

IMPROVING THE EFFECTIVENESS OF
INFORMATION RETRIEVAL WITH
GENETIC PROGRAMMING

NIR OREN
DECEMBER 2002

IMPROVING THE EFFECTIVENESS OF INFORMATION RETRIEVAL WITH GENETIC PROGRAMMING

A RESEARCH REPORT
SUBMITTED TO THE FACULTY OF SCIENCE,
UNIVERSITY OF THE WITWATERSRAND, JOHANNESBURG,
IN PARTIAL FULFILMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE

NIR OREN
DECEMBER 2002

DECLARATION

I declare that this research report is my own, unaided work. It is being submitted for the Degree of Master of Science in the University of the Witwatersrand, Johannesburg. It has not been submitted before for any degree or examination in any other University.

Nir Oren

_____ day of _____ 2002

Abstract

Information retrieval (IR) systems are responsible for the storage and retrieval of large amounts of data in an efficient manner. An important subtask of IR is searching, which deals with retrieving documents stored within the system in response to a user's information needs.

Many IR systems deal with text documents, and a myriad of techniques has been proposed to fulfil this task. All techniques attempt to classify documents based on their relevance to the user's needs. Amongst the most popular of approaches, due to both their simplicity and relative effectiveness, are vector based techniques based on a scheme utilising the frequency of words in the document and document collection. One class of these schemes is known as *tf.idf*. While popular, *tf.idf* schemes have very little theoretical grounding, and in many cases, their performance is not adequate.

This research seeks to find alternative vector schemes to *tf.idf*. This is achieved by using a technique from machine learning known as genetic programming. Genetic programming attempts to search a program space for "good" solutions in a stochastic directed manner. This search is done in a manner motivated by evolution, in that the good programs are more likely to be combined to form new programs, while poor solutions die off.

Within this research, evolved programs consisted of a subset of possible classifiers, and one program was deemed better than another if it better classified documents as relevant or irrelevant to a user query.

The contribution of this research is an evaluation of the effectiveness of using genetic programming to create classifiers in a number of IR settings. A number of findings were made: It was discovered that the approach proposed here is often able to outperform the basic *tf.idf* method: on the CISI and CF datasets, improvements of just under five percent were observed. Furthermore, the form of the evolved programs indicates that classifiers with a structure different to *tf.idf* may have a useful role to play in information retrieval methods. Lastly, this research proposes a number of additional areas of investigation that may further enhance the effectiveness of this technique.

Acknowledgments

I'd like to thank my supervisor, David Lubinsky, as well as my other panel members, Scott Hazelhurst and Sheila Rock for all their valuable advice.

I'd also like to thank my family and friends for their support, encouragement and (occasionally) entertainment value. Without these people, this research report would have never been written, or, in some cases, been ready a little earlier. In alphabetical order, they include: Brynn Andrew, Adi Attar, Anton Bergheim, Helen Every, Stewart Gebbie, Tina Götschi, Aileen Marson, Gideon Oren, Hana Oren, Iris Oren, Pepsi Oren, Pekka Pihlajasaari, Phillip Schmitz and Greg Watkins.

Contents

Declaration	ii
Abstract	iii
Acknowledgments	iv
Contents	v
List of Figures	vii
List of Tables	ix
1 Introduction	1
1.1 Problem Introduction and Motivation	1
1.1.1 The Information Retrieval Process	1
1.1.2 Genetic Programming	2
1.2 Statement of problem and objectives of the research	2
1.3 Organisation of the research report	3
1.4 Summary of results	3
2 Background	4
2.1 Genetic Algorithms and Genetic Programming	4
2.1.1 Genetic Algorithms	4
2.1.2 Genetic Programming	5
2.2 Information Retrieval	8
2.2.1 Simple text searches	9
2.2.2 Boolean and probabilistic approaches to IR	10
2.2.3 The vector model of IR	12
2.2.4 Other IR models	13
2.2.5 Evaluation methods for IR systems	16
2.2.6 Improving IR performance	20
2.2.7 Applications of machine learning techniques to IR	21

2.3	Context	23
3	Applying Genetic Programming to Information Retrieval	24
3.1	Introduction	24
3.2	Research question	24
3.3	Combining Genetic Programming with vector-based information retrieval	25
3.4	Experimental setup	25
3.5	The experiments	30
3.5.1	Experiment 1: Evaluation using new queries	30
3.5.2	Experiment 2: Evaluation with new queries and documents	32
3.5.3	Experiment 3: Evaluation on a new document collection	32
3.5.4	Remaining research questions	32
3.6	Evaluation criteria	33
4	Results and evaluation	34
4.1	Method of evaluation	34
4.2	Results	34
4.2.1	Experiment 1: Evaluation using new queries	34
4.2.2	Experiment 2: Evaluation with new queries and documents	40
4.2.3	Experiment 3: Evaluation on a new document collection	44
4.3	A few words about training time	47
4.4	An examination of evolved individuals	47
4.5	Summary and Conclusions	48
5	Conclusions and further research	50
5.1	Summary of research	50
5.2	Summary of results and conclusions	50
5.3	Future work	51
5.3.1	Modifying breeding parameters	51
5.3.2	Increased dataset size	51
5.3.3	Different evaluators for queries and documents	51
5.3.4	Additional terminal types and operators	52
5.3.5	Multiple evaluation stages	52
5.3.6	Relevance feedback training	52
5.4	Contributions of this research	53
5.5	Final word	53

List of Figures

2.1	A sample parse tree.	6
2.2	The basic components of an information retrieval system.	8
2.3	A sample precision-recall diagram.	18
3.1	Pseudo code for training algorithm	26
3.2	A sample document extract and query.	27
3.3	Algorithm for initial individual creation	31
4.1	The maximum, mean and minimum fitness of the best individual within each generation during training, as evaluated using both fitness measures.	35
4.2	A typical average fitness curve obtained by computing the average fitness for each generation against the training data.	36
4.3	Testing fitness of the individuals deemed fittest for each generation during training, evaluated with both fitness metrics.	36
4.4	Minimum, mean and maximum precision–recall curves for the best individuals obtained using the average precision metric run on the testing dataset, as well as minimum, mean and maximum precision–recall curves for <i>tf.idf</i> run on the same test data.	37
4.5	Minimum, mean and maximum precision–recall curves for the best individuals obtained using the precision at 50% recall metric run on the testing dataset, as well as minimum, mean and maximum precision–recall curves for <i>tf.idf</i> run on the same test data.	39
4.6	Precision–recall curves for both metrics, evaluated over the full query set.	39
4.7	The simplified form of the best evaluator obtained during Experiment 1. Letters represent terminals as shown in Table 3.1.	40
4.8	Maximum, mean and minimum fitnesses of the best evaluators for each generation, tested on testing documents and testing data.	41
4.9	Maximum, mean and minimum fitnesses of the best evaluators for each generation, tested on testing documents and training queries.	42
4.10	Maximum, mean and minimum fitnesses of the best evaluators for each generation, tested on training set documents and test set queries.	42
4.11	The simplified form of the best evaluator for Experiment 2.	43

4.12	Minimum, maximum and mean fitness of the best evaluators for six experiments with two training set combinations, using the same identical training and testing datasets.	45
4.13	Minimum, maximum and mean evaluation fitness of the best evaluators when trained on the Cystic Fibrosis dataset and tested on the CISI dataset.	46
4.14	Precision recall curve for the best evaluator and <i>tf.idf</i> as trained on the standard data set and evaluated on the CISI dataset.	46
4.15	The simplified form of the fittest individual evolved during experiment 3, as judged during the evaluation phase.	47
4.16	The unsimplified form of the best individual evolved during experiment 1.	48

List of Tables

3.1	Nodes used to generate programs	29
3.2	Breeding operations together with the likelihood of their being used.	30
4.1	Precision–recall values for the average precision metric and <i>tf.idf</i> , evaluated on the testing dataset.	38
4.2	Precision–recall values for the precision at 50% recall metric and <i>tf.idf</i> , evaluated on the testing dataset.	38
4.3	Precision–recall values for both metrics ,evaluated over the full query set.	40
4.4	Precision–recall values for the best evaluator and <i>tf.idf</i> , evaluated on the CISI dataset.	44

Chapter 1

Introduction

1.1 Problem Introduction and Motivation

Individuals have access to ever increasing amounts of digital information. Similarly, organisations are storing increasing amounts of their “corporate knowledge” in digital forms. When faced with a task requiring specific information, users require tools that provide them with fast access to relevant information.

