

The Effects of Fitness Functions on Genetic Programming-Based Ranking Discovery For Web Search

*Weiguo Fan** *Edward A. Fox*

Virginia Polytechnic Institute and State University

Praveen Pathak

University of Florida

Harris Wu

University of Michigan

Abstract

Genetic-based evolutionary learning algorithms, such as genetic algorithms (GAs) and genetic programming (GP), have been applied to information retrieval (IR) since the 1980s. Recently, GP has been applied to a new IR task — discovery of ranking functions for web search — and has achieved very promising

*Corresponding author. Email: wfan@vt.edu. Tel: (540) 231-6588. Fax: (540) 231-2511. Mailing address: 3007 Pamplin Hall, Blacksburg, VA, 24061.

results. However, in our prior research, only one fitness function has been used for GP-based learning. It is unclear how other fitness functions may impact ranking function discovery for web search, especially since it is well known that choosing a proper fitness function is very important for the effectiveness and efficiency of evolutionary algorithms. In this paper, we report our experience in contrasting different fitness function designs on GP-based learning using a very large web corpus. Our results indicate that the design of fitness functions is instrumental in performance improvement. We also give recommendations on the design of fitness functions for genetic-based information retrieval experiments.

1 Introduction

Genetic algorithms (GAs) (Holland, 1992) and genetic programming (GP) (Koza, 1992) are a set of artificial intelligence search algorithms based on evolutionary theory. They represent the solution to a problem as an individual (also called chromosome) in a population pool. They evolve the population of individuals (chromosomes) generation by generation following the genetic transformation operations — such as reproduction, crossover, and mutation — with the aim of discovering chromosomes with better fitness values. A fitness function is available to assign the fitness value for each individual.

The difference between GA and GP is the internal representation — or data structure — of the individual chromosome. In GAs, each individual is commonly (though not always) represented by a fixed-length bit string, like (1101110...) or a fixed-length sequence of real numbers (1.2, 2.4, 4, ...). In GP, more complex data structures, e.g.,

tree, linked list, or stack, are used (Langdon, 1998). Moreover, the length or size of the data structure is not fixed, although it may be constrained to within a certain size limit by implementation. GAs are often used to solve difficult optimization problems, while GP is typically used to approximate complex, nonlinear functional relationships (Koza, 1992). Because of the intrinsic parallel search mechanism and powerful global exploration capability in a high-dimensional space, both GA and GP have been used to solve a wide range of hard optimization problems that oftentimes have no best known solutions.

Because of these merits, there has been increasing interest in applying GAs and GP to intelligent information retrieval (IR) in recent years (Chen, 1995; Chen et al., 1998a,b; Cordon et al., 2000, 2002; Gordon, 1988, 1991; Fan et al., 1998, 2000, 2003a,b, 2004; Horng and Yeh, 2000; Kraft et al., 1994, 1995, 1997; Lopez-Pujalte et al., 2002, 2003; Martin-Bautista et al., 1999; Pathak et al., 2000; Robertson and Willett, 1994, 1996; Smith and Smith, 1997; Vrajitoru, 1998; Yang and Korfhage, 1993a,b; Yang et al., 1993). The application areas cover a wide range of IR topics such as document indexing; query induction, representation, and optimization; document clustering; and document matching and ranking.

Fan et al. previously reported applying GP to discover ranking functions for both individual queries (information routing tasks) (Fan et al., 1998, 2000, 2003a) and multiple queries (ad-hoc retrieval tasks) (Fan et al., 2003b, 2004). Given a query (or a set of queries), a ranking function is used by a search engine to rank documents according to their match with a particular query. Since there is no known best ranking function for a

query (or a set of queries), we model this problem as a GP search problem. Candidate ranking functions are represented as individuals in a GP population using a tree structure, and then evolved by GP to discover ranking functions with better fitness values. The fitness function used was *average precision* (also called PAVG, defined in Section 3). It is known in the GA/GP literature that the design of a good fitness function is vital to the search success rate (Koza, 1992). Although the PAVG fitness function helped GP discover and achieve reasonable performance improvement over many existing solutions (Fan et al., 2003a,b, 2004), it is still unclear whether other fitness functions may guide GP to perform better. This matter is what we discuss in this paper.

The fitness functions used in GA/GP learning in retrieval experiments can generally be classified into two types: one takes into account only the number of relevant documents retrieved among the top n hits of a search; the other takes into account not only the number of relevant documents retrieved, but also their relative position in the ranked list. The latter is called an *order-based fitness function* (Lopez-Pujalte et al., 2003). Lopez-Pujalte et al. (2003) compared a number of commonly used fitness functions (both with and without the relevance order information) in GA/GP studies and found that the order-based ones get more favorable results. However, their study tested only a *limited number* of order-based fitness functions on a set of *small* IR collections at the *individual query level* using *genetic algorithms (GAs)*. Our study complements their study by comparing *more order-based fitness functions* in the web search context using a *much larger text corpus*. Our aim is to use *genetic programming (GP)* to optimize *the ranking function* for *multiple queries*. The additional evidence from this study will pro-

vide richer insights regarding the merits of various order-based fitness functions. It will give future researchers guidelines and more choices on the usage of order-based fitness functions in genetics-based evolutionary learning.

