Information Retrieval, 8, 359–381, 2005

Learning to Rank

ANDREW TROTMAN andrew@cs.otago.ac.nz
Department of Computer Science, University of Otago, Dunedin, New Zealand

Abstract. New general purpose ranking functions are discovered using genetic programming. The TREC WSJ collection was chosen as a training set. A baseline comparison function was chosen as the best of inner product, probability, cosine, and Okapi BM25. An elitist genetic algorithm with a population size 100 was run 13 times for 100 generations and the best performing algorithms chosen from these. The best learned functions, when evaluated against the best baseline function (BM25), demonstrate some significant performance differences, with improvements in mean average precision as high as 32% observed on one TREC collection not used in training. In no test is BM25 shown to significantly outperform the best learned function.

Keywords: searching, document ranking, genetic programming, machine learning

1. Introduction

Information retrieval searching is typically a two-step process, first potentially relevant documents are identified, and then the found documents are ranked.

The identification process is often conducted as set intersection—from the set of all documents, the potentially relevant documents are those that contain all or some of the search terms (often constrained by a Boolean expression, sometimes metadata is additionally indexed).

Ranking involves combining a set of heuristics derived from the corpus, the result set, and individual documents; typical heuristics include tf and IDF. The similarity of each document to the query is computed and documents are sorted according to this. Ranking has been the focus of much attention for many years with several "standard" functions emerging including inner product (Witten et al. 1994), cosine (Harman 1992), probability (Robertson and Sparck Jones, 1976) and BM25 (Robertson et al. 1995).

The performance of a retrieval system is of greater importance than the mathematics behind ranking. A user is unlikely to ask "upon which theoretical model is this IR system built?." Instead, their criterion of goodness is how well the system appears to perform to them. In response the IR community has developed test collections and metrics to measure ranking performance. Such collections include TREC (Harman 1993), cystic fibrosis (Shaw et al. 1991) and others.

Examining ranking from a machine learning perspective offers new opportunities. A TREC collection can be viewed as an artificial intelligence training set. The IR system can be viewed as an evaluation function. Precision can be viewed as fitness. The task of machine learning is to choose a function that maximizes fitness over the training set.

In this investigation new general purpose ranking functions are evolved using genetic programming and the TREC WSJ collection. Performance is measured with mean average precision during and after training. When the best evolved ranking functions are tested on TREC collections not used in training significant improvements of as high as 32% are seen when compared to BM25. On no test collection is BM25 shown to significantly outperform the learned functions.

2. Background

2.1. Precision

There are numerous "standard" measures of precision. `Trec_eval` (Buckley 1991), the program used to measure precision for TREC, outputs over 20 different statistics. New metrics have recently been introduced for graded relevance judgments (Kekäläinen and Järvelin 2002) such as those in the cystic fibrosis collection.

Precision-at-document-*n* is considered a good measure for the web (Anh and Moffat 2002), as a user will typically examine only the first page of *n* results. Buckley and Voorhees (2000) counter argue precision-at-document-*n* is unstable and demonstrate the stability of mean average precision (MAP) for general purpose retrieval. Either could be used for evolving a ranking function. As this investigation is not examining web documents, or graded relevance judgments, MAP is considered the most appropriate measure.

For purposes herein, MAP is computed for a set of queries by taking the mean of the average precisions of each query in the set. Average precision is computed as the sum of precisions for each found and relevant document, divided by the number of relevant documents. Using this construction, relevant but not found documents receive a precision of zero.

2.2. Significance

Comparing MAP scores suggests little about the nature of the improvements. One vastly improved query can easily compensate for many worsened queries. Such a change is only an improvement when measured as MAP. If measured as the ratio of improved queries to all queries (ROI), the nature of the improvement is more clear.

Small random fluctuations between queries may appear large if measured with ROI alone, so the significance must also be computed. The average precision of each query for each ranking algorithm is computed and significance computed with a one-tailed t-test. This gives the probability, P, the difference between the two result sets is purely chance.

Once MAP, ROI, and P are known the difference between two ranking algorithms can be expressed as the likelihood any new query will be improved (ROI), by how much (MAP), and the certainty of this statement (1-P). There, does, however, remain a chance of type II errors.

2.3. *Retrieval methodology*

First the set of potentially relevant documents is identified as any document containing any search term, then a document/query similarity is computed for those documents. As ranking does not alter the choice of documents, recall cannot be affected by function choice (unless relevance cut-offs are applied). Only the order of the documents is affected by ranking.

The inverted lists present in an inverted file information retrieval system identify which documents contain which terms. This is the ideal environment in which to test ranking. Zobel et al. (1998) have already demonstrated the performance superiority of inverted files over other retrieval methods.

Kaszkiel and Zobel (1998) show a further performance advantage to using term-oriented searching over document-oriented searching. In term-oriented searching, each term is examined one at a time. The postings for a term are fetched, processed, and discarded before moving on to the next. By contrast, for document-oriented searching all terms are evaluated for one document before moving on to the next. Document-oriented searching requires all terms to be held in memory concurrently (which is prone to paging). Alternatively partial postings can be loaded, processed, and discarded before loading further chunks (taking multiple disk operations per term). As random access disk I/O is the bottleneck in IR (Williams and Zobel 1999, Zobel and Moffat 1995), document-oriented searching is prone to slow execution however it is attempted.

Term-orientation also has disadvantages. The document weight must be computed for all documents concurrently and piecewise, term by term. Temporary results are usually stored in a set of accumulators, one for each document (Zobel et al. 1998). Not all published ranking functions can be calculated in this way. Cover density ranking (Clarke et al. 2000), for example, relies on term-proximity and must be computed in a document-oriented manner.

For this investigation, the performance advantage of term-oriented inverted list retrieval is considered to outweigh any ranking disadvantage even though this limits the set of possible learnable functions.

2.4. *Genetic programming*

Holland (1975) formulated the genetic algorithm (GA) learning process: First a population of individuals is chosen completely at random. Each is then evaluated against a fitness function. A subsequent population (the next generation) is created through reproduction, crossover, and mutation, the process is repeated iteratively until a problem solution is found.

Koza (1992) introduced genetic programming (GP). By contrast to genetic algorithms, which learn a series of numbers, genetic programming learns arithmetic expressions. Koza's suggestion was to use an abstract syntax tree as an individual.

To learn an IR ranking function using genetic programming, it must be possible to represent the function as an individual (a tree) and to define fitness, reproduction, crossover, and mutation.

Ranking functions are often expressed in the form

$$w_{dq} = \sum_{t \in q} g(t, d) \tag{1}$$

where the weight, w_{dq}, of a document, d, with respect to a query, q, is the sum of influences, $g()$, of each term, t, with respect to document d. Calculated term-wise, partial results are stored in accumulators. Each $g(t, d)$ is deterministic, independent of term ordering, and can be represented as an abstract syntax tree for GP learning.