An information retrieval (IR) system is a system that caters for the storage and retrieval of electronic data. Often, the data stored in an IR system consists of text documents, but more and more IR systems cater for multimedia content such as video and audio [5].

Data stored within an IR system is accessed by posing an information request to the system, usually in the form of a query describing the desired data. The system then returns that data which it deems to be relevant in answering the query.

While IR systems are gaining popularity due to the lack of alternatives in managing the overwhelming amount of electronic data, the quality of their responses is still poor [2]. Increasing the appropriateness of the results returned by IR systems is critical to dealing with the ever rising amounts of data that people and organisations must handle.

1.1.1 The Information Retrieval Process

Traditional IR systems focus on the retrieval of documents from a collection based on a query provided to the system by the user. After processing the query, the system returns a list of documents to the user, usually consisting of the document title and a short extract. The list of documents is most commonly sorted with the ordering based on the system’s belief of the document relevance to the user’s query.

One of the most popular paradigms in IR, due to its relative power and simplicity, is the *vector* model of IR. In this approach, documents and queries are represented as vectors, with each element within the vector taking on a value based on the presence or absence of a word within the text. To determine the relevance of a document for a given query, a similarity operation (normally a dot product) is conducted on the vectors, yielding a single number.

The value assigned to a vector element is commonly computed using one of a family of weighting algorithms known as *tf.idf* — these algorithms are based on the term frequency (tf) and inter-document frequency (idf) of words appearing in the documents. Empirical studies [41] indicate that *tf.idf* approaches, while simple, are very effective. However, nothing indicates that *tf.idf* type approaches are optimal in any theoretical sense. The research presented here builds on this fact by performing a search through a portion of the space of possible weighting techniques, attempting to find algorithms that return documents more relevant to a user's needs, as compared with standard *tf.idf*. The search is performed using genetic programming (GP).

1.1.2 Genetic Programming

The genetic programming paradigm attempts to “evolve” programs well suited to performing a specific task. By creating a large population of individual programs, GP can perform a highly parallel search of program space. Programs from the population that appear best suited to solve the given task are copied, perhaps with modifications, to form a new generation of programs. This new generation is then once more evaluated on the task, and the process is repeated until one either runs out of patience or programs that solve the task sufficiently well are found.

Genetic programming was used to create new evaluation functions. These functions were rated based on how well they classify documents over queries taken from a training set. Functions that accurately rate documents as relevant (or irrelevant) were allowed to propagate to the next generation.

1.2 Statement of problem and objectives of the research

Over the years, many different weighting schemes have been proposed for vector based IR. This research proposes another method of creating effective weighting algorithms. The research question that I attempt to answer is:

Can a weighting algorithm be produced using genetic programming methods that

- 1. has better recall and precision than existing weighting schemes*
- 2. operates well on generalised document collections, and*
- 3. requires the same order of magnitude of resources (computing power, time) as existing methods?*

The objectives of this research are to:

1. Determine the feasibility of using GP approaches to discover new indexing algorithms.
2. Determine the effectiveness of algorithms created using this approach to answer queries posed on general collections.

Achieving these objectives leads to either the creation of a new indexing algorithm, or further validation for the continued use of an existing indexing approach.

1.3 Organisation of the research report

The remainder of the report is structured as follows:

Chapter 2 – Background. This chapter presents the background literature that relates to GP and IR. An overview of GP is given, starting with a look at genetic algorithms (GA). The three dominant models of IR are then examined, after which extensions to these models are presented. The evaluation of IR systems poses some problems, and is therefore discussed in some detail. Extensions to the basic IR model are then described. The chapter concludes with an examination of the application of machine learning techniques to IR, with a focus on GP.

Chapter 3 – Applying GP to IR. In this chapter, the approach taken within this research is discussed in further detail. The experimental setup is examined, and details of the experimental setup are provided.

Chapter 4 – Results and Evaluation This chapter reports on the results obtained from running the experiments. This chapter also interprets and analyses the results, attempting to understand what they mean in the context of the approach, as well as why the experiments yielded the results they did.

Chapter 5 – Conclusions and Further Research This chapter discusses the experiments, results, and research as a whole in the context of the field of information retrieval. It also suggests some ideas for the extension of this research.

1.4 Summary of results

Overall, the approach proposed by the research appears to have some merit. Improvements of close to five percent in retrieval accuracy were seen in some experiments, and, as Chapter 5 shows, many possible enhancements exist for further improving performance.

It was seen that the ability of the genetic programming to generate applicable classifiers appeared dependent on the amount of available training data. Thus, while the approach performed poorly in some of the experiments, much of this could be attributed to the small size of the data set available for training within that experiment.

Chapter 2

Background

This chapter aims to provide the reader with the background necessary to evaluate the research described by this document.

This chapter begins by providing an overview of the genetic programming paradigm. The following section gives a brief survey of some common information retrieval techniques. Methods for evaluating the effectiveness of an IR system are then examined, followed by an examination of the effectiveness of various IR schemes. After examining ways of improving the performance of basic IR techniques, machine learning in IR is discussed.

2.1 Genetic Algorithms and Genetic Programming

Genetic programming is a machine learning technique that is used to generate programs that can perform a specific set of tasks. GP has been successfully used in a number of areas, including circuit design [30], robotics [13], and artificial life research [10].

Before venturing into an examination of genetic programming, a brief aside must be taken to look at genetic algorithms. A more detailed discussion of both of these topics can be found in Mitchell's book [31].

2.1.1 Genetic Algorithms

Genetic algorithms [23] describe a set of optimisation techniques that, given a goal or fitness function, are used to search a space for optimal points. The space is searched in a directed, stochastic manner, and the method of searching borrows some ideas from evolution.

An application of a genetic algorithm contains a population of individuals, each representing a point in the space. Each individual has an associated fitness value, calculated using a global fitness function, which reflects how close the individual is to an optimal solution. The population is evolved through successive generations, with the population at each generation "bred" from the fittest members of the previous generation. Other parameters for controlling the genetic algorithm are: the population size, the maximum number of generations to be run, and parameters controlling the details of breeding.

Most commonly, an individual consists of a fixed length vector (also known as its chromosome), and the individual's fitness is computed directly from this vector. If any individual is found with a fitness exceeding a predetermined threshold, the algorithm terminates, and that individual is returned as the solution to the problem. The run is also terminated if the maximum number of generations have been bred, in which case the fittest individual is found and returned. If the run is not terminated, a new generation is bred.

Many different methods for breeding the new generation of individuals have been proposed. In the most common approach, each time an individual in the new generation is to be created, a breeding operation is selected probabilistically from reproduction, crossover, or mutation. The probability of a specific operation being selected is based on prespecified parameters. The selected breeding operation is then carried out on various individuals in the population, with higher fitness individuals more likely to be operated upon.

The *reproduction* operation involves copying the selected individual into the next generation.

Crossover creates a number of new individuals from the same number of old individuals by randomly recombining chosen subvectors from the selected individuals in the old population. Usually, crossover operates on a pair of individuals, in which case it resembles biological sexual reproduction, with the new individuals consisting of parts of their parents.

Mutation operations create a new individual from an existing individual by randomly changing the value of one of the individual's vector components. Some genetic algorithm schemes do not treat mutation as a unique breeding operation, instead, every time reproduction or crossover occurs, a small probability exists that the resulting individual will be mutated.

Common variations on this basic scheme include limiting the number of times a breeding operation can occur within a single generation, and altering the method by which individuals are selected for breeding. More complex operations [17] include using diploid chromosomes (pairs of vectors, with dominant components contributing to the fitness evaluation), position independent components, and a number of other operators to improve genetic algorithm performance.