Our paper is organized as follows. In Section 2 we briefly review the framework of applying genetic programming (GP) to ranking function discovery in our prior work. In Section 3 we formally describe and compare a number of order-based fitness functions. Section 4 presents the experimental setup. We summarize the experimental results in Section 5 and conclude the paper in Section 6.

2 Ranking Function Discovery by GP

As mentioned above, a ranking function is used by a search engine to rank documents according to their matching scores with a particular query. Since there is no known best ranking function for a query (or a set of queries) (Salton and Buckley, 1988; Zobel and Moffat, 1998), the problem of ranking function discovery is modeled as a GP search problem. Candidate ranking functions are represented as individuals in a GP population using tree structures, and then are evolved by GP to discover better performing ranking functions than existing ones (Fan et al., 2000, 2003a,b, 2004).

To apply GP for ranking function discovery, each ranking function is represented as an individual (chromosome) in a GP population. An example of such a representation for the famous TFIDF ranking is shown in Figure 1.

Several required key components of a GP system are defined in Table 1. Figure

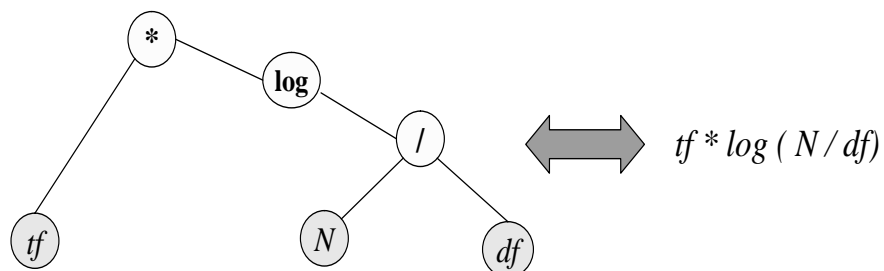


Figure 1: A sample tree representation for the TFIDF ranking function

Table 1: Essential GP components.

<i>Components</i>	<i>Meaning</i>
Terminals	Leaf nodes in the tree structure. i.e. df , N , as in Figure 1.
Functions	Non-leaf nodes used to combine the leaf nodes. Commonly, numerical operations: $+$, $-$, $*$, $/$
Fitness function	The objective function GP aims to optimize.
Reproduction	A genetic operator that copies the individuals with the best fitness values directly into the population of the next generation without going through the crossover operation.
Crossover	A genetic operator that exchanges sub-trees from two parents to form two new children. Its aim is to improve diversity as well as the genetic fitness of the population. This process is shown in Figure 2.
Mutation	A genetic operator that randomly selects a sub-tree and replaces it with another randomly-created one. Its aim is to improve diversity as well as prevent being trapped in a local optimal solution.

2 illustrates graphically the process of crossover operation. It can be easily seen from Figure 2 that the two children generated after the crossover operation are different from their parents. The crossover operation can not only allow children to inherit good “genes” from their parents, but also improve the diversity of the population. Thus it is used extensively in genetic programming (Koza, 1992).

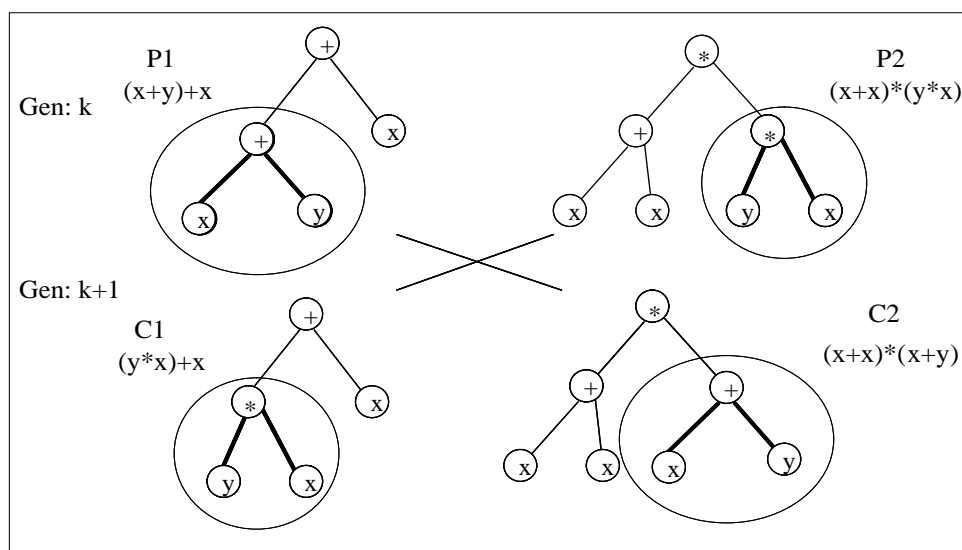


Figure 2: A graphical illustration of the crossover operation.

The configuration of the GP system used for ranking function discovery is set up as shown in Table 2.

Table 2: Modeling setup for ranking function discovery by GP. Refer to Table 1 for explanations of the various components.

Components	Values
Terminals	Features shown in Table 3 as terminals.
Functions	+, ×, /, log
Fitness function	Various ones designed in Section 3
Genetic operator	Reproduction, Crossover, Mutation

The overall ranking function discovery framework is shown in Figure 3.