The stability and ubiquity of mean average precision makes it a good measure of fitness. MAP, $f()$, of individual n is defined as

$$f(n) = \frac{\sum_{q \in Q} p_{nq}}{|Q|} \tag{2}$$

where Q is the set of queries, and p_{nq} is the average precision of individual n with respect to query q, and $|Q|$ is the number of queries in the set.

Individual fitness says nothing about the strength of an individual with respect to the current generation, that is the individual's fitness-proportion, *fp*(n).

$$fp(n) = \frac{f(n) - F + \varepsilon}{\sum_{m=1}^{G} (f(m) - F + \varepsilon)} \tag{3}$$

where G is the number of individuals in the generation, F is the minimum observed $f()$in the generation, and ε is included to prevent division by zero. This linear dynamic scaled fitness proportion (Grefenstette 1986) is preferred over its unscaled counterpart as it is general purpose (Khuri et al. 1994).

The sum of fitness-proportions for a single generation is 1. A generation can therefore be represented as a number line in the range [0... 1]. Each individual takes a part of the line equal to its fitness-proportion. In the process of fitness-proportionate selection, a random number between 0 and 1 is chosen and the individual at that point on the line is selected.

In reproduction, an individual is chosen from the current generation using fitness-proportionate selection and then carried over into the next generation.

For mutation, an individual is chosen using fitness-proportionate selection, and a random node of the tree is chosen. That node is then replaced with a new and random node. The process is represented in figure 1. Should the arity of the node decrease, the branches are pruned right then left. Should the arity increase, new leaves are created left to right through random selection of atomic operands. The new individual is carried over into the next generation.

Using fitness-proportionate selection, two individuals (A and B) are chosen for crossover. A node, a, is randomly chosen from A. Likewise a node, b, is chosen from B. Two new

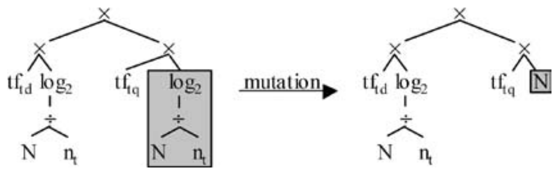

Figure 1. Mutation of a $\log_2$ node into an atom requires the deletion of all nodes below the mutation point.

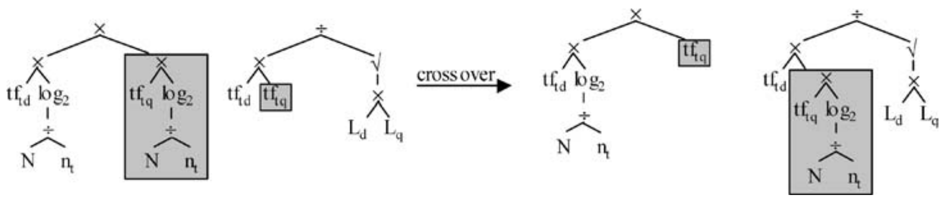

Figure 2. Crossover of two individuals results in two new individuals.

individuals are then created. The first by replacing node a with b in A, the second by replacing b with a in B. Figure 2 represents this process. Both new individuals are carried over into the new generation.

Reproduction, mutation, and crossover occur with configurable probabilities (which sum to 1). Once a new generation has been created, the genetic process is repeated iteratively until a problem solution is found.

3. Ranking functions

There are many theoretical models for information retrieval. Salton et al. (1975) suggested the vector space model, Robertson and Sparck Jones (1976) suggested the probability model, there are also the Boolean model, 2-Poisson model (Raghavan et al. 1983), set model (Pôssas et al. 2002), and other models.

Estimates of the number of published ranking functions range from counting in tens to counting in thousands. Salton and Buckley (1988) suggest a taxonomy allowing for 729 ranking functions. Zobel and Moffat (1998) expand on this and suggest a naming scheme allowing 1,500,000 functions, of which about 100,000 are unique. Whatever the choice of model or ranking function, it is the implementation that is measured, not the validity of the model.

Most ranking functions are required to quickly compute document weight. The TREC web track VLC2 collection contains over 18.5 million documents (Hawking et al. 1999). In the worst case, a weight must be computed for every document. To scale, the ranking function must be a simple combination of evidence already "freely" available at search time, not information that is costly to compute. Such evidence can be generated during indexing (such as term frequency), but cannot rely on document analysis during search.

Choice of which evidence to use dictates the best performance of a ranking function. If, for example, no tf_{td}-derived metrics are present, ranking cannot be based on the number of times a term occurs in a document—and is likely to perform badly when compared to functions that use this metric. For this investigation, inner product, cosine, probability, and Okapi BM25 are used for baseline comparison. To expect to do at least as well as each of these, it must be possible to learn all of these (there is no *a priori* way to know which is best). Analysis of each baseline function results in a list of necessary atoms.

3.1. Available atomic evidence

The query, the document collection, each search term, and each document carry atomic information, namely:
The query

- The length of the query (number of terms including repetition): T_q
- The length of the query vector (squared): L_q
- The query unique term count: u_q
- The term frequency in the query of the term occurring most frequently in the query: m_q

Each search term

- The document frequency: n_t
- The term frequency in the corpus: n_c
- The term frequency in the given document: tf_{td}
- The term frequency in the query: tf_{tq}

Each document

- The length of the document (number of terms including repetition): T_d
- The length of the document vector (squared): L_d
- The document unique term count: u_d
- The term frequency in the document of the term occurring most frequently in the document: m_d

The document collection

- The number documents: N
- The sum of document lengths (in terms): T
- The longest document's length (in terms): $T_{\max}$
- The number of unique terms in the corpus: U
- The largest document unique term count: $U_{\max}$
- The largest corpus term frequency: M
- The largest document frequency: $M_{\max}$
- The largest term frequency: $tf_{\max}$
- The length of the longest document vector (squared): $L_{\max}$

Computation of vector lengths

The squares of vector lengths are computed as they are guaranteed to be integers and are easier to store in indexes than floating point numbers (query vector length is squared in keeping). The vector-squares are computed by taking the sum, for each unique term, of the square of the number of occurrences of that term.

3.2. Combination of evidence

The above atoms can be combined using arithmetic operators, combined with constants and with the current accumulator value.

Constants

- Constants: $\Re_1, \Re_2, \Re_3, \ldots$ (in the range [0..100])
- The current document accumulator value (initially 0): $\mathcal{A}_{d,t}$

Arithmetic operators

- $+$
- $-$
- $\times$
- $\div$

Function operators

- Log of the absolute value of x ($\log|x|$)
- Log to base 2 of the absolute value of x ($\log_2|x|$)
- Minimum of x and y ($\min(x, y)$)
- Maximum of x and y ($\max(x, y)$)
- Square root of the absolute value of x ($\sqrt{|x|}$)

Operators

If the number of atoms significantly outweighs the number of operators, randomly produced individuals at the start of learning are more likely to be atomic than functional. To alleviate this, the probability of choosing an operator was three times that of choosing an atom or constant.