In practice, genetic algorithms have proven very effective in searching through complex, highly nonlinear, multidimensional search spaces [4]. Another attractive feature is the fact that they are highly domain independent: the algorithm needs no information about the problem domain, as it is the implementor's responsibility to define "good" vector and fitness representations, using their domain knowledge.

2.1.2 Genetic Programming

A program designed to solve a particular class of problems can be thought of as residing at a point on a scalar field (a scalar function evaluated at every point in the space) within the space of all possible programs. The field is calculated according to the error of the program at satisfying the class of problems [27].

Genetic programming, first introduced by Koza [25] adapts the approach used in genetic algorithms to search through program space in an attempt to find a program which solves a predefined problem as accurately as possible.

A number of obstacles exist when attempting to use genetic algorithms directly. Firstly, it is difficult to find a fixed size vector representation for a set of programs, as program lengths vary. Similarly, determining

a natural representation for instructions as vector components is not trivial. Finally, a way of performing the various breeding operations needs to be determined.

On the other hand, determining fitness for an individual program is trivial: simply run the program on a number of datasets for which results are known, and set the program's fitness value in inverse proportion to the deviation of the output from the expected result.

To overcome the difficulties described above, genetic programming usually models a program as a variable sized tree. Each node in the tree represents an operation, which takes in values computed by its children as parameters. Terminal nodes commonly represent input values taken from the problem domain. Figure 2.1 illustrates a parse tree representation of a program used to calculate the term weightings of a single term in information retrieval, as described in Section 2.2.

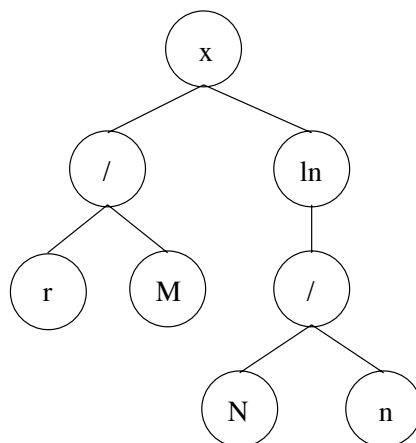


Figure 2.1: A parse tree representation of the *tf.idf* function $\frac{r}{M} \ln \frac{N}{n}$. Terminal nodes r , M , N , and n represent various document attributes, as described in Table 3.1.

The operation of the nodes in a GP is predefined by the implementor — the only freedom the algorithm has is in deciding on the structure and size of the node tree.

By operating on the tree, a process similar to the one described in the previous section can be followed to create a program suitable for the problem domain:

1. A population of random programs, represented as parse trees, with nodes representing possible operations, is created. It is often possible to shorten training times, as well as integrate domain knowledge into the algorithm by introducing hand crafted programs as well as random programs into the initial population.
2. The population is run over a set of sample data from within the problem domain, and a fitness measure is allocated to each program based on the accuracy of its result.
3. A new generation is bred by carrying out breeding operations on the programs from the current generation. Programs with a higher fitness are more likely to be operated upon.

4. The process is repeated until a certain number of generations has been created, or a program with sufficient fitness is bred.

The breeding operations discussed in the previous section carry through in a straightforward manner. The copy operation moves a program tree into the next generation unchanged. Crossover operations involve deciding on a crossover point within the two programs to be mixed, and then moving the subtree below the point from the first program to the second, and vice-versa. Mutation is only sometimes used, and involves a two stage process: initially a mutation point, consisting of a node in the tree, is selected. The subtree below this point is then removed, and replaced with a randomly generated subtree, possibly of a different size to the original. Additional, rarely used, operations in GPs involve permutation, where the subtree below the permutation point is reversed, and editing, which uses prespecified rules to simplify the program (for example, by changing all Boolean disjunction expressions with identical child nodes to a single instance of the child node).

A commonly encountered problem in genetic programming is the handling of different types in the parse tree. This situation arises when child nodes return values that cannot be handled by a parent node (for example, a node returning a true or false value to a node that expects an integer). A number of methods exist to deal with this problem [25]: it is possible to deem all programs where this occurs unfit, thereby minimising their chances of progressing to the next generation. In some cases, it is possible to massage the returned types into a form suitable for consumption by the parent — in the previous example, truth could be represented by 1, and falsehood by 0, allowing a node expecting integers to handle this situation.

A related problem, with similar solutions, involves dealing with illegal operations. An example of this would involve a division node asked to divide any number by 0. An alternate solution to this problem would involve modifying the operation to handle illegal data. For example, in the division by zero case, the division operator could be modified to set the denominator to a very small but non-zero value.

Overtraining, also known as overfitting, is a problem that GP shares with many other machine learning techniques. The definition for overtraining used here is similar to Mitchell's definition of overfitting for decision trees [31]: Given a program space P , a program $p \in P$ is said to overfit the training data if there exists some alternative program $p' \in P$, such that p has a smaller error than p' over the training examples, but p' has a smaller error than P over the entire distribution of instances.

Overtraining normally appears after a number of generations have been evaluated on the training set. It forms due to the generated individuals adapting to the data found in the training set rather than the underlying data that the training set is supposed to represent. The easiest way to delay the appearance of overtraining is thus to have a large set of training examples. Another common method of mitigating overtraining is the use of an additional validation dataset. Training takes place on one dataset, while the fitness of the individuals is monitored using a second dataset. Fitness drops on the second dataset indicate that overtraining may be occurring. Since the use of a validation dataset requires additional training data, which may not be available, a technique known as k -fold cross-validation has been developed. Under this approach, the dataset is partitioned into k approximately equal sized sets. During each generation, one of these subsets is chosen as the validation dataset, with the remaining sets combined to form the training dataset. These techniques, together with a number of other approaches to dealing with overtraining are described in Mitchell's book [31].

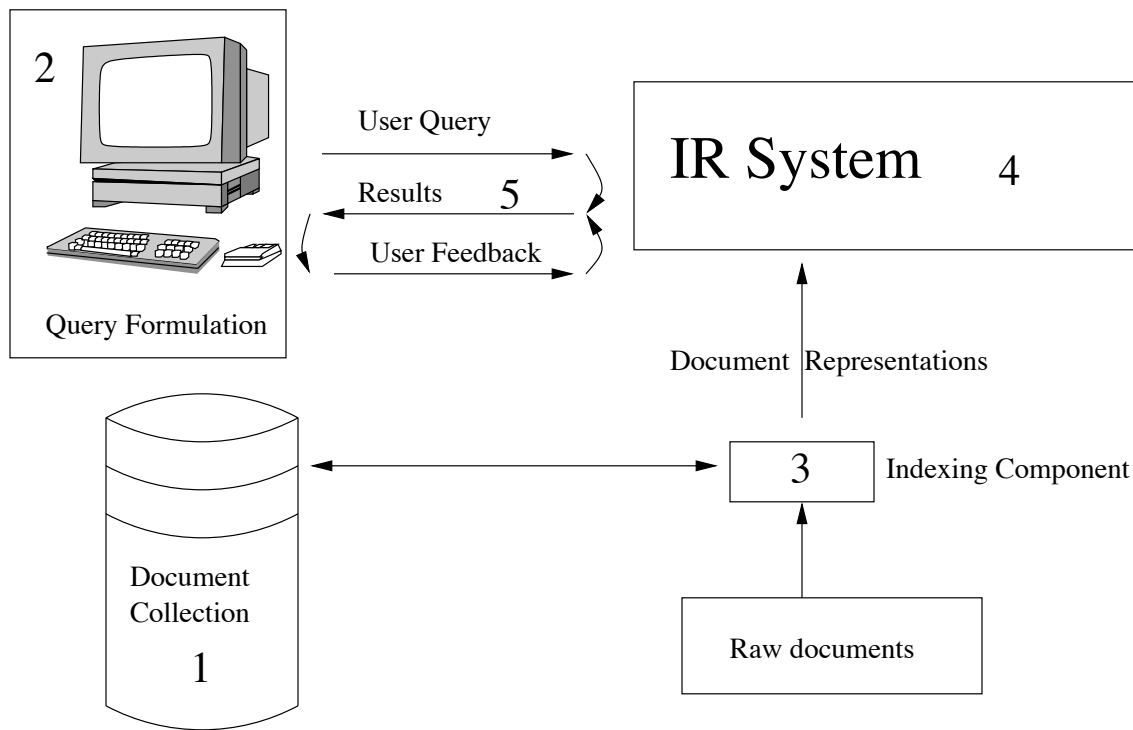


Figure 2.2: The basic components of an information retrieval System (as described in [2])

Many extensions to genetic programming have been proposed (for example automatically defined functions [3]). While these schemes often enhance the performance of the approach, they were not used in the course of this research, and are thus not examined here.