Table 3: The terminals used in our GP system

Terminals	Meaning
tf	Number of occurrences a term in a document
tf_max	Maximum tf for a document
tf_avg	Average tf for a document
tf_doc_max	Maximum tf in the document collection
df	Number of unique documents with a term
df_max	Maximum df for a given query
N	Total number of documents in the text collection
length	Length of a document
length_avg	Average length of a document in the collection
R	Real constant number randomly generated by the GP system
n	Number of unique terms in a document

1. Generate an initial population of random “ranking trees”.
2. Perform the following sub-steps on training documents for N_{gen} generations:
 - 2.1 Use each ranking tree to score and rank the collection separately for multiple queries,
 - 2.2 Calculate the fitness of each ranking tree,
 - 2.3 Record the top N_{top} ranking trees,
 - 2.4 Create new population by:
 - a) Reproduction
 - b) Crossover
 - c) Mutation
3. Apply the recorded ($N_{gen} \times N_{top}$) candidate “ranking trees” to a set of validation documents and select the best performing tree as the unique discovery output.

Figure 3: Overall Ranking Function Discovery Framework. N_{gen} and N_{top} are user-specified parameters. We set $N_{gen} = 30$ and $N_{top} = 10$ for all our experiments.

3 Design of Fitness Functions

The fitness function plays a very important role in guiding GA/P to obtain the best solutions within a large search space. Good fitness functions will help GA/P to explore the search space more effectively and efficiently. Bad fitness functions, on the other hand, can easily make GA/P get trapped in a local optimum solution and lose the discovery power.

Our experience applying GA and GP to ranking function design and discovery suggests that the fitness functions for information retrieval experiment should take into account both the number of relevant documents and the order in which they appear in the ranked list, which is in line with the findings in (Lopez-Pujalte et al., 2003). The following example illustrates this point.

Suppose two ranking functions both retrieve 5 relevant documents in the top 10 results. Their relevance information (1 being relevant and 0 being non-relevant) in the ranking list are shown as follows:

Rank list 1: 1 0 1 1 0 0 1 0 0 1

Rank list 2: 1 1 0 1 1 1 0 0 0 0

If ranking order is ignored, the performance of the these two rank lists is the same — both returning 5 relevant documents. However, we should prefer the second rank list over the first because the second rank list presents relevant documents sooner (i.e., has higher precision).

3.1 A utility-theoretic view on fitness function design

We approach fitness function design following the principles of utility theory (Fishburn, 1988). According to utility theory, there exists a utility function (a user’s preference function) that assigns a utility value (the gained value from a user’s perspective) for each item. These values vary from item to item. The item can be a book, a product, or a document as in our case. In general, we assume *the utility of a relevant document decreases with its ranking order*. More formally, given a utility function $U(x)$, and two ranks x_1, x_2 , with $x_1 < x_2$, according to this assumption, we expect the following condition to hold:

$$U(x_1) > U(x_2) \tag{1}$$

We call Equation (1) the *order preserving condition* in this paper.

For example, suppose the top two documents in a search rank list are both relevant. According to Condition (1), the first relevant document in the rank list should give a reader more utility value than the second one. The further down the rank list, the less utility a relevant document gives. This relationship can be illustrated in Figure 4.

The question is how to define the utility function. There are many possible functions that can be used to model this utility function satisfying the order-preserving condition given in (1). To name but a few: $f(x) = 1/x$; $f(x) = p^x, p \in (0, 1)$; $f(x) = \log_{10}(1/x)$; etc. However, some of these functions may decrease too fast as x increases such as $f(x) = 1/x$ or $f(x) = p^x, p \in (0, 1)$. An ideal utility function should look like the one

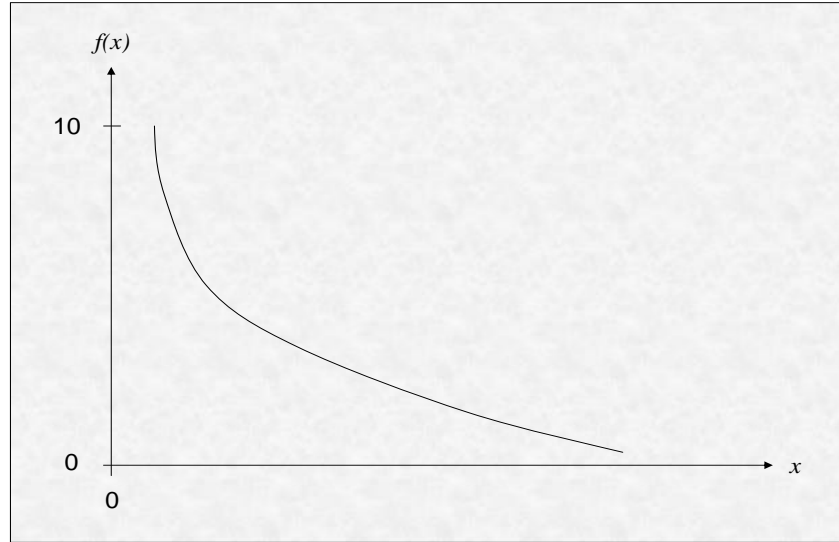


Figure 4: An ideal utility function $f(x)$ for rank x

shown in Figure 4. As can be seen from Figure 4, the ideal utility function is a nonlinear function of x . It favors the documents ranked at the top (smaller x). As x gets bigger, it quickly loses value. Further, the utility function should be bounded within a certain range.