3.3. Taxonomy

Although the above operators and atoms generate an infinite number of ranking functions, not all published functions can be described. PageRank (Page et al. 1998) requires information about citation counts, which can change each time a new document is added to the collection. Cover density ranking (Clarke et al. 2000) requires term proximity data, which changes depending on choice of search terms. HITS (Kleinberg 1999) performs post-search document analysis. Nonetheless, inner product, cosine, probability, Okapi BM25, and Boolean can be described using the notation above.

3.3.1. Inner product. Consider the document and query to be vectors. The inner product (Witten et al. 1994) computes the projection of one vector on the other—in other words,

what proportion of one vector the other describes. The inner product of the document and query vector describes what proportion of the query is described by the document.
Document weight w_{dq} with respect to query, q, is

$$w_{dq} = \sum_{t \in q} (w_{dt} \times w_{qt}) \tag{4}$$

where w_{dt} is the (modified) document vector with respect to the given term,

$$w_{dt} = tf_{td} \times \log_2 \frac{N}{n_t} \tag{5}$$

and w_{qt} is

$$w_{qt} = tf_{tq} \times \log_2 \frac{N}{n_t} \tag{6}$$

3.3.2. Cosine measure. Considering the document and the query to be vectors, the cosine measure computes the cosine of the angle between those vectors (Harman 1992). A large cosine is given for a small angle. The smaller the angle, the closer the document is to the query.

Document weight w_{dq} with respect to query q is given by

$$w_{dq} = \frac{\sum_{t \in q} (tf_{td} \times tf_{tq})}{\sqrt{L_d \times L_q}} \tag{7}$$

the equivalent sum for each term is

$$w_{dq} = \sum_{t \in q} \frac{tf_{td} \times tf_{tq}}{\sqrt{L_d \times L_q}} \tag{8}$$

3.3.3. Probability measure. The probability model computes the probability that a given document is related to a given query (Robertson and Sparck Jones 1976). This cannot be calculated directly, however Harman (1992) gives an approximation:

$$w_{dq} = \sum_{t \in q} ((C + IDF_t) \times f_{td}) \tag{9}$$

where

$$IDF_t = \log_2 \frac{N - n_t + 1}{n_t} \tag{10}$$

and

$$f_{td} = K + (1 - K) \times \frac{tf_{td}}{m_d} \tag{11}$$

where $C = \Re_1 = 1.0$ and $K = \Re_2 = 0.3$.

3.3.4. Okapi BM25. Originally found during ranking experiments for TREC, Okapi BM25 (Robertson et al. 1995) has proven to be a successful general purpose ranking function. It computes a weight between document and query based on term probability. The higher the weight the stronger the relationship between the query and the document.

$$w_{dq} = \left(\sum_{t \in q} w_t \times \frac{(k_1 + 1) \times tf_{td}}{K + tf_{td}} \times \frac{(k_3 + 1) \times tf_{tq}}{k_3 + tf_{tq}} \right) \tag{12}$$

where

$$w_t = \log_2 \frac{N - n_t + 0.5}{n_t + 0.5} \tag{13}$$

and

$$K = k_1 \times \left((1 - b) + \frac{b \times T_d}{T_{\text{av}}} \right) \tag{14}$$

and

$$T_{\text{av}} = \frac{T}{N} \tag{15}$$

and $k_1 = \Re_1 = 1.2$, $k_3 = \Re_2 = 7$, and $b = \Re_3 = 0.75$ (so called BM25(1.2, 0, 7, 0.75)).

Many different general purpose constants have been proposed for k_1, k_3 and b (they are collection specific). Robertson et al. originally proposed $k_1 = 2, k_3 = \infty b = 0.75$, however other suggested general purpose values include $k_1 = 1.2$, $k_3 = 7$ or 1000, and $b = 0.75$ (Robertson and Walker 1999). Each of these three constant sets was evaluated on the training set and the best used.

3.3.5. Other possible ranking functions. Thus far, each ranking function is described as the sum of something for each term. Other forms could be used. The accumulator is an atom and can be included as an integral part of the ranking function. Such a ranking function might apply greater influence to terms at the beginning of a query than those at the end. Expressed recurrently,

$$\begin{aligned} \mathcal{A}_{d,0} &= 0 \\ \mathcal{A}_{d,t} &= \mathcal{A}_{d,t-1}^2 + \left(tf_{td} \times \log_2 \frac{N}{n_t} \right) \end{aligned} \tag{16}$$

where $\mathcal{A}_{d,t}$ is the value of the accumulator after the influence of term t from query q $(t \in q)$ has been computed. The final document weight w_{qd} is $\mathcal{A}_{d,q}$, the value of the accumulator after all terms have been considered.

3.3.6. Boolean ranking. Boolean ranking can be expressed recurrently in the same way

$$\begin{aligned} \mathcal{A}_{d,0} &= 0 \\ \mathcal{A}_{d,t} &= \mathcal{A}_{d,t-1} + (1 - \mathcal{A}_{d,t-1}) \end{aligned} \tag{17}$$

resulting in a weight of 1 for all documents containing at least one term and 0 for all other documents (additionally a list of documents satisfying the Boolean expression is needed; this is available in the Boolean-ranking hybrid search engine used). In this way, the number of possible derivable ranking functions is infinite and not limited to simple sums over each term.

4. Related work

4.1. GPs For unstructured ranking

Examining the cystic fibrosis collection (Shaw et al. 1991), Oren (2002a, 2002b) ran experiments to determine if it was possible to learn new *tf.idf* like ranking functions. Splitting the 100 queries, 70 for training, 30 for evaluation, a between 5 and 8% improvement on *tf.idf* was seen during training on the training queries. Relative to *tf.idf* ranking, an at best 4.4% improvement in mean 11 point average prevision was demonstrated on the evaluation queries. Considering this a negative result, the experiment was reported with a negative conclusion.

The Oren (2002a) function most adapted to the training set is not tested here as it includes the problematic sub-expression

$$\log_2\left(n_t + \frac{\log_2(n_t) \times (M + N + (tf_{td}/m_d))}{N - 0.366 \times T_d + n_t}\right) \tag{18}$$

the top line of this expression is always greater than zero; the bottom line is not. In the cystic fibrosis collection, $N = 1{,}239$ and $M = 16{,}640$, and tf_{td}/m_d is always less than 1 (so it will be discarded). This gives

$$\log_2\left(n_t + \frac{\log_2(n_t) \times (16640 + 1239)}{1239 - 0.366 \times T_d + n_t}\right) \tag{19}$$

an expression relating document length to the number of occurrences of a given term. Examining the case where $n_t = 1$, a term that occurs only once in the collection gives

$$\log_2(n_t) = 0 \tag{20}$$

a document can be any length. However, when the terms occurs in only two documents ($n_t = 2$) the expression becomes

$$\log_2\left(2 + \frac{17879}{1241 - 0.366 \times T_d}\right) \tag{21}$$

values for T_d can now be found such that this expression is undefined. For the case of $n_t = 2$, the expression is undefined when T_d is in the range [3,390. . . 27,815]. The longest document in the cystic fibrosis collection is 5,295 terms long; it is inside the undefined range. Should a term occur in the collection exactly 2 times and also occur in the longest document, an overflow would occur when searching for that term. Oren must not have encountered this situation during his experiments (perhaps due to stemming). As this equation can only rank some documents, it was not tested in this investigation.