2.2 Information Retrieval

An IR system is used to retrieve data applicable to a user's needs based on queries posed to the system by the user. Figure 2.2 illustrates the operation of a generic IR system. As shown, a user poses a query to the system. This query is parsed by the system, and is used to select documents from the document collection indexed by the system. Results are then returned to the user. In many systems, the user is able to refine a query based on the results returned by the system, and can then resubmit the modified query. This process repeats itself until the user is satisfied.

As can be seen in Figure 2.2, a generic IR system consists of a number of components:

1. A document store, often referred to as the document collection, where documents searchable by the system reside.
2. A query formulation subsystem, which allows users to pose queries to the system.
3. An indexing component, responsible for accepting raw documents and converting them into a form usable (i.e. searchable) by the system. Documents can only be added to the document collection by

going through this component.

4. A processing component which operates on query and document representations, deciding what to return in response to a query, and in what order.
5. An output component, which displays the results of the user's query. This component is usually closely linked with the query formulation system so as to enable the user to further operate on the results of a query.

The process of information retrieval can operate on one of two broad types of information sources: structured and unstructured. Structured sources are exemplified by records within a database and standardised forms, where certain information must appear in specific locations. Unstructured sources are common in most fields, with examples including books, articles, and most other informal means of human communication. Performing retrieval on structured data sources is usually a simple task. A search is executed for the contents of a combination of predefined fields, for example, retrieving records from a database based on the ranges of a certain field.

The focus of much research in information retrieval is the retrieval of documents from unstructured data sources. Intuitively, this is a much harder problem than structured document retrieval. The difference between structured and unstructured sources is not as well defined as one may initially believe. Many electronic documents come with a variety of (structured) meta-data, such as author name, title, keywords and the like, which may be searched in a structured manner. The contents of the document (and perhaps some of the meta-data) must still usually be treated as an unstructured data source.

The study of information retrieval is a very large field, and includes research into the retrieval of non-text and hypertext documents, such as video, still images, web pages and audio (see for example, [5] and [7]). Within the context of this report however, we only look at the retrieval of fully unstructured single-language text files.

This section begins by examining a number of different approaches to deciding which documents should be returned by the IR system. It then proceeds to examine methods for evaluating the effectiveness of IR systems, and concludes by looking at techniques that can be used to improve the efficiency of the basic IR process.

2.2.1 Simple text searches

Probably the simplest approach to IR is the full text search. A user queries the system by providing a text string. All documents in the collection are searched for this string, and any documents containing the text fragment are returned to the user. Extensions of this basic approach include searching for regular expressions rather than simple strings, and providing support for Boolean connectives when performing string matching (e.g. searching for documents containing the phrase “the quick”, but excluding from the results any documents with the phrase “brown fox”). Algorithms that can cater for spelling mistakes within user queries have also been developed [14], enhancing the robustness of this approach.

10 CHAPTER 2. BACKGROUND

Text scanning, while very straightforward, is sufficient for many applications. Furthermore, most text scanning implementations provide for very quick updates of the collection, and require very little space overheads over the space used to store the document collection. Unfortunately, these advantages come at the cost of relatively expensive (linear time complexity) searches [14].

A number of methods have been proposed to overcome the costs of searching for text in flat files. These approaches also provide a number of advantages when used in conjunction with more advanced information retrieval techniques. One very common approach is the inverse file approach. Here, a document is stored as a sorted list of keywords, together with the position of the words within the document. Determining whether (and where) a word exists in a document simply involves checking whether the word occurs within the list.

2.2.2 Boolean and probabilistic approaches to IR

Boolean IR

The Boolean approach to IR consists of a very simple, intuitive framework which is based on set theory and Boolean algebra. A further advantage of the Boolean model is the fact that queries are made up of Boolean expressions, providing very precise semantics.

A user query is made up of terms, combined using the Boolean relations *and*, *or* and *not*. A document is modelled by a binary vector. This vector's size is the same as the number of unique words in the document collection, and is indexed according to these unique words. The i th vector term is set to 1 if the i th unique word is present in the document, and 0 otherwise.

Determining whether a document is relevant to a query is done by computing a similarity measure between the two. This similarity measure returns a 1 if a document is believed relevant to the posed query, and a 0 otherwise. The similarity measure is computed by converting the query into disjunctive normal form, and then checking whether any of the conjunctive components of the query are present in any of the documents. If this occurs, the document is deemed relevant for the query.

Unfortunately, the approach is limited in a number of ways. First, retrieval decisions made using this approach are binary: documents are either relevant to the query, or irrelevant, with no grading scale; this severely limits this model's performance. Also, it is often difficult for users to formulate their requests into Boolean expressions. Thus, the queries posed to the system are often simplistic, and return to the user either too much, or incorrect information. Even with these drawbacks, the simplicity of the Boolean model means that it has seen much use in many commercial IR systems.

Probabilistic IR

The probabilistic model of IR, first introduced by Robertson and Sparck Jones [38], attempts to determine the probability of a document within the collection being of interest to the user based on the user's query.

As in the Boolean approach, a similarity measure is used to rank document usefulness. This similarity measure is defined as the ratio between the probability that a document is relevant to the query, and the

probability that it is non-relevant. In other words,

$$\text{sim}(d_j, q) = \frac{P(R|d_j)}{P(\bar{R}|d_j)}$$

where $P(R|d_j)$ represents the probability of relevance given document d_j , and q is the query.

Utilising Bayes' rule, and noting that the prior probabilities for relevance and irrelevance are identical for all documents in the collection, it can be seen that

$$\text{sim}(d_j, q) \sim \frac{P(d_j|R)}{P(d_j|\bar{R})}$$

Assuming that the presence or absence of a term is the only indicator of relevance, as well as assuming the independence of index terms, one can rewrite $P(d_j|R)$ as

$$\prod_{k_i \text{ appears}} P(k_i|R) \times \prod_{k_i \text{ does not appear}} P(\bar{k}_i|R)$$

Each k_i represents an index term appearing in the query. The above equation can therefore be interpreted as the product of the probability that the term appears in a document given that it is relevant, and the probability that it does not.

Van Rijsbergen shows in his book [37] that the similarity calculation can be rewritten as

$$\text{sim}(d_j, q) \sim \sum_i^t w_{i,q} \times w_{i,j} \times \left(\log \frac{P(k_i|R)}{1 - P(k_i|R)} + \log \frac{1 - P(k_i|\bar{R})}{P(k_i|\bar{R})} \right)$$

where $w_{i,q}$ and $w_{i,j}$ are weights set to 1 if the term indexed by i appears in the query and document respectively, and 0 otherwise.

Clearly, obtaining accurate estimates for $P(k_i|R)$ and $P(k_i|\bar{R})$ is critical for the accurate functioning of probabilistic information retrieval schemes. Before the system is used, $P(k_i|R)$ for all k_i is usually set to the same value (commonly 0.5), and the approximation is made that the distribution of terms within non-relevant documents is equal to the distribution of these terms within all documents in the collection. This assumption can be written as $\frac{P(k_i|\bar{R})=n_i}{N}$, where n_i is the number of documents containing the term k_i , and N is the total number of documents in the collection.