Now we will detail the fitness functions that we designed which satisfy Condition (1). It is to be noted that these functions and the associated parameters were chosen after a great deal of exploratory data analysis.

– FFP1

$$Fitness_{FFP1} = \sum_{i=1}^{|D|} r(d_i) \times k_1 \times \ln^{-1}(i + k_2) \quad (2)$$

where $r(d) \in (0, 1)$ is the relevance score assigned to a document, it being 1 if the document is relevant and 0 otherwise. The relevance information is obtained

from users during the relevance feedback. The fact that $r(d) = 0$ for nonrelevant documents reflects the model assumption that *nonrelevant documents contribute nothing in utility calculation*. This makes sense because nonrelevant documents do not contribute anything useful for a user's search and thus nonrelevant documents should be assigned a utility of 0. We make this assumption for all of our designed fitness functions. $|D|$ is the total number of retrieved documents. k_1, k_2 are scaling factors. After exploratory analysis we set $k_1 = 6$ and $k_2 = 1.2$ in our experiment. This function is called *FFP1* to denote the first fitness function designed by the authors.

– FFP2

$$Fitness_{FFP2} = \sum_{i=1}^{|D|} r(d_i) \times k_3 \times \log_{10}(1000/i) \quad (3)$$

where $r(d)$, $|D|$ are the same as in Equation (2). k_3 is a scaling factor. We set $k_3 = 2$ in our experiment. This function is called *FFP2* to denote the second fitness function designed by the authors.

– FFP3

$$Fitness_{FFP3} = \sum_{i=1}^{|D|} r(d_i) \times k_4^{-1} \times (e^{-k_5 \times \ln(i) + k_6} - k_7) \quad (4)$$

where $r(d)$, $|D|$ are the same as in Equation (2). k_4, k_5, k_6, k_7 are scaling factors that are set as 3.65, 0.1, 4, and 27.32, respectively. This function is called *FFP3* to denote the third fitness function designed by the authors.

– FFP4

$$Fitness_{FFP4} = \sum_{i=1}^{|D|} r(d_i) \times k_8 \times k_9^i \quad (5)$$

where $r(d)$, $|D|$ are the same as in Equation (2). Two scaling factors, k_8 and k_9 , are set as 7 and 0.982, respectively. This function is called *FFP4* to denote the fourth fitness function designed by the authors.

The utility assigned by these four fitness functions in relation to the rank i is depicted in Figure 5. It is to be noted that some functions are steeper than others. Such a property may affect its performance as will be clear from experiments in future sections.

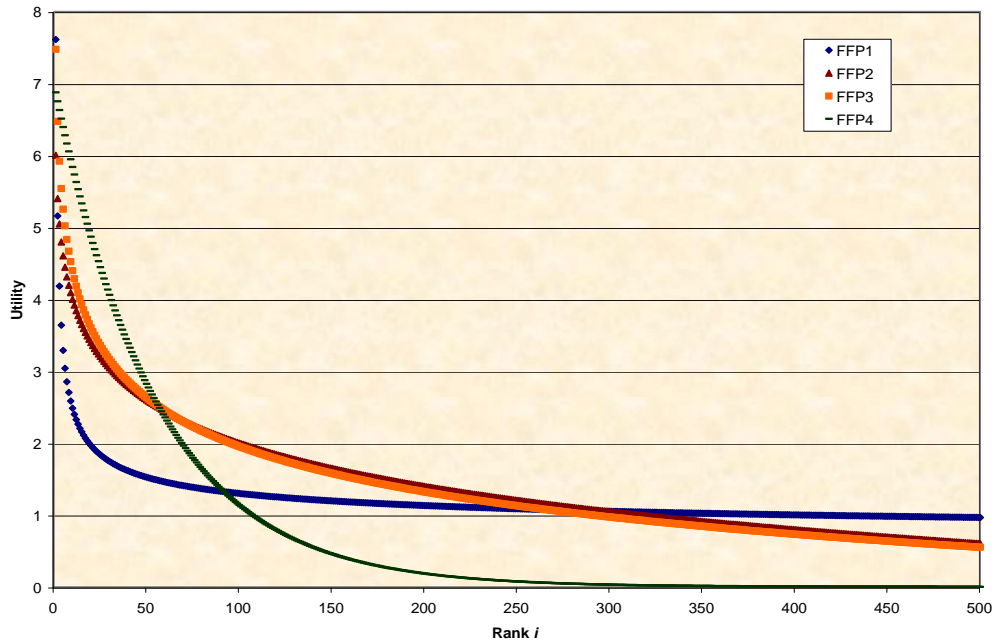


Figure 5: Comparison of four utility functions

3.2 Other order-based fitness functions

There are several other fitness functions that take into account the order or rank information into the fitness calculation.

– PAVG

PAVG, standing for *average precision*, is a commonly used performance measure used in TREC evaluations (Harman, 1996). It can be formally defined as follows:

$$Fitness_{PAVG} = \frac{\sum_{i=1}^{|D|} \left(r(d_i) \times \left(\frac{\sum_{j=1}^i r(d_j)}{i} \right) \right)}{TRel} \quad (6)$$

where $r(d) \in (0, 1)$ is the relevance score assigned to a document, being 1 if the document is relevant and 0 otherwise. $|D|$ is the total number of retrieved documents. $TRel$ is the total number of relevant documents in a collection.