Oren reported other less successful functions also not examined herein (they were less successful). One function, evolved using all documents and all queries from the cystic fibrosis collection, showed a 4.96% improvement on *tf.idf* when tested on the CISI collection. This function was reported negatively as a less than 5% improvement was seen.

Using the TREC AP collection, Fan et al. (2004) applied genetic programming to a set of *tf.idf* like atoms and 4 operators ($+$, $\times$, $\div$, log); without seeding with good functions. Using queries constructed from the title and description fields, a 3.13% MAP improvement is seen on BM25. Using long queries constructed from title, narrative, description and concepts, a 5.71% improvement is seen. When their new function was tested on the 10GB TREC web track collection, they demonstrated a 3.4% improvement on BM25. Fan et al. use a set of atoms similar to that of Oren, and a subset of Oren's chosen operators, consequently their result is similar to that of Oren, a near 5% improvement is seen at best.

4.2. GPs for structured and semi-structured ranking

Fan et al. (1999) suggest using genetic programming to learn ranking functions but report no results. Fan et al. (2004) report results from learning experiments on semi-structured documents in HTML. Atomic ranking evidence is collected for each of entire-document, title, abstract, body, and anchor tags. These are combined using $+$, $\times$, $\div$, and log. They report an almost 20% improvement on BM25 when using document structures, but a 2% degradation when not using document structures in ranking. Comparison is against BM25 as that is the best performing baseline function they try.

The evidence and operators available for genetic learning dictate the maximum performance of any learned function. Should tf_{tq} not be available for learning, it is unreasonable to expect to learn a function significantly outperforming a handcrafted function using it. Equally, if document structure heuristics are unavailable to the handcrafter, it is unreasonable to assume any handcrafted function will exceed the performance of a learned function using these.

Neither subtraction nor tf_{tq} are available for combination in the experiments of Fan *et al.*, consequently, their result is unsurprising—only by using document structures are improvements seen on known good functions.

The investigation herein and the results presented later demonstrate this point. By ensuring each of the baseline functions can be learned it is possible to guarantee performance at worst equal to the best performing baseline function (as learning can be seeded with the baseline functions).

5. Methods

Experiments were conducted to determine if it is possible to learn a ranking function that significantly outperforms the current popular functions.

5.1. Training and test set

The TREC Wall Street Journal collection (1987–1992) from TREC disks 1 and 2 was indexed using a term-oriented inverted file information retrieval system and used for training.

A set of queries were built from topics 1–200 by taking the title, description, and narrative fields, and stopping commonly occurring words. These queries were used to compare the performance of each of inner product, cosine, probability and the BM25 variants. The best performing of these was then compared to known good results published by others to lend validity to the implementation. These queries were then discarded.

A second set of queries was built in the same manner, but using just the description field (training on short queries was preferred for efficiency reasons). Performance of each function against this set was then computed. Again, the results were compared to known good results.

Queries for topics 1–100 were then discarded (for efficiency reasons). Additionally, topics 121, 175, 178 and 181 were discarded, each having fewer than 5 positive judgments against the Wall Street Journal collection. The remaining queries were divided into two groups; topics 151–200 were used for training, topics 101–150 for evaluation.

Ranking performance is known to vary greatly from collection to collection (Zobel and Moffat 1998) so any test of ranking function performance must be on a diverse set of test collections. The cystic fibrosis collection and the TREC collections on disks 4 and 5 were used.

The cystic fibrosis collection is a 1239 document subset of Medline on the topic of cystic fibrosis. The collection has 100 queries, each with complete graded judgments from each of four human judges. The judgments were converted into binary decisions by considering any positive judgment to be positive, and only complete negative judgments to be negative. Each query was built by stopping commonly used words. All queries with fewer than 5 positive judgments were discarded.

The TREC collections from disks 4 and 5 were indexed individually (FT, CR, FR94, FBIS, LATIMES), collectively for each disk (TREC 4, TREC 5) and collectively (TREC 4 + 5). Topics 301–350 were converted into queries by taking the narrative and stopping common words. Topics with fewer that 5 positive judgments on a given collection were discarded when searching that collection.

5.2. Genetic parameters

An initial population of 100 individuals was created from 96 random individuals and 4 prior ranking functions.

To create a random individual, first a list of objects was constructed consisting of each atom and constant once and each arithmetic and function operator three times (see Sections 3.1 and 3.2). From this list an object was chosen at random. If the object was atomic, the individual was functionally complete so added to the population. If not, it was an operator so the process was repeated recursively for each operand of the operator until the individual was functionally complete. If, at any point, the depth of recursion exceeded 5, the list was restricted to prevent operators from being selected.

To guarantee the worst possible performance during training was at least as good as the best of the baseline functions, each of inner product, cosine, probability, and BM25 (Eqs. (4), (7), (9) and (12) respectively) was added to the random individuals to form an initial population of 100.

Purely by chance, the fittest individual in a generation might not be selected to carry over into the next generation. Should this happen, the fittest-so-far would perish. To prevent this, the fittest individual in each generation is always selected to reproduce; reproduction was elitist (De Jong 1975).

The mutation rate was set to 0.05, crossover to 0.9; and reproduction to 0.05. Mean average precision was used as fitness.

5.3. Experimental process

Mean average precision was calculated and recorded for the training and evaluation sets using inner product, cosine, probability and BM25.

The initial generation was randomly chosen and seeded with the baseline functions. Each individual's mean average precision for the training set was computed and recorded. For the fittest individual in each generation, mean average precision for the evaluation set was computed and recorded. The fittest individual in the generation was then recorded. A new generation was created through reproduction, mutation, and crossover, and the process repeated iteratively.

The experiment was run concurrently on 13 different computers starting from different random number seeds (taken from the system clock) on the same random number generator. Learning terminated after 100 generations. More work is required to determine good conditions for concluding experimental runs.

Any functions showing a greater than 10% improvement in training set MAP were considered good. For each good function, average 11-point precision was graphed against recall.

Each query in the evaluation set was then individually examined. Average precision for each good function was computed and recorded. ROI was computed and the t-test was then conducted on these scores and those of the best baseline function (BM25).

Each good function was finally evaluated against inner product, cosine, probability and BM25 using the cystic fibrosis collection and against BM25 for the TREC collections.

5.4. *Cross validation*

Normally, in supervised learning, the computer is learning a mapping from a set of known samples to a set of known results. Given x and y, the task is to learn f such that $f{:}x \rightarrow y$. In ranking experiments y (the "correct" average precision) cannot be known in advance. Consequently cross validation is not possible.