Once queries are posed to the system, it is possible to refine these estimates: $P(k_i|R)$ can be set to the number of retrieved documents containing the term ($|r_i|$) scaled by the total number of retrieved documents ($|r|$), and $P(k_i|\bar{R})$ can be computed by the function $\frac{n_i - |r_i|}{N - |r|}$.

The estimates can also be improved with the user actively indicating which documents are, or are not relevant to the posed query. This process is a form of relevance feedback, and is discussed in more detail further on.

The similarity value can be used to decide which documents should be returned. A cut-off similarity can be specified, with documents less similar than this value being deemed irrelevant, and documents above the value being returned.

The similarity value can also be used to rank documents according to the probability of relevance, with more similar documents appearing earlier in the returned document list.

The probabilistic model described above is only one of a number of suggested probabilistic approaches. Van Rijsbergen [37] provides one approach to circumventing the assumption of term independence. Fuhr [16] utilises an approach that creates description vectors from a document-query pair, after which a regression method is used to obtain an estimate of the probability of relevance. This scheme has some obvious parallels with machine learning as creating the regression model requires a “training” document set. Machine learning in IR is discussed in Section 2.2.7. More advanced probabilistic IR techniques are discussed in Section 2.2.4.

2.2.3 The vector model of IR

With the parallel growth in the world wide web and its search engines, vector-based techniques have become the dominant way in which to retrieve online information.

The vector approach [40] shares many aspects with the methods described above. Each document is represented as a vector, with element i within the vector taking on a value (usually between 0 and 1) based on the presence or absence of the word indexed by i within a global word list. Queries are represented in a similar manner. Determining whether a document is relevant for a given query involves computing a similarity measure between the document and query. Most commonly, this is done by calculating the cosine of the angle between the two vectors. The resulting value, ranging between 0 and 1, is normally used to rank the documents by believed relevance, with higher ranked documents appearing before lower ranked documents. A retrieval value threshold is commonly set, if a document-query similarity value is below this threshold, the document is deemed irrelevant, and is not retrieved.

The *tf.idf* family of schemes

One of the earliest and most popular ways with which to create weighting vectors is the *tf.idf* family of weighting schemes. The term frequency component (*tf*) of a term t_i for a document d_j is calculated according to

$$tf_{i,j} = \frac{\text{frequency}_{i,j}}{\max_{l \in \text{terms}} \text{frequency}_{l,j}} \quad (2.1)$$

i.e. the raw frequency of a term divided by the frequency of the most common term in the document.

The *idf*, or inter-document frequency component is normally computed as follows:

$$idf_i = \log \frac{N}{n_i}$$

where N is the total number of documents in the collection, and n_i is the number of documents in which the term t_i appears.

In *tf.idf* weighting schemes, the component of the weighting vector for document d_j at position i (i.e.

for term t_i) is of the form

$$w_{i,j} = tf_{i,j} \times idf_i$$

A number of variations have been proposed to this basic formula. Salton and Buckley [41] have experimented with a number of these variations, and found that the basic formula is sufficient for most collections.

The query term weights are often calculated in a different manner to document term weights. Salton and Buckley recommended using the formula

$$w_{i,q} = idf_i \times (0.5 + 0.5 \times tf_{i,q}) \quad (2.2)$$

where the term frequencies are computed over the query rather than over the original collection. Inverse document frequencies for the query weighting vector are still computed over the entire document collection. The additional terms in the equation modify the influence of the idf and tf components.

One very interesting result from Salton and Buckley's paper is that different weighting vector schemes perform better or worse on collections and queries with different properties. For example, when short queries are most common (such as in a web search engine environment), query weighting vectors should be modified to further increase the influence of the term frequency component in Equation 2.2. Similarly, retrieval of documents in collections with highly varied vocabularies seem to benefit from an increased tf weighting. This result indicates that some sort of "adaptive" information retrieval algorithm, which modifies itself to perform better on a specific collection, may be worth pursuing. The research described in this report suggests one approach to creating an adaptive information retrieval algorithm.

2.2.4 Other IR models

The boolean, probabilistic and vector models described previously are often referred to as the "classic" models of information retrieval [5]. Other Boolean, vector, and probabilistic techniques do exist, many of which bear only a passing resemblance to those previously described. In this section, some of the more popular and novel of these techniques are discussed.

Other set theoretic techniques for information retrieval

Within the taxonomy of information retrieval methods, the Boolean model can be classified under set theoretic information retrieval approaches [5]. As described in an earlier section, the basic model, while simple, has a number of drawbacks. The approaches described in this section build on the methodology, attempting to overcome these problems.

Extending the boolean approach The Boolean approach uses keywords combined with Boolean connectives to specify a query. Wondergem et al. [46] have proposed an extension to this which allows a user to pose queries using a combination of Boolean connectives and noun phrases, for example "(cycling \vee hiking) in Holland".

Another way of extending the Boolean model was proposed by Salton, Fox and Wu [42]. Their extended boolean model computes the similarity between a user query and a document in a different manner to the standard Boolean method. Unlike the standard Boolean model, this similarity measure is not binary, and can therefore provide a ranking by believed relevance for the retrieved documents. While this extended approach borrows some ideas from the vector based approach to IR, it retains the simplicity, elegance and theoretical underpinnings of the Boolean model.

Fuzzy information retrieval The Boolean approach can be viewed as set theoretic as it attempts to find all documents with keywords belonging to the set described by the user query. The fuzzy set model as described by Ogawa et al. [32] extends this concept by assuming that each query term describes a fuzzy set, and each document has a degree of membership within each query term set. A query set is obtained by combining the term sets into a final set using fuzzy Boolean operators. Retrieved documents can be ordered by their degree of membership within the final set.

Other algebraic models

The taxonomy described by Baeza-Yates et al. [5] places vector-based information retrieval amongst a number of other algebraic IR models. Standard vector-based retrieval assumes that document index terms are orthogonal (and therefore independent). Clearly this is a bad assumption. Wong, Ziakro, and Wong [47] propose an approach which circumvents this assumption.

Index term vectors are vectors with a 1 at the position of the index of the term, and 0 elsewhere. Standard vector IR techniques represent document j as the following vector:

$$D_j = \sum_i w_{i,j} t_i$$

where t_i is the index term vector, D_j is the j -th document in the collection, and $w_{i,j}$ is a weighting value for word i within the document. Thus, each index term produces a vector, the sum of which creates a vector consisting of weights for a document based on all words within the collection.

The generalised vector approach no longer assumes that the index term vectors are defined as described in the previous paragraph. Instead, the vectors are composed of smaller components, called minterms, which are orthogonal with respect to each other. By composing the term vectors using minterms, correlations between the various index terms can be represented.

Performing information retrieval using index terms, even when compensating for the effects of correlations between words, is inherently flawed: commonly, when posing a query, the user is not looking for documents containing precisely those (or related) terms, but rather, they are attempting to find documents which cover a concept described by their query. One approach that attempts to retrieve documents based on their “latent semantics” is Latent Semantic Indexing [12]. The main idea behind LSI involves approximating the high-dimension document and query vectors into a lower dimensional space. Searching then takes place in the reduced space, with the hope that this space better represents concepts than the original.

Singular value decomposition is normally used to perform the dimensionality reduction, though Hofmann has suggested a probabilistic approach to LSI [22].

One other common approach classified under algebraic techniques is neural network based information retrieval. This technique is covered in Section 2.2.7.

Other probabilistic approaches

The highly successful INQUERY system [9] utilises a probabilistic technique based on Bayesian networks [34]. Briefly, a Bayesian network is a directed acyclic graph, with nodes representing events and edges indicating dependencies between events. Conditional probabilities are associated with each edge. In INQUERY the basic shape of the network is predetermined, but links, as well as link probabilities, are modified based on the documents in the collection. A user query is represented as a set of true and false nodes within the system. By performing recursive inference, it is possible to return a list of documents ordered by their probability of relevance to the user's query.