PAVG is the primary fitness function used in IR. It is also mathematically very similar to the *interpolated average precision* as used in (Lopez-Pujalte et al., 2002, 2003).

However, the component $— \sum_{j=1}^i r(d_j)/i —$ in Equation (6) is not a utility function because it does not satisfy the order preserving condition set in (1). For example, given a 1 0 1 1 ranking sequence, substituting i with 1, 2, 3, 4 in Equation (6), we get 1, 0.5, 0.67, 0.75. It is easy to see that the fourth ranked document has higher value than the third ranked relevant document, which violates the utility order preserving condition set in (1). Whether using a utility function in fitness

functions gives any edge in performance over using the non-utility one will be explored later in the experiments.

– CHK

The CHK fitness function is defined as follows:

$$Fitness_{CHK} = \frac{1}{|D|} \sum_{i=1}^{|D|} \left(r(d_i) \times \sum_{j=i}^{|D|} \frac{1}{j} \right) \quad (7)$$

The same notation as in PAVG is used here. This fitness has been used for GA-based experiments in (Chang, 1999; Kwok, 1997; Lopez-Pujalte et al., 2002, 2003).

Because it was first introduced by Chang and Kwok, we denote it as the *CHK* fitness function for ease of discussion.

Note that in Equation (7), $\sum_{j=i}^{|D|} \frac{1}{j}$ can be treated as a discrete utility function and it satisfies the order preserving condition set in (1).

– LGM

The last fitness function we consider in this paper is the fitness function used in (Lopez-Pujalte et al., 2002, 2003):

$$Fitness_{LGM} = \left(\sum_{i=1}^{|D|} \left(r_B(d_i) \times \frac{1}{A} \left(\frac{A-1}{A} \right)^{i-1} \right) \right) \times \frac{\sum_{i=1}^{|D|} r(d_i)}{|D|} \quad (8)$$

where $r_B(d) \in (1, -1)$ is a function returning the relevance of document d , being +1 if d is relevant, -1 otherwise. $r(d)$ and $|D|$ are the same as in Equation (2).

A is a user-defined parameter. If we set A to 2 as in (Lopez-Pujalte et al., 2002,

2003), then Equation (8) can be simplified as

$$Fitness_{LGM} = \left(\sum_{i=1}^{|D|} (r_B(d_i) \times 2^{-i}) \right) \times \frac{\sum_{i=1}^{|D|} r(d_i)}{|D|} \quad (9)$$

For ease of reference and discussion, we denote this fitness function as the *LGM* fitness function, taking the first letter from the authors' surnames.

It is not very difficult to see that 2^{-i} also can be treated as a utility function, satisfying the order preserving condition set in Equation (1). However, 2^{-i} has a much steeper curve in assigning utility values (documents after rank 20 receiving utility of almost 0). Moreover, LGM assigns negative utilities to nonrelevant documents ($r_B(d_i) \times 2^{-i} < 0$ if d_i is nonrelevant). This is quite different from all the previously designed fitness functions which simply ignore all nonrelevant documents.

We need to test whether this design will lead to good search performance.

4 Experiment

The aim of the experiment is to test the effects of various order-based fitness functions on GP-based ranking discovery in a web search context.

4.1 Baselines

To be consistent with previous experiments (Fan et al., 2003a,b, 2004), we use the Okapi BM25 ranking formula (Robertson et al., 1996) as the baseline for this experiment. Okapi BM25 is designed based on the 2-Poisson model and has consistently performed very well

in TREC competitions (Robertson et al., 1996; Hawking, 2000; Hawking and Craswell, 2001). More formally, given a query Q and document D , the Okapi BM25 ranking formula is defined as follows:

$$Sim_O(Q, D) = \sum_{T \in Q} \frac{(k_1 + 1)tf}{k_1 \times ((1 - b) + b \frac{length}{length_{avg}}) + tf} \times \log \frac{N - df + 0.5}{df + 0.5} \times qtf \quad (10)$$

where

tf is the term frequency of a term (word) in the document text.

qtf is the term frequency of a term (word) in the query text.

N is the total number of documents in the collection

df is the number of documents in the collection in which the term under consideration is present.

$length$ is the length of the document (in words).

$length_{avg}$ is the average document length in the collection (in words).

k_1, b are the parameters used to fine-tune the search performance.

We use the same value as in Robertson et al. (1996) for k_1 and b : $k_1 = 2$ and $b = 0.75$.

So Equation (10) becomes:

$$Sim_O = \sum_{T \in Q} \frac{3 \times tf}{0.5 + 1.5 \times \frac{length}{length_{avg}} + tf} \times \log \frac{N - df + 0.5}{df + 0.5} \times qtf \quad (11)$$

4.2 Test collection

We use the widely-used testbed for web search — TREC 10GB collection (Hawking, 2000) — as our test collection. It contains 1.69 million documents. On average, each document has 311 words. We use 100 queries constructed from topics 451-550 used in TREC 9 and TREC 10 web tracks as the test queries. The length of the queries varies from 2 to 13. We index all documents and queries into the standard vector space format after applying stop-word removal and stemming (Salton, 1989).