A cross validation estimate comes through an estimation of best possible performance. Genetic programming is assumed to return a reasonable approximation to the optimal ranking function. A function is then learned for only the evaluation set (population 100, 100 generations, 13 runs). Performance of this cross validation function and the good functions is then compared.

The most probable use of this research is offline learning then online evaluation by users. This is comparable to learning on one set of TREC queries and evaluation on another. Equally, any learned functions would be used across different datasets. Mirroring this behavior, the good functions are evaluated using the cystic fibrosis collection and TREC collections.

5.5. *Numeric overflow*

Oren's function was not tested as overflow could occur. In general, overflow must be detected and addressed. Any individual causing overflow is not a valid candidate for general-purpose ranking.

To reduce the chance of overflow, absolute values of operands were taken before log or square root operations but there remained the possibility of division by zero, or log(0), due to subtraction (e.g. log(abs(N-N))).

To catch all cases of overflow, operating system routines (such as *finite*( )) were called each time an individual was evaluated against a term/document pair. If overflow was detected, the fitness of the given individual was set to zero (the individual perished) and computation moved on to the next individual. Although this does not guarantee overflow or underflow will not occur, it does guarantee that it did not. Learned functions are therefore unlikely to overflow during use.

6. Results

Table 1 presents the mean average precision for each of the baseline functions. BM25(1.2, 0, 7, 0.75) outperforms the other tested functions when just the narrative is used to construct queries. When additionally the title and description are used, only BM25(1.2, 0, 1000, 0.75) betters it. Because experiments are against the narrative only, BM25(1.2, 0, 7, 0.75) is chosen as the comparison baseline (and reported hereafter as BM25).

Others have reported results for BM25(2, 0, ∞, 0.75), so the implementation of this lower performing variant was compared to that of others.

Robertson et al. (1994) examined performance of BM25 against TREC disks 1 and 2 using topics 151–200 using title, narrative, and description. They report a mean average

Table 1. Mean average precision evaluated for TREC WSJ collection using topics 1-200 and queries constructed from just the description field or the title, narrative and description. 1,000,000 was used as an approximation of ∞.

Equation	⟨narr⟩	⟨title⟩⟨narr⟩⟨desc⟩
BM25(1.2,0,7,0.75)	0.2224	0.3256
BM25(1.2,0,1000,0.75)	0.2218	0.3287
BM25(2,0,∞,0.75)	0.2171	0.3197
Probability	0.1585	0.1807
Inner Product	0.1523	0.2083
Cosine	0.0538	0.0944

precision of 0.337. The value computed herein (0.3197) is considered inline with their reported results, even though lower–a document subset and query superset were used.

Savoy et al.(1995) report results for BM25 on the same TREC WSJ collection used herein, with TREC topics 1–200, but report using mean 11 point average precision. Using just the narrative they report 0.2256, additionally using title and description they report 0.3249. The values computed herein (0.2171 and 0.3197 respectively) are also considered inline with this. Differences are expected for two reasons: the exact algorithms they use for constructing the query from the topics is not given (including stop word list), and the measures are subtly different.

Figure 3 shows how mean average precision in the training and evaluation sets changed over time. Plotted is the mean across all 13 runs. From this, it appears as though 100 generations is not enough to learn to completion (the gradient at generation 100 is not 0). This result is surprising as Fan et al. (2004) run their experiments for only 30 generations. Oren ran for up to 150 generations.

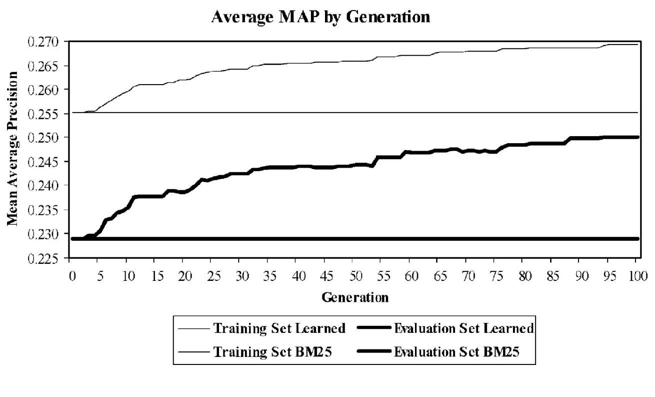

Figure 3. Mean (for 13 runs) of training set MAP at each generation during training.

Table 2. Final mean average precision scores for each run and BM25.

	Training set		Evaluation set	
Run	MAP	Improv.(%)	MAP	Improv.(%)
1	0.26142	2.41	0.24054	5.11
2	0.26814	5.05	0.25054	9.48
3	0.27078	6.08	0.25733	12.44
4	0.26992	5.75	0.24833	8.51
5	0.28493	11.63	0.26319	15.00
6	0.27166	6.43	0.26013	13.67
7	0.26552	4.02	0.24142	5.49
8	0.26469	3.70	0.23949	4.65
9	0.26018	1.93	0.23415	2.31
10	0.26750	4.80	0.25555	11.67
11	0.26361	3.27	0.23708	3.59
12	0.26818	5.07	0.24669	7.79
13	0.28377	11.17	0.27567	20.46
Mean	0.26925	5.48	0.25001	9.24
BM25	0.25526	0	0.22885	0
Cross Valid.			0.26733	16.81

Table 2 presents mean average precision calculated for the fittest individual in the final generation of each run–which, due to elitism, is the fittest individual in any generation. All of the runs show an improvement on BM25. Two of the runs show an improvement of over 10% in the training set and are considered good.

The best good function, run 5 and presented as Eq. (22), exceeded BM25 on the evaluation set by 15%. It is in three parts; first a variant on log odds, second a function of tf_{td} and the document length T_d, third a function of tf_{tq}. This equation combines evidence from the document collection, the document, the search term, and from the query–all the categories identified above as containing evidence suitable for combination. Similar combinations are common in many ranking functions including BM25.

$$\begin{aligned} w_{dq} = \sum_{t \in q} \Bigg(& \log_2 \left| \frac{N - \log_2 |N|}{n_t + n_t} \right| \\ & \times \frac{n_c \times tf_{td}}{\max\left(\Re_3, \Re_4 + \frac{\Re_1 \times (\log|\Re_2 + tf_{tq}| + n_c) \times T_d}{T}\right) + tf_{td}} \times \frac{M \times tf_{tq}}{n_t} \Bigg) \end{aligned} \tag{22}$$

where $\Re_1 = 33.40102$, $\Re_2 = 23.94623$, $\Re_3 = 1.2$, $\Re_4 = 0.25$, and all logs are of absolute values.

Functions similar to already good BM25 were expected to evolve. According to Igel and Chellapilla (1999) the closer an operator is to the root of the individual, the more destructive a genetic change at that location. Changes away from a three-part equation are highly destructive and unlikely to survive. As learning was seeded with the already good BM25, it is reasonable to assume many learned functions will maintain the same general shape. It is not clear what might happen if the initial population is not seeded.