Miller et al. [29] reported on an IR approach utilising hidden Markov Models. A Markov Model is created to model the generation of a query by the user based on the set of documents the user wants to retrieve. The complexity of the model can be altered based on the amount of information available regarding the query generation/document representation process, as well as the computational resources that can be expended to answer the query. This approach shares some common ground with language modelling approaches.

According to Ponte and Croft, a language model refers to a probability distribution that captures the statistical regularities of a document [35]. Under the language model approach, a language model is computed for each document in the collection. When a query is posed to the system, the probability of generating the query is calculated according to each document's language model. The documents are then returned ranked by decreasing probabilities. Some of the deficiencies apparent in Ponte's original paper have been repaired in a later paper by Song and Croft [44]. The language modelling technique is currently amongst the most promising approaches for IR, and is undergoing investigation by a large number of researchers [28], [26], [21].

Hiemstra and de Vries [20] have shown that language model approaches, under a number of restrictions, can be viewed as a subset of the vector based information retrieval paradigm.

Other techniques

Croft and Lewis [11] noted that the information retrieval task can naturally be reclassified as a natural language processing problem: a user's query is a natural language representation of an information request. Documents that are relevant to the query are relevant to the semantics of the request, rather than its syntactic content. They thus attempted to use natural language processing techniques on both documents and requests. Due to the computational complexity of natural language techniques, very little follow up work using this approach appears to have been done. It should be noted that at some level, most IR techniques attempt to match the semantic content of a request with the semantics of a document. For example, while vector

10 CHAPTER 2. BACKGROUND

approaches deal only with frequencies of tokens in documents and queries, they only work due to the implicit link between these frequencies and the semantics of the documents and queries.

While many other IR techniques have been proposed, they are not discussed here. One area that must be examined in further detail is the application of machine learning techniques, and especially evolutionary techniques, to information retrieval. Many machine learning techniques have been utilised to aid rather than replace existing IR methods. Thus, before examining the role of machine learning in information retrieval, this report looks at methods for evaluating an IR system, as well as at a number of ways of improving the results of standard information retrieval techniques.

2.2.5 Evaluation methods for IR systems

Two aspects of an IR system can be measured: efficiency, and effectiveness. Efficiency can be measured in terms of the resources required by the system, including the storage space required to store the document collection, and the computing resources needed to perform operations on the collection, such as the addition and removal of documents, and performing queries. While it is sometimes possible to compute the time and space complexity of the various aspects of the system, more concrete efficiency measures are often required when implementing a concrete system, but these measurements are difficult to perform in a machine independent manner.

Effectiveness attempts to measure, as the name implies, the effectiveness of an IR system at satisfying a set of queries. Given a sufficiently general document and query collection, the effectiveness should provide a domain neutral measure of the system's ability to satisfy user queries. The measurement of effectiveness is further complicated by the fact that it is dependent on the type of task being evaluated. Interactive systems must be evaluated in a different way to systems in which user feedback plays no role.

Many IR researchers [37] believe that a satisfactory approach to the evaluation of an information retrieval system is yet to be found. Since this is still a rich and ongoing area of research, this report only looks at a few of the most popular evaluation methodologies.

Precision and recall

Probably the most widespread method of evaluating an IR system involves plotting its precision–recall diagram for a set of queries posed on a specific document collection.

Given a document collection and a query, let R be the number of relevant documents in the set for this query, and A be the number of documents retrieved by the IR system. Finally, let I be the number of relevant documents within the retrieved document set. Recall and precision can then be defined as:

$$\text{Recall} = \frac{I}{R}$$

$$\text{Precision} = \frac{I}{A}$$

When defined like this, recall and precision range between 0 and 1. In many cases, these values are

multiplied by 100 to yield a percentage value.

The basic precision and recall measures assume that the IR system returns an unsorted list of results, which is then evaluated in full. If this is not the case, recall and precision values change as more documents within the result list are examined (the assumption is made that the list is ordered from the highest to least believed relevance). This is done by recomputing precision over the seen documents whenever a relevant document is found in the retrieved document list. For example, assume the following documents are retrieved in response to a query, with an asterisk indicating relevant documents:

rank	document number	relevant
1	d_{23}	
2	d_{42}	*
3	d_{12}	*
4	d_1	
5	d_7	
6	d_{100}	*
7	d_{43}	
8	d_{67}	
9	d_{94}	*
10	d_{10}	

Assume that this query has five relevant documents. At a recall level of 20% (i.e. one out of the five relevant documents have been seen), precision is 50%, since one out of the two seen documents are relevant. At the 40% recall level, precision increases to 66%. At 60% and 80% recall, precision values are 50% and 44%. Since no relevant documents are retrieved after the ninth document, and relevant documents still exist that should have been retrieved, precision drops to 0% at the 90% recall level.

Precision-recall curves are normally drawn by computing precision at 11 standard recall values, namely, 0%, 10%, 20%, ..., and 100%. If, as in the above example, insufficient relevant documents exist to compute recall at all these points, the values at the standard points are set as the maximum known precision at any known recall points between the current and next standard points. The example presented above would therefore yield the precision-recall curve illustrated in Figure 2.3.

Usually, precision-recall curves are computed by averaging the precisions obtained at the standard recall values over all queries posed to the system.

When averaged over a number of queries, precision-recall curves tend to follow an exponential decay curve. Intuitively this results from the fact that an algorithm would generally rank at least a few relevant documents quite highly, therefore yielding high precision for low recall values. As the number of relevant documents returned by the system increases however, more and more irrelevant documents are returned within the results. Obviously, a perfect algorithm would have a flat precision-recall curve at the 100% level.

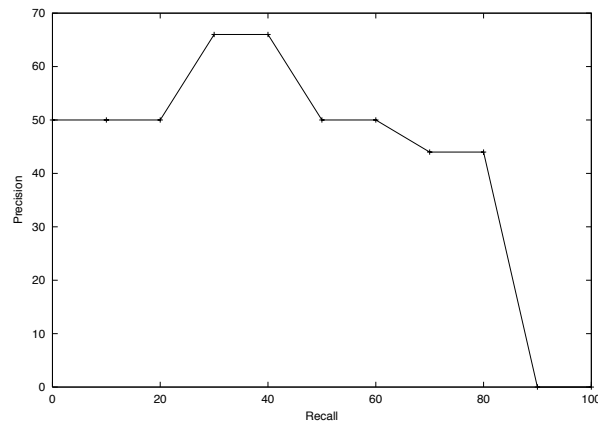


Figure 2.3: A sample precision-recall diagram.

Single value measures

While precision-recall figures are often useful, there are many situations in which a single figure is needed as a system performance measure. Baeza-Yates and Ribeiro-Neto [5] provide a number of measures appropriate for situations when one wishes to evaluate the performance of an algorithm over a single query. These measures include:

- Average precision at seen relevant documents, which involves averaging the precision figures obtained each time a new relevant document is seen.
- R-precision, where R is the number of relevant documents for the query, computes the precision after R documents have been seen (e.g. with 10 relevant documents for a query, the R-precision would return the precision after the first 10 documents have been retrieved by the system).

Another set of single value measures can be used to summarise the performance of a system on a set of queries. The harmonic mean, calculated as

$$F(j) = \frac{2}{\frac{1}{R(j)} + \frac{1}{P(j)}}$$

combines precision and recall to yield a single value for the j -th document within a ranking. $P(j)$ is the precision for this document, while $R(j)$ is the recall. This measure can be used to find an effective cutoff for the returned document list, as it is highest when recall and precision are both high.

Other methods

While precision and recall are common measures of the ability of an IR system, a number of criticisms have been levelled against the scheme [5]:

- For a single query, two users may deem different sets of documents as relevant.

- Recall is hard to measure for large document collections.
- It is difficult to compute the curve for systems in which no ordering of returned documents is made (or in which the ordering is not based on believed relevance).
- Defining precision and recall in batch style IR settings is simple. When interactive IR systems are used, the definition of a retrieved document becomes more blurred. This blurring clearly extends to the system's precision and recall values.

