on test data performance result (the upper-bound performance). We will compare this upper-bound performance with the predictive performance to see how much performance is lost by following the three data set design. This will help us understand if the over-fitting problem has been addressed.

For comparison with other well known systems, we implemented the Okapi BM25 system [10]. Similarly neural network was implemented as per the design in [12]. BP algorithms [8] was used to train the NN. These two systems will be used as benchmarks for comparison.

## 6. Results

We compared the GP results for both short and long queries with their corresponding upper-bound performance (refer to experimental procedure section) in terms of P\_avg and found that there is less than 8% performance drop in both cases. We consider such small discrepancies reasonable and consider the over-training problem as largely solved in our methodology.

We graphically compared<sup>5</sup> the three systems (GP, Okapi, and NN) for all 35 queries on the P\_avg measure.<sup>6</sup> It was seen that except for a few queries, GP based system outperforms the other two systems (Okapi and NN) for retrieval. To test for the statistical significance of these differences we performed pair-wise t-tests on P\_avg. Table 3 summarizes the aggregated performance comparison (on P\_avg) between GP, Okapi, and NN systems.

As can be seen from this table, the query effect is quite dramatic. In the GP system, the performance improvement was 44% when long queries were used compared to short queries. In

---

<sup>5</sup> Due to space limitations graphical results are not shown. Please contact authors for more details.

<sup>6</sup> Due to space constraints results for other performance measures are not shown. More details available from authors.

the Okapi and NN systems, such improvements were 35% and 109% respectively. This reflects the importance of accurate representation of users' information needs.

Query	Systems			Improvement of GP over baselines	
	GP	Okapi	NN	GP – Okapi	GP – NN
Short	0.25	0.23	0.11	+10.71%*	+130.74%*
Long	0.36	0.31	0.23	+17.01%*	+56.08%*
<b>Improvement of Long over Short</b>	44%	35%	109%		

**Table 3 Performance comparison of GP with Okapi and NN systems on P\_avg**

Overall, GP outperforms Okapi in P\_avg by 10.71% for short queries and by more than 17% for long queries. Similarly GP outperforms NN by 130% for short queries and by 56% for long queries. All these are statistically significant differences.

Using these results we can conclude that our method using GP yields more effective retrieval compared to the results obtained from Okapi and NN systems as discussed previously. Using the categorization of differences in performance as suggested in [14], we can say that our performance is ‘materially’ (more than 10% improvement) better than that obtained by the two baseline systems used.

Our analysis of the top performing functions that GP produced<sup>7</sup> indicated that GP discovered some of the well known ranking strategies like TFIDF ( $tf * N/df$ ) and novel normalization factors. It can be argued that OKAPI itself should be discovered by our GP methodology. Our analysis indicates that OKAPI did not feature in the top performing GP trees. This is so because our methodology was not explicitly meant to discover OKAPI function but was meant to discover trees that would significantly enhance retrieval performance. Our results indicate that our methodology yields to ‘materially’ [14] better retrieval results.

## 7. Conclusions and future research

\* Indicates the differences are statistically significant at  $p < 0.05$  using the paired  $t$ -test.

<sup>7</sup> Due to space constraints the actual functions are not shown. They are available upon request.

In this paper we have introduced a novel method to context-specific ranking function discovery by Genetic Programming. Since most search engines built on the vector space model use only a fixed ranking function for all contexts (the so-called “one size fits all” limitation), this poses serious problems when they are used to satisfy a wide range of user information needs. Instead, by combining the common statistical clues or features used in the traditional term weighting strategies in an intelligent way by GP, we found that we can significantly improve the retrieval performance. The results on the TREC conference data have demonstrated the advantages of our method.

We have also addressed the issue of overfitting posed by GP learning. Our three data set design helped us overcome such a problem. We also studied the effect of query specificity. Two different versions of queries (short and long) were used in the experiment and were found to make substantial difference in performance. This exemplifies the fact that better query representation may lead to better ranking function discovery. We compared the retrieval performance of our GP system to those obtained by two other well known systems. It was observed that our system produces ‘materially’ better [14] retrieval results than those obtained with the other two systems. We can thus conclude that a GP based optimization of ranking functions is a viable method in information retrieval. We believe our work has important implications for designing adaptive search engines, and context-specific information discovery and delivery.

The GP framework discussed in this paper is used on a single query, and thus is more appropriate for filtering/routing tasks in retrieval. We intend to extend this framework to discover functions for multiple queries and for various contexts (different users, underlying document datasets such as WWW and XML corpus). We also plan to build a prototype system based on the framework and perform extensive real-user tests to see its viability.

## **Acknowledgments**

We are grateful to the AE and three anonymous referees for many helpful comments.

## **References**

- [1] M. Gordon. Probabilistic and genetic algorithms for document retrieval. *Communications of ACM*, 31(2): 152–169, 1988.
- [2] M. Gordon and P. Pathak. Finding information on the WWW: the retrieval effectiveness of search engines. *Information Processing and Management*, 35(2): 141–180, 1999.
- [3] D. K. Harman. Relevance feedback revisited. In *Proceedings of the 11th ACM SIGIR conference*, pages 321–331, 1992.
- [4] D. K. Harman. Overview of the fourth text retrieval conference (TREC-4). In D. K. Harman, editor, *Proceedings of the Fourth Text Retrieval Conference*, pages 1–24. NIST Special Publication 500-236, 1996.
- [5] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA, 1992.
- [6] F. W. Lancaster and A. J. Warner. *Information Retrieval Today*. Information Resources Press, 1993.
- [7] W. Langdon, and R. Poli. *Foundations of Genetic Programming*. Springer-Verlag, 2002
- [8] T. M. Mitchell. *Machine Learning*, volume McGraw Hill. 1997.
- [9] H. Ng, W. Goh, and K. L. Low. Feature selection, perceptron learning, and a usability case study for text categorization. In *Proceedings of the 20th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, New York, NY, 1997. ACM press.
- [10] S. E. Robertson, S. Walker, S. Jones, M. M. Hancock-Beaulieu, and M. Gatford. Okapi at TREC-4. In D. K. Harman, editor, *Proceedings of the Fourth Text Retrieval Conference*, pages 73–97. NIST Special Publication 500-236, 1996.
- [11] G. Salton. *Automatic Text Processing*. Addison-Wesley Publishing Co., MA, 1989.
- [12] H. Schutze, D. Hull, and J. Pedersen. A comparison of classifiers and document representations for the routing problem. In *Proceedings of the 16th ACM SIGIR'95*, pages 229–237, Seattle, USA, 1995.
- [13] A. Singhal, G. Salton, M. Mitra, and C. Buckley. Document length normalization. *Information Processing and Management*, 32(5): 619–633, 1996.
- [14] Karen Sparck Jones. Automatic Indexing. *Journal of Documentation*, 30:393-432, 1974.
- [15] E. M. Voorhees. Variations in relevance judgments and the measurement of retrieval effectiveness. *Information Processing and Management*, 36(5): 697–716, 2000.