The function most adapted to the evaluation set exceeded BM25 by 20% and evolved in run 13. Fan et al. (2004) show a performance degradation of 2% compared to BM25 on HTML documents and only exceed BM25 (by nearly 20%) when using document structure. Fan et al. (2004) show only small improvements on BM25 of between 3.5 and 5.7% when training on the TREC AP collection. The results presented herein suggest performance gains as large as those of Fan et al. can be gained without using document structure.

Presented as Eq. (23), run 13 is dissimilar to both BM25 and run 5, and includes use of $\mathcal{A}_{d,t}$, the current accumulator value (terms were processed in alphabetical order; as a bag of words).

In both run 5 and run 13 there has been no attempt to optimize the value of $\Re_n$, to do so would be to present a function that was not learned. The interaction of of $\Re_1$, $\Re_2$, $\Re_4$ with $\Re_3$ in run 5, and the interaction of $\Re_1$ with $\Re_4$ in run 13 require investigation.

$$w_{dq} = \sum_{t \in q} \left(\Re_2 \times \sqrt{\left| \frac{\log \left| \frac{\max(L_d, m_d)}{L_{\max} - \frac{(\max(\min(\log_2 |A_{d,t}|, L_d), L_q) + T_{\max}) \times T_q}{n_c + \Re_3}} \right| \times \log_2 \left| \frac{n_c}{\min(N, n_t)} \right| \times tf_{td}}{(n_c + \Re_3) \times \left(\Re_3 \times \max\left(\Re_4, \frac{N \times \sqrt{|\Re_1 \times M_{\max} + tf_{td}|}}{T} \right) + tf_{td} \right)} \right|} \right) \tag{23}$$

where $\Re_1 = 8.58941$, $\Re_2 = 2.2$, $\Re_3 = 1.2$, $\Re_4 = 0.25$, and all logs and square roots are of absolute values.

The performance of each of run 5, run 13 and BM25 on the evaluation set is plotted as an 11-point precision/recall graph in figure 4.

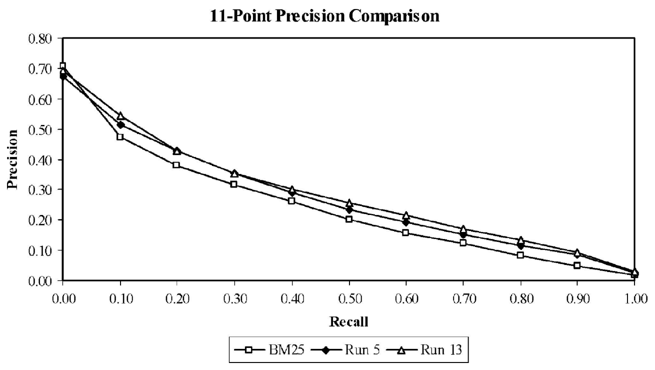

Figure 4. 11-point precision graphs showing performance of each algorithm on the evaluation set.

For the cross validation experiment, 13 runs of 100 generations, using 100 individuals including seeding were trained on the evaluation set. A successful experiment would result in over-fitting to this set, and an estimate of the best possible MAP using the evidence available. This experiment failed to show improvements in line with those learned during training (see Table 2, last row). The assumption genetic programming will learn the optimal function with these parameters is therefore invalid. As it is reasonable to assume genetic programming will succeed, possible explanations include: first, if learning is not complete after 100 generations, the optimal function will not have been learned when the cross validation runs were terminated. Second, if the population size was too small, there may be insufficient diversity in the population to learn the optimal function. Third, if there are an insufficient number of runs, and each run became stuck at local maxima then the global maxima will not have been found. Evidence of the first can be seen in figure 3 where the gradient is not 0 after 100 generations, further evidence is needed to lend validity to the others. No significance test is necessary as the cross validation experiment failed.

As the validation experiment failed to find an optimum, it is also reasonable to assume the training experiments failed to find an optimum. If this is the case no evidence of over fitting is expected. In every case Table 2 shows performance of the evaluation set exceeding that on the training set, this is likely to be because no over fitting occurred; and the evaluation set is an easier set than the training set. If no over fitting occurred, learning was not complete and improvements in excess of those reported here are expected.

The average precision for each evaluation query can be seen in figure 5. Figure 6 presents the improvements, both are compared to BM25. For run 5, 63% of queries show

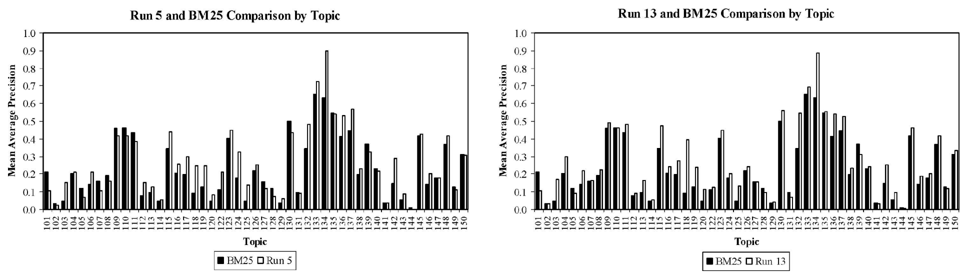

Figure 5. Query by query comparison of BM25 and learned functions on the evaluation set.

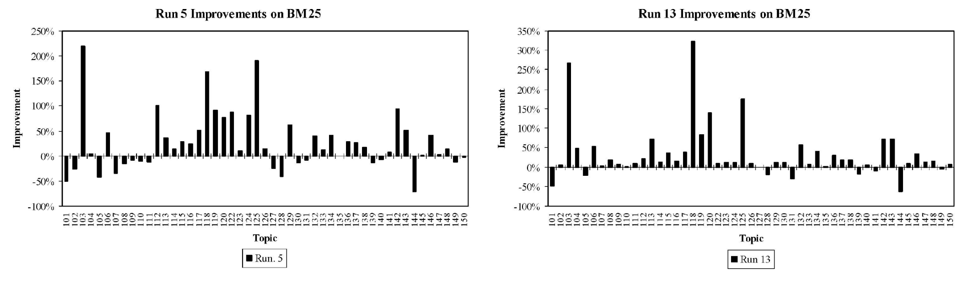

Figure 6. Query by query improvements shown in learned functions on the evaluation set.

Table 3. MAP computed against cystic fibrosis collection. Improvements are shown relative to BM25.

Equation	MAP	Improv.(%)	ROI(%)	P(%)	Certainty (1-P)
Inner Product	0.28722	5.30	60.20	2.57	97.43
Run 5	0.28604	4.86	55.10	0.67	99.33
Run 13	0.28422	4.20	60.20	1.67	98.33
Oren	0.28201	3.38	56.12	9.30	90.70
Probability	0.27814	1.97	51.02	14.37	85.63
Okapi BM25	0.27277	0.00	0.00	0.00	100.00
Cosine	0.15996	−41.36	9.18	0.00	100.00

improvement whereas in run 13, 82% of queries show an improvement. A one-tailed t-test resulted in a P value of less than 1% in both cases; there is less than a 1% probability that the differences are due to chance alone.