User-oriented measures [5] attempt to provide an effectiveness measure for IR systems while taking these factors into account. These measures include the coverage ratio, defined as $\frac{|R_k|}{|U|}$ where R_k consists of those relevant documents of which the user was previously aware, and U being the complete set of relevant documents that the user knows are in the collection. Another useful measure, called the novelty ratio, is defined as the fraction of relevant documents retrieved of which the user was previously unaware. A number of other user-centric measures can also be defined. The essence of all these metrics is that they are centred around the user's expectations of the system, and thus provide some insight into how effective the system is in answering the user's information needs.

Another, very different approach to evaluating an IR system was proposed by Hersh et al. [19]. They propose testing the effectiveness of an IR system by examining how well it meets users' information needs. This is achieved by asking the users a number of questions, and then letting them attempt to answer some of these questions (the ones which they were most uncertain about) with help from the IR system. By comparing the user's scores before and after using the IR system, a measurement for its utility can be obtained. The measure obtained from this approach appears to provide a very good indication as to how effective the system is in meeting a user's information requirements. This approach does however suffer to some extent in that it measures not only the IR system's retrieval ability, but also the manner in which it interacts with the user. This measure may thus yield results biased by the users used in evaluating the system.

While these other approaches to IR system evaluation do overcome many of the criticisms levelled at precision-recall based approaches, they remain unpopular. Most of this unpopularity stems from the increased amount of subjectivity required to compute these other metrics. Furthermore, while generating relevance judgements for large collections is not trivial, the extra amount of effort required to use the other evaluation methodologies makes them prohibitively work intensive for anything but the smallest collections.

Comparing the effectiveness of IR algorithms

To determine whether one IR approach is better than another requires more than comparing their precision and recall values. Precision and recall results obtained by the same technique on different datasets may vary by large amounts – changes in precision of more than 20% at certain recall values can be found in research papers [48]. Computing the average precision value is no better — Xu found that the average precision value for *tf.idf* ranged between approximately 32% and 54% between two datasets [48].

To overcome this problem, researchers commonly use one of the following solutions:

1. Evaluation of a technique is performed on a well known dataset, for which comparable results for many other approaches exist. One of the TREC datasets is often used in this role.
2. A benchmark technique is run on the same dataset as the technique to be evaluated, and results are then provided for both methods. Probably the most common benchmark method is the standard *tf.idf* algorithm, due to its simplicity and ease of implementation. This is the approach taken by the research described in this document.

2.2.6 Improving IR performance

The techniques presented in the previous section made little, if any, use of any information found in the collection or query other than the presence or absence of specific tokens. This section examines a number of ideas that can improve the effectiveness of IR by making use of more information.

Relevance feedback

Relevance feedback involves resubmitting a modified query to the IR system. This modified query is generated from the results of the user's original query. "Manual" relevance feedback [40] involves the user tagging the list of documents returned by his or her original query as relevant (or irrelevant). Similarities between the documents deemed most relevant by the user are then used to form the modified query. While manual relevance feedback for large collections may at first appear impractical due to the large number of documents that the user must judge, it has been found that using only the first few returned documents is sufficient to bring about large improvements in final retrieval relevance [8].

When using a term based IR technique, the two most common methods in which relevance feedback is implemented involve query expansion and term reweighting. Query expansion modifies the original query by adding extra terms based on terms found in the relevant documents. Term reweighting consists of changing the weights of the terms in the query vector, based on the original query vector's interaction with the relevant document vectors.

It is often undesirable for a user to perform manual relevance feedback when performing a query. Automatic relevance feedback methods have been developed to improve the quality of the returned document set while reducing or eliminating user intervention. These techniques can be subdivided into two categories: local analysis and global analysis methods.

Local analysis techniques

The first method, consisting of local analysis techniques [48], is similar to relevance feedback: an initial returned set of documents (the local set) is used to determine which additional documents should be returned to the user. An IR system using local analysis methods requires no user interaction between query execution and the displaying of results.

Local analysis operates by looking at the highest ranked retrieved documents, and constructing a list of words that co-occur most frequently within these documents. This word list is then added to the user query,

after which it is resubmitted to the IR system. Local analysis operates on the assumption that words that commonly occur in relevant documents should be useful when searching for further relevant texts.

Local analysis does not severely impact the speed of an IR system, requiring only one extra access to the document collection, together with an additional search request. However, local analysis techniques suffer when a query retrieves only a few relevant documents, as well as when the documents retrieved by the initial query are not relevant to the user's needs.

Global analysis methods and thesauri

Global analysis methods utilise information shared between all documents in the collection (the global set) to determine which terms should be added to the user's original query. Some systems allow the user to select which additional terms should be added to a query, while others automatically retrieve results using the additional terms.

Global analysis techniques make use of a thesaurus (abstractly defined as "a set of items and a set of relations between these items" [24]). Manually created thesauri are not commonly used due to the expenses involved in their building and maintenance. Instead, researchers have investigated various methods to automatically generate thesauri from a document collection. While global analysis approaches yield large improvements to the quality of search results, the computational power required to generate a thesaurus limits their use to small, static document collections.

Xu and Croft [48] have proposed an approach known as local context analysis. This approach combines features from both global and local analysis techniques, and has shown good results.

2.2.7 Applications of machine learning techniques to IR

The field of machine learning (ML) deals with techniques whereby an algorithm is able to induce knowledge from data. Most machine learning approaches require some form of initialisation before working. This initialisation usually consists of providing the algorithm with training data. Broadly, ML algorithms can be classified into one of two categories depending on the type of training data they use. Supervised learning algorithms require the trainer to inform the algorithm what type of output is expected for each datum, while unsupervised learning algorithms attempt to differentiate between various data classes with no outside intervention.

Classification problems are a very common target for machine learning strategies. These types of problems involve determining to which of a set of classes an input belongs. IR can be viewed as a classification problem: given a query, the system is required to classify documents into one of two classes: relevant or irrelevant. It should therefore not be surprising that researchers have investigated the application of ML techniques to IR for some time. Many of these approaches have shown promise, and this section examines a number of existing ML techniques, as used within an IR system. The section concludes by examining previous research involving evolutionary techniques as applied to information retrieval.

Neural networks have been used extensively in various areas of IR. A neural network is an idealised model of brain functioning. It consists of nodes (called neurons) connected by directed edges, with each

edge having a numeric “weight”. Certain nodes are deemed input nodes, and are assigned numeric values. Nodes connected to these input nodes are then assigned values based on the edge weights, the values stored in the input nodes, and a combination function. This process is then repeated until all nodes in the network have values. The values of certain neurons, referred to as output neurons, are used as the output of the network.

The neural network model [45] of IR consists of a three layer network. The input layer contains the same number of neurons as there are possible query terms. An input neuron is set to 1 if the term is present in the query, and 0 otherwise. These values are then propagated into the middle layer, which consists of all possible document terms. Once the document term neuron values have been computed, the third layer’s neuron values are calculated. This third layer represents individual documents. Thus, the third layer contains the same number of neurons as there are documents in the collection. Once the third layer’s values are computed, processing does not stop. The third layer feeds its values back to the second layer, and a feedback loop is thus initiated. With each cycle, values are fed to the next layer only if they are above a certain threshold. Thus, as the cycles progress, fewer values are updated, until an equilibrium is achieved. Once this state is reached, the third layer indicates the documents to be retrieved, with relevance ordered by neuron value. Training the neural network on a specific document collection involves generating the correct weights for inter-neuron connections, as well as creating the correct number of neurons for the specific collection. The neural network model has not been tested on large collections, and its general performance as compared to other approaches is still in question [5].

ML techniques have often been used to perform relevance–feedback type tasks rather than as an IR model. Gordon [18] describes a scheme which utilises genetic algorithms to alter keywords associated with various documents as the user interacts with the system. By doing this, more accurate keywords can be assigned to each document, which can then be matched with keywords in user queries so as to provide improved performance. Yang and Korfhage [49] investigated a similar method. Instead of altering keywords, they used genetic algorithms to alter the weights assigned to various tokens. As the population evolved, the accuracy of the keyword weightings in representing the topic of a document in the collection improved.