4.3 Experimental design

Since the purpose of the experiment is to test the effects of fitness functions on GP-based ranking function discovery for multiple queries using feedback information, this experiment can be qualified as a supervised learning task. We follow a three data-sets design (Mitchell, 1997; Fan et al., 2003a,b) in this experiment. We randomly split the data into training (49%), validation (21%), and test (30%) parts. The introduction of the validation data-set is to help alleviate the problem of overfitting of GP on the training data and select the best generalizable ranking function. All performance results are reported based on the test data-set. The detailed experimental procedure follows the framework articulated in Figure 3.

We test the GP-based discovery framework using each of 7 fitness functions described earlier. There are thus total 7 independent experiments. Each experiment is run 10 times using different random seeds. The best result among the 10 runs is reported and used for comparison.

4.4 Performance evaluation

We report performance results using the standard *non-interpolated average precision* (PAVG) and *precision at the top 10 hits* (P10) (Harman, 1996; Hawking and Craswell, 2001). These are the most popular measures for reporting performance of different IR systems. The Results section shows the average results of these two measures over 100 queries.

For each of the 7 experiments, we also report the training time GP uses to finish the 10-run experiment. All experiments are run on a 4-1.2GHz processor Linux server with 2GB memory.

5 Results

The experimental results using various fitness function designs, in comparison with the baseline, are summarized in Table 4.

Table 4: Results of GP-based ranking function discovery using different fitness functions and their comparison with the baseline.

Method	PAVG	P10	Training Time
Okapi BM25	0.2230	0.2404	—
GP with FFP1	0.2203(-1.21%)	0.2420(+0.67%)	75 hours
GP with FFP2	0.2895(+29.82%)	0.2932(+21.96%)	80 hours
GP with FFP3	0.2899(+30.00%)	0.2932(+21.96%)	67 hours
GP with FFP4	0.2995(+34.30%)	0.3114(+29.53%)	61 hours
GP with PAVG	0.2509(+12.51%)	0.2511(+4.45%)	69 hours
GP with CHK	0.2666(+19.55%)	0.2580(+7.32%)	70 hours
GP with LGM	0.2136(-4.22%)	0.2386(-0.75%)	84 hours

As can be seen from Table 4, three out of the four fitness functions (FFP1 – FFP4)

of our own design perform extremely well — achieving more than 20% improvement over the baseline. The best performance is obtained using FFP4 (shown in Equation 5), which achieves more than 34% improvement in PAVG and more than 29% in P10 performance measure over the baseline Okapi BM25 ranking function.

As mentioned earlier, prior work using GP based IR used PAVG as the fitness function. Although using PAVG gives reasonable performance improvement over the baseline (12.51% in PAVG and 4.45% in P10), the superior experimental results using the FFP2, FFP3, and FFP4 confirm our hypothesis that there exist other more effective fitness functions that may help guide GP to do more effective retrieval. Our fitness function design based on utility theory has clearly been beneficial.

In terms of search efficiency, the FFP4 uses the least amount of search time when finishing the 10-run experiment. Note that the search time of GP is largely dependent on several factors: the complexity of the tree size (complex trees take more time to evaluate than simple trees), the size of the population, and the fitness calculation. Since we fix the population size for all fitness function experiments and the fitness calculation time is negligible compared to the tree evaluation, the search time depends mainly on the complexity of the ranking trees in the GP population. It turns out from the post-experiment study that trees discovered by FFP4 are less complex (of smaller depth or size) than ones obtained using other fitness functions. This seems a nice property of FFP4 considering the fact that less complex trees means they can be more easily interpreted by human beings.

In order to see more clearly why some fitness functions perform better than others,

we compare the utility functions used in FFP1 (the second worst) with the top 4 fitness functions (FFP4, FFP3, FFP2, CHK) in Figure 6.