The results of testing against the cystic fibrosis collection are presented in Table 3. Oren's function is included even though overflow occurred. When overflow occurred for a given document, that document was removed from the result set and average precision computed as if it had not been identified as relevant. Oren's choice to use stemming may explain why Oren did not encounter this problem. On this collection both learned functions outperform BM25 and Oren, as does inner product. However, the difference between inner product and the learned functions is very small (0.00118, less than 0.5%).

Run 5 was compared to BM25 using the collections on TREC disks 4 and 5 and topics 301–350. Results are presented in Table 4. On only one collection (FBIS) is there a difference significant at the 1% level, in which case run 5 out performs BM25 by 23.24%. In no cases was numeric overflow seen in run 5.

Table 4. Comparison between run 5 and BM25 on collections from TREC disks 4 and 5 using queries constructed from the narrative of topics 301–350.

		BM25	Run 5				
Collection	# of Topics	MAP	MAP	Improv.(%)	ROI(%)	P(%)	Certainty (1-P)
TREC4	40	0.17675	0.17021	−3.70	42.50	20.90	79.10
FT	36	0.20224	0.22227	9.91	61.11	8.13	91.87
CR	19	0.19492	0.17267	−11.42	17.27	21.49	78.51
FR94	8	0.20790	0.18196	−12.48	25.00	6.20	93.80
TREC5	38	0.18243	0.18342	0.54	52.63	47.95	52.05
FBIS	25	0.20818	0.25656	23.24	68.00	0.41	99.59
LATIMES	36	0.19964	0.18809	−5.78	18.81	27.16	72.84
TREC 4+5	48	0.16580	0.16249	−2.00	50.00	39.63	60.37

Table 5. Comparison between run 13 and BM25 on collections from TREC disks 4 and 5 using queries constructed from the narrative of topics 301–350.

			Run 13				
Collection	# of Topics	BM25 MAP	MAP	Improv.(%)	ROI(%)	P(%)	Certainty (1-P)
TREC 4	40	0.17675	0.18509	4.72	47.50	30.54	69.46
FT	36	0.20224	0.25019	23.71	61.11	0.94	99.06
CR	19	0.19492	0.19608	0.59	19.61	49.04	50.96
FR94	8	0.20790	0.19033	−8.45	25.00	19.91	80.09
TREC 5	38	0.18243	0.18833	3.23	42.11	40.90	59.10
FBIS	25	0.20818	0.27667	32.90	56.00	4.82	95.18
LATIMES	36	0.19964	0.19204	−3.81	19.20	37.76	62.24
TREC 4 + 5	48	0.16580	0.18733	12.99	43.75	17.52	82.48

When run 13 was compared to BM25 (Table 5), again no numeric overflow was seen (when $\mathcal{A}_{d,t}$ is 0, $log_2(\mathcal{A}_{d,t})$ is infinite, however, $\min(L_d, \log_2(A_{d,t}))$ must be L_d). Of the differences significant at the 1% level, run 13 always betters BM25. At the 5% level, again run 13 always betters BM25. On the TREC collections, run 13 always outperformed run 5.

On the collections tested and with the parameters tested, run 13 is a better function than run 5 and BM25, but it is not yet reasonable to recommend it as a replacement for BM25. Tests on web documents, tests as query length varies, tests with stemming, and tests with passage retrieval are yet to be performed.

7. Efficiency

Learning was terminated after 100 generations due to time considerations; a learning run of this length took all weekend on a 2.4 GHz Pentium 4 with 500 Mb memory and indexes held on a network drive.

Searching is disk-I/O bound (Williams and Zobel 1999, Zobel and Moffat 1995), therefore any reduction in I/O will reduce the learning time. Three optimizations were used.

First, MAPs computed against the training set were stored with each individual. Should the individual reproduce, the MAP would not change so it was not recalculated. The time to complete a single generation is therefore dependant on the number of new individuals in that generation. This optimization will have no effect in the first generation (where all individuals are new) but does in subsequent generations.

Second, if overflow was detected a search was cut short. If overflow was detected in the first term of a multi-term query, the influence of the subsequent terms was not computed. The time to complete the search was reduced, thus the time to complete the generation was reduced.

Third, to reduce query evaluation time, short queries constructed from just the TREC topic description were used.

Even without optimizations, the time required to learn a ranking function is small by comparison to the lifetime expectancy of the function.

7.1. Possible further improvements

There are many proposed extensions to the genetic programming learning algorithm. The experiments herein used elitism (De Jong 1975) to ensure the fittest individual in each generation would be carried into the next generation. There is a plethora of further extensions shown to be effective in many domains; two that might prove effective in learning general purpose ranking functions are: memetic searching, and the island model. These might be used either alone or together

In a memetic search, the global genetic search strategy is mixed with a localized hill-climbing search. When an individual is created through mutation or crossover, the new individual is used as a starting point for a localized search. After some number of local search iterations the new individual is added to the next generation. This new local-found-individual is typically fitter then the genetic seed. Memetic searching has been effectively used with genetic algorithms (Hart and Belew 1996) and genetic programming (Smart and Zhang 2004).

In the island model (Tongchim and Chongstitvatana 2000), a number of populations are run in parallel and concurrently; as if each were on a separate isolated island. After some fixed number of generations, a small proportion of individuals migrate from one island to another. This shifts good genetic material from one population to another. Should an individual island population have become stuck at a local maximum, this injection of differently good individuals has been shown at effectively help the search jump out of the local maximum.

There is also a plethora of small adaptations of the core learning algorithm. Herein fitness proportionate selection was used to choose individuals for reproduction, tournament selection (Koza 1992) might be used as an alternative. Herein, the initial population was created with completely random individuals, and then seeded with known good functions. As an alternative the ramped half and half method (Koza 1992) might be used. Consideration might also be given to dropping the seeding process.

Further investigation is necessary to determine the best learning environment.

8. Summary

A term-oriented inverted file information retrieval system was used to learn two new general purpose ranking functions. Candidate documents were identified by the presence of at least one search term. These documents were then ordered using a ranking function. As the ranking function did not impact the identification of candidate documents, different functions did not affect recall. The choice of retrieval methodology limited the evidence available for ranking, but did not limit the ranking function to being a sum over each query term.

Mean average precision for the TREC WSJ collection was computed for inner product, probability, cosine, and BM25. BM25 was shown to outperform the others. Genetic programming was then used to learn new functions on this collection. 13 parallel runs of 100 individuals were run for 100 generations. Two good functions were identified, each exceeding BM25 by over 10% during training.

The two new functions, run 5 and run 13 were evaluated on queries and documents not used during training. In no test was BM25 shown to significantly outperform the new functions; however the new functions were shown to significantly outperform BM25 for some collections. Significant mean average precision improvements as high as 32% were observed, however this was atypical.