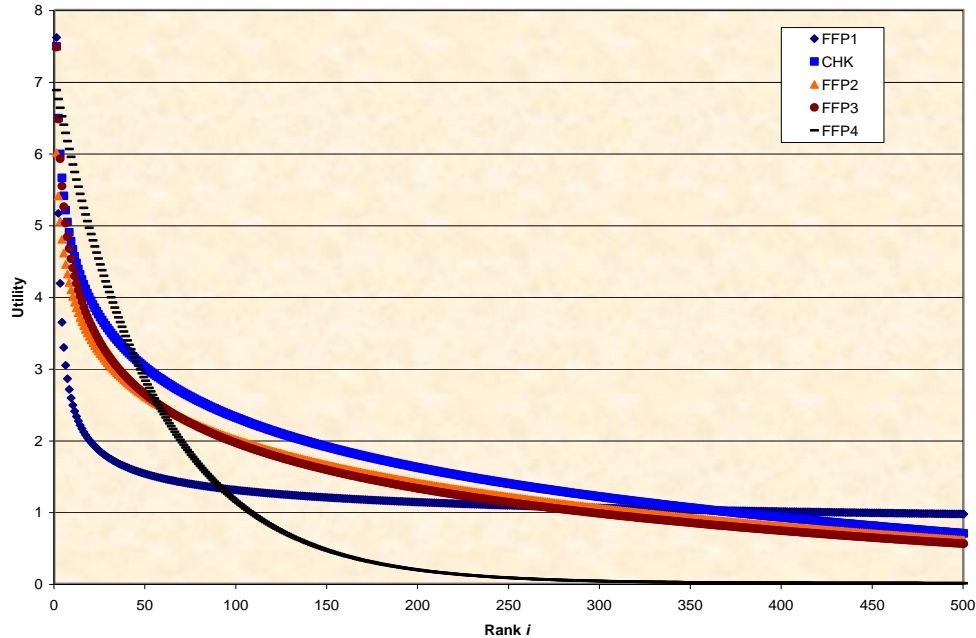


Figure 6: Comparison of five utility functions

Note that all five fitness functions depicted in Figure 6 satisfy the order preserving condition set in 1. However, only one of them — FFP1 — fails to improve the GP search performance result. From Figure 6, we can infer that one of the possible reasons is that the utility value drops off too fast — FFP1 has the steepest curve. It begins to level off at about the rank of 200, while all of the others level off at more than 350th rank. Dropping utility value too fast hurts the GP ranking discovery since some of the relevant documents may show up in the ranks between 250 and 350. Moreover, among the four that level off at more than 350th rank, it seems that the one with the faster dropping rate (steeper curve) comes along with better performance — FFP4 has the best performance, followed by FFP2 and FFP3, then by CHK. This indicates the importance

of choosing an appropriate utility function and its corresponding scaling parameters.

In general, the four best fitness functions have in common the following merits:

- a) All relevance information is considered, with the focus on relevant documents;
- b) The order of the relevant documents is taken into account. The utility assigned to the relevant documents decreases with their ranks. This is essentially the order preserving condition we set in Equation (1).
- c) The utility function follows the nonlinear descending curve that levels off at more than 350th¹ rank.

The above three points also can be treated as guidelines for fitness function design in genetic-based learning.

6 Conclusion

In this paper, we designed a set of order-based fitness functions based on utility theory. We tested this set of fitness functions along with other existing order-based fitness functions on the task of ranking function discovery for web search using genetic programming. Our experiments on a large web corpus show that several of our designs are very effective in guiding the GP search.

Among the various fitness functions we tested, FFP4, FFP2, FFP3, and CHK are the four we recommend. These four all follow the order preserving condition set in Equation

¹We use the top 1000 documents for each query in fitness calculation. This number will be proportionally smaller if less documents are used in fitness calculation.

(1) and work very well in the web search context. We hope the guidelines mentioned in the previous section can help future researchers design better fitness functions of their own in their GA/P applications.

References

- Chang, C.-H. (1999). *The design of an information system for hypertext retrieval and automatic discovery on WWW*. PhD thesis, Department of CSIE, National Taiwan University.
- Chen, H. (1995). Machine learning for information retrieval: neural networks, symbolic learning, and genetic algorithms. *Journal of the American Society for Information Science*, 46(3):194–216.
- Chen, H., Chung, Y., Ramsey, M., and Yang, C. (1998a). A smart itsy bitsy spider for the web. *Journal of the American Society for Information Science*, 49(7):604–618.
- Chen, H., Shankaranarayanan, G., She, L., and Lyer, A. (1998b). A machine learning approach to inductive query by examples: An experiment using relevance feedback, ID3, genetic algorithms, and simulated annealing. *Journal of the American Society for Information Science*, 49(8):693–705.
- Cordon, O., Moya, F., and Zarco, M. C. (2000). A GA-P algorithm to automatically formulate extended boolean queries for a fuzzy information retrieval system by means of ga-p techniques. *Mathware & Soft Computing*, 7(2-3):309–322.

- Cordon, O., Moya, F., and Zarco, M. C. (2002). A new evolutionary algorithm combining simulated annealing and genetic programming for relevance feedback in fuzzy information retrieval systems. *Soft Computing*, 6(5):308–319.
- Fan, W., Gordon, M. D., and Pathak, P. (1998). Automatic generation of matching functions by genetic programming for effective information retrieval. In *Proceedings of 1998 Americas Conference on Information Systems*, Milwaukee, USA.
- Fan, W., Gordon, M. D., and Pathak, P. (2000). Personalization of search engine services for effective retrieval and knowledge management. In *Proceedings of 2000 International Conference on Information Systems (ICIS)*, pages 20–34, Brisbane, Australia.
- Fan, W., Gordon, M. D., and Pathak, P. (2003a). Discovery of context-specific ranking functions for effective information retrieval using genetic programming. *IEEE Transactions on Knowledge and Data Engineering*. In press.
- Fan, W., Gordon, M. D., and Pathak, P. (2003b). A generic ranking function discovery framework by genetic programming for information retrieval. *Information Processing and Management*. In press.
- Fan, W., Gordon, M. D., Pathak, P., Wensi, X., and Fox, E. (2004). Ranking function optimization for effective web search by genetic programming: An empirical study. In *Proceedings of 37th Hawaii International Conference on System Sciences*, Hawaii. IEEE.
- Fishburn, P. C. (1988). *Non-Linear Preference and Utility Theory*. Johns Hopkins University Press, Baltimore.

- Gordon, M. (1988). Probabilistic and genetic algorithms for document retrieval. *Communications of ACM*, 31(2):152–169.
- Gordon, M. (1991). User-based document clustering by redescribing subject descriptions with a genetic algorithm. *Journal of the American Society for Information Science*, 42(5):311–322.
- Harman, D. K. (1996). Overview of the fourth text retrieval conference (TREC-4). In Harman, D. K., editor, *Proceedings of the Fourth Text Retrieval Conference*, pages 1–24. NIST Special Publication 500-236.
- Hawking, D. (2000). Overview of the TREC-9 web track. In Voorhees, E. M. and Harman, D. K., editors, *Proceedings of the Ninth Text Retrieval Conference*, pages 86–102. NIST Special Publication 500-249.
- Hawking, D. and Craswell, N. (2001). Overview of the TREC-2001 web track. In Voorhees, E. M. and Harman, D. K., editors, *Proceedings of the Tenth Text Retrieval Conference*, pages 61–67. NIST Special Publication 500-250.
- Holland, J. H. (1992). *Adaptation in Natural and Artificial Systems*. MIT Press, 2nd edition.
- Horng, J.-T. and Yeh, C.-C. (2000). Applying genetic algorithms to query optimization in document retrieval. *Information Processing and Management*, 36:737–759.
- Koza, J. R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA.

- Kraft, D. H., Petry, F. E., Buckles, B. P., and Sadasivan, T. (1994). The use of genetic programming to build queries for information retrieval. In *Proceedings of IEEE Symposium on Evolutionary Computation*.
- Kraft, D. H., Petry, F. E., Buckles, B. P., and Sadasivan, T. (1995). Applying genetic algorithms to information retrieval systems via relevance feedback. In *Fuzzy Sets and Possibility Theory in Database Management Systems*, pages 330–346.
- Kraft, D. H., Petry, F. E., Buckles, B. P., and Sadasivan, T. (1997). Genetic algorithms for query optimization in information retrieval: relevance feedback. In *Genetic Algorithms and Fuzzy Logic Systems: Soft Computing Perspectives*, pages 155–173.
- Kwok, K. L. (1997). Comparing representations in Chinese information retrieval. In *Proceedings of the 1997 ACM SIGIR*, pages 34–41.
- Langdon, W. B. (1998). *Data Structures and Genetic Programming: Genetic Programming + Data Structures = Automatic Programming*. Kluwer Publishing.
- Lopez-Pujalte, C., Bote, V. P. G., and de Moya Anegon, F. (2002). A test of genetic algorithms in relevance feedback. *Information Processing and Management*, 38:793–805.
- Lopez-Pujalte, C., Bote, V. P. G., and de Moya Anegon, F. (2003). Order-based fitness functions for genetic algorithms applied to relevance feedback. *Journal of the American Society for Information Science and Technology*, 54(2):152–160.
- Martin-Bautista, M. J., Vila, M., and Larsen, H. L. (1999). A fuzzy genetic algorithm

- approach to an adaptive information retrieval agent. *Journal of the American Society for Information Science*, 50(9):760–771.
- Mitchell, T. M. (1997). *Machine Learning*. McGraw Hill.
- Pathak, P., Gordon, M., and Fan, W. (2000). Effective information retrieval using genetic algorithms based matching function adaptation. In *Proceedings of the 33rd Hawaii International Conference on System Science (HICSS)*, Hawaii, USA.
- Robertson, A. M. and Willett, P. (1994). Generation of equiprequent groups of words using a genetic algorithm. *Journal of Documentation*, 50(3):213–232.
- Robertson, A. M. and Willett, P. (1996). An upperbound to the performance of rank-output searching: optimal weighting of query terms using a genetic algorithm. *Journal of Documentation*, 52(4):405–420.
- Robertson, S. E., Walker, S., Jones, S., Hancock-Beaulieu, M. M., and Gatford, M. (1996). Okapi at TREC-4. In Harman, D. K., editor, *Proceedings of the Fourth Text Retrieval Conference*, pages 73–97. NIST Special Publication 500-236.
- Salton, G. (1989). *Automatic Text Processing*. Addison-Wesley Publishing Co., Reading, MA.
- Salton, G. and Buckley, C. (1988). Term weighting approaches in automatic text retrieval. *Information Processing and Management*, 24(5):513–523.
- Smith, M. P. and Smith, M. (1997). The use of genetic programming to build boolean

- queries for text retrieval through relevance feedback. *Journal of Information Science*, 23(6):423–431.
- Vrajitoru, V. (1998). Crossover improvement for the genetic algorithm in information retrieval. *Information Processing and Management*, 34(4):405–415.
- Yang, J. J. and Korfhage, R. R. (1993a). Effects of query term weights modification in document retrieval: a study based on a genetic algorithm. In *Proceedings of the Second Annual Symposium on Document Analysis and Information retrieval*, pages 271–285, Las Vegas, NV.
- Yang, J. J. and Korfhage, R. R. (1993b). Query optimization in information retrieval using genetic algorithms. In *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 603–611.
- Yang, J. J., Korfhage, R. R., and Rasmussen, E. (1993). Query improvement in information retrieval using genetic algorithms: a report on the experiments of the TREC project. In *Proceedings of the First Text Retrieval Conference*.
- Zobel, J. and Moffat, A. (1998). Exploring the similarity space. *SIGIR Forum*, 32(1):18–34.