Genetic programming has proven a successful way to develop general purpose ranking functions for unstructured information retrieval.

References

Anh VN and Moffat A (2002) Improved retrieval effectiveness through impact transformation. Australian Computer Science Communications, 24(2):41–47.

Buckley C (1991) trec_eval. Available: ftp.cs.cornell.edu/pub/smart.

Buckley C and Voorhees EM (2000) Evaluating evaluation measure stability. In: Proceedings of the 23rd ACM SIGIR Conference on Information Retrieval, pp. 33–40.

Clarke CLA, Cormack GV and Tudhope EA (2000) Relevance ranking for one to three term queries. Information Processing and Management, 36(2):291–311.

De Jong KA (1975) An analysis of the behavior of a class of genetic adaptive systems. Unpublished Ph.D., University of Michigan.

Fan W, Gordon MD and Pathak P (1999) Automatic generation of a matching function by genetic programming for effective information retrieval. In: Proceedings of the 1999 American Conference on Information Systems.

Fan W, Gordon MD and Pathak P (2004) A generic ranking function discovery framework by genetic programming for information retrieval. Information Processing and Management, 40(4):587–602.

Fan W, Gordon MD, Pathak P, Xi W and Fox EA (2004) Ranking function optimization for effective web search by genetic programming: An empirical study. In: Proceedings of the 37th Annual Hawaii International Conference on System Sciences.

Grefenstette JJ (1986) Optimization of control parameters for genetic algorithms. IEEE Transactions on Systems, Man, and Cybernetics, 16(1):122–128.

Harman D (1992) Ranking algorithms. In: Frakes WB and Baeza-Yates R. Eds., Information retrieval: Data Structures and Algorithms Englewood Cliffs, New Jersey, USA, Prentice Hall, PP. 363–392.

Harman D (1993) Overview of the first TREC conference. In: Proceedings of the 16th ACM SIGIR Conference on Information Retrieval, pp. 36–47.

Hart WE and Belew RK (1996) Optimization with genetic algorithm hybrids that use local searches. In: Belew, R.K. and Mitchell M. Eds. Adaptive Individuals in evolving Populations: Models and Algorithms Addison-Wesley Longman Publishing Co., Inc. pp. 483–496.

Hawking D, Craswell N, Thistlewaite P and Harman D (1999) Results and challenges in web search evaluation. In: Proceedings of the 8th International Conference on World Wide Web, pp. 1321–1330.

Holland JH (1975). Adaptation in Natural and Artificial Systems. Ann Arbor, University of Michigan Press.

Igel C and Chellapilla K (1999) Investigating the influence of depth and degree of genotypic change on fitness in genetic programming. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '99), pp. 1061–1068.

Kaszkiel M and Zobel J (1998) Term-ordered query evaluation versus document-ordered query evaluation for large document databases. In: Proceedings of the 21st ACM SIGIR Conference on Information Retrieval, pp. 343–344.

Kekäläinen J and Järvelin K (2002) Using graded relevance assessments in IR evaluation. Journal of the American Society for Information Science and Technology, 53(13):1120–1129.

Khuri S, Bäck T and Heitkötter J (1994) An evolutionary approach to combinatorial optimization problems. In: Proceedings of the 22nd Annual ACM Computer Science Conference, pp. 66–73.

Kleinberg JM (1999) Authoritative sources in a hyperlinked environment. Journal of the ACM, 46(5):604–632.

Koza JR (1992) Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge, MA, USA.

Oren N (2002a) Improving the effectiveness of information retrieval with genetic programming. Unpublished M.Sc., University of the Witwatersrand, Johannesburg.

Oren N (2002b) Reexamining tf.idf based information retrieval with genetic programming. In: Proceedings of the 2002 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists on Enablement through Technology (SAICSIT), pp. 224–234.

Page L, Brin S, Motwani R, and Winograd T (1998) The PageRank Citation Ranking: Bringing Order to the Web (1999–66), Stanford Digital Library Technologies Project.

Pôssas B, Ziviani N, Meira W, and Ribeiro-Neto B (2002) Set-based model: A new approach for information retrieval. In: Proceedings of the 25th ACM SIGIR Conference on Information Retrieval, pp. 230–237.

Raghavan VV, Shi H-p, and Yu CT (1983) Evaluation of the 2-Poisson model as a basis for using term frequency data in searching. In Proceedings of the 6th ACM SIGIR Conference on Information Retrieval, pp. 88–100.

Robertson SE and Sparck Jones K (1976) Relevance weighting of search terms. Journal of the American Society for Information Science, 27(3):129–146.

Sparck Jones, Robertson SE and Walker S (1999) Okapi/Keenbow at TREC-8. In: Proceedings of the 8th Text REtrieval Conference (TREC-8).

Robertson SE, Walker S, Beaulieu MM, Gatford M, and Payne A (1995) Okapi at TREC-4. In: Proceedings of the 4th Text REtrieval Conference (TREC-4), pp. 73–96.

Robertson SE, Walker S, Jones S, Beaulieu MM, and Gatford M (1994) Okapi at TREC-3. In Proceedings of the 3rd Text REtrieval Conference (TREC-3), pp. 109–126.

Salton G and Buckley C (1988) Term weighting approaches in automatic text retrieval. Information Processing and Management, 24(5):513–523.

Salton G, Wong A, and Yang CS (1975) A vector space model for automatic indexing. Communications of the ACM, 18(11):613–620.

Savoy J, Ndarugendamwo M, and Vrajitoru D (1995) Report on the TREC-4 experiment: Combining probabilistic and vector-space schemes. In: Proceedings of the 4th Text REtrieval Conference (TREC-4), pp. 537–547.

Shaw WM, Wood JB, Wood RE, and Tibbo HR (1991) The cystic fibrosis database: Content and research opportunities. Library and Information Science Research, 13: 347–366.

Smart W and Zhang M (2004) Applying online gradient-descent search to genetic programming for object recognition. In: Proceedings of the 2nd Workshop on Australasian Information Security, Data Mining and Web Intelligence, and Software Internationalisation.

Tongchim S and Chongstitvatana P (2000) Comparison between synchronous and asynchronous implementation of parallel genetic programming. In: Proceedings of the 5th International Symposium on Artificial Life and Robotics (AROB), pp. 251–254.

Williams HE and Zobel J (1999) Compressing integers for fast file access. Computer Journal 42(3):193–201.

Witten IH, Moffat A, and Bell TC (1994) Managing Gigabytes: Compressing and Indexing Documents and Images. Van Nostrand Reinhold, New York, USA.

Zobel J and Moffat A (1995) Adding compression to a full-text retrieval system. Software—Practice and Experience, 25(8):891–903.

Zobel J and Moffat A (1998) Exploring the similarity space. SIGIR Forum, 32(1):18–34.

Zobel J, Moffat A, and Ramamohanarao K (1998) Inverted files versus signature files for text indexing. Transactions on Database Systems, 23(4):453–490.