Gordon, together with a number of collaborators, has continued to investigate evolutionary techniques for information retrieval. Pathak et al. [33] investigated using genetic algorithms to choose between a number of similarity functions. It was believed that since different similarity functions appear to greatly change the quality of results for different collections and query sets, it would be possible to search through the space of matching functions and find an optimal combination of these for a specific collection.

Fan et al. have recently started to investigate the effectiveness of an approach similar to the one proposed in this research [15]. Briefly, genetic programming is used to generate a population of weighting vector generators. By evolving this population they attempted to create individuals well suited to answering specific queries. The research described here, while proceeding in a similar manner attempts to achieve a different set of goals: first, I try to evolve individuals that operate well for all queries posed to a specific document collection, and secondly, try to determine if it is possible to evolve a weighting function that performs well for general collections.

Apart from the research mentioned previously, very little work appears to have been done on the use of

evolutionary techniques in IR. One possible reason for this is that until recently, these techniques were too computationally intensive to see much use. With the continuing improvements in processor performance, more and more applications of evolutionary machine learning techniques for IR should appear in the future.

2.3 Context

The volume of research done in the area of IR is huge. This aim of this chapter was not to provide an exhaustive overview of the field. Instead, only the most popular algorithms were examined.

Amongst the methods covered, vector based IR continues to be one of the simplest and most popular approaches. The rest of this document looks at one way of modifying the basic *tf.idf* scheme to improve retrieval results. By modifying the most basic component of vector IR, one can still make use of other advances in the field such as relevance feedback with no modifications.

Apart from looking at a number of IR models, this chapter briefly examined Genetic Programming, as well as methods for evaluating IR systems.

The aim of this chapter was to provide the reader with a sufficient background to evaluate the remainder of this document within the context of existing research.

Chapter 3

Applying Genetic Programming to Information Retrieval

3.1 Introduction

The aim of this research is to investigate the application of genetic programming to the creation of *tf.idf* like evaluators which can be used in vector-based information retrieval. By utilising this approach, it is possible to create fine-tuned retrieval tools for specific document collections. Searching through the space of possible evaluation functions also brings up the possibility of finding an evaluator superior to *tf.idf* when working with arbitrary document collections.

This chapter continues by describing the research hypothesis. An outline of my algorithm is then given, after which the experimental setup is discussed in detail. The experiments themselves are then discussed. Finally, the evaluation criteria for each experiment are described.

3.2 Research question

This research focuses on using genetic programming methods to construct classifiers for use on collections of text documents, and on comparing the effectiveness of these classifiers with baseline existing methods. This research therefore attempts to answer the following question:

Can an indexing metric be produced using genetic programming methods which

- 1. has better recall and precision than existing indexing methods;*
- 2. operates well over a large number of domains; and*
- 3. requires the same order of magnitude of resources, such as computational time and space, for answering queries as existing methods?*

It was originally hoped that any new classifiers with improved performance could be analysed to obtain some insight into the theoretical underpinnings of vector-based IR. This did not happen as the evolved classifiers were too varied and complex to be easily understood.

3.3 Combining Genetic Programming with vector-based information retrieval

As described in Chapter 2, vector-based information retrieval associates a vector with each document in the collection. This vector can further be thought of as a combination of two components: an index term vector, consisting of a single 1 entry at the position indexed by the term and 0's elsewhere, and a weighting scalar whose value is based on the frequency of the term in the document and collection. The index term vector is multiplied by the scalar weighting component to obtain a final term vector. All term vectors are summed to obtain a final document representation vector.

The weighting component is normally calculated using some modified form of *tf.idf*. While some justification has been provided for this choice of weighting scheme, only empirical evidence exists for its effectiveness. This research proposes a method to generate new, more effective, weighting schemes.

Figure 3.1 on page 26 illustrates the basic algorithm used in this research.

Briefly, a number of weighting schemes are created and represented as programs. Each weighting scheme is tested on a subset of queries from the document collection. The precision and recall of each scheme is then combined to give a final fitness value. Using standard genetic-algorithm techniques, described in the previous chapter, fitter weighting schemes propagate to form the next generation. This procedure is repeatedly followed until a sufficient number of generations has been created.

3.4 Experimental setup

Experiments were run on the Cystic Fibrosis (CF) dataset with the CISI dataset [1] used in experiment 3, as described later. The CF dataset consists of 1239 documents and 200 queries, with relevance judgements by four experts indicating the degree to which each expert believed a document was relevant to each query. A sample document extract, relevance judgement and query are illustrated in Figure 3.2.

Document and query preprocessing

A number of actions occur when documents and queries are added to the system. Some of these are performed to improve the system's speed, while others are taken to reduce the amount of data that needs to be stored and processed.

1. Extract only the main content field from the document, and discard all other text.
2. Stop words, consisting of common words such as "the", "a" and the like, are removed. The list of stop words consisted of 319 words, and was taken from Van Rijsbergen's book [37].
3. An attempt is made to isolate English words by removing all words containing non-alphabetic characters.
4. The remaining words are stemmed according to Porter's stemming algorithm [36]. The stemming process takes a word and changes it to a common root; for example, stemming the words "combining" and "combined" would result in the word "combin".

```

program Training(Documents, Queries, RelevanceJudgements, maxGenerations)
  Programs:= a set of random programs and a single tf.idf program
  create Collection by preprocessing Documents, Queries, RelevanceJudgements
  generation:=0
  repeat until generation==maxGenerations
    generation++
    fitnesses:=EvaluateFitness(Programs,Collection)
    write Programs and fitnesses to disk
    create new Programs according to fitnesses
  end

fitnesses EvaluateFitness(Programs,Collection)
  returns a vector with the element at i being the fitness of program[i].
  GivenRelevance is an array describing the relevance judgments e.g.:
  GivenRelevance_12 is 1 if document 1 is relevant for query 2.

  for each p in Programs
    for each d in Documents within Collection
      t_pd:=EvaluateProgram(p,d)

    for each q in Queries within Collection
      q_pq:=EvaluateProgram(p,q)

    For each Document d and Query q
      Relevance_pdq:= t_pd . q_pq
      RelevanceList_pq:=List of documents for query q. program p
                          ordered by Relevance_pdq

    Compute precision and recall for p based on RelevanceList_pq and
    relevance judgments.

    fitness_p=avg precision OR precision at 50% recall depending on
    experiment.

vector EvaluateProgram(Program,Words)
  runs the genetic program Program on Words. Returns a vector with each entry
  between 0 and 1, with the entry at i corresponding to the weight of the word
  indexed at i by the Collection

```

Figure 3.1: Pseudo code for training algorithm

```

<RECORD>
<PAPERNUM>PN74002</PAPERNUM>
<CITATIONS>
<CITE num="1" author="IBY NSC" publication="AND J RESPIR DIS" d1= "56" d2="38 9" d3="5"/>
</CITATIONS>
<RECORDNUM>00002 </RECORDNUM>
<MEDLINENUM>74260154</MEDLINENUM>
<AUTHORS><AUTHOR>Rossiter-M-A</AUTHOR>
</AUTHORS>
<TITLE>Amylase content of mixed saliva in children.</TITLE>
<SOURCE>Acta-Paediatr-Scand. 1974 May. 63(3). P 389-92.</SOURCE>
<MAJORSUBJ><TOPIC>SALIVA: en</TOPIC><TOPIC>AMYLASES: me</TOPIC></MAJORSUBJ>
<MINORSUBJ><TOPIC>INFANT</TOPIC><TOPIC>CHILD-PRESCHOOL</TOPIC>
</MINORSUBJ>
<ABSTRACT>Salivary amylase levels were determined in normal subjects from
birth until adult life and in children with conditions sometimes
associated with low pancreatic amylase such as malnutrition, coeliac
disease and cystic fibrosis. Mixed saliva was collected under
carefully standardised conditions and amylase was measured by the
method of Dahlqvist. There was a wide scatter of values in the 84
normal subjects, but concentrations rose from very low levels at
birth to reach adult levels by the age of 6 months to 1 year.
Salivary amylase activity rose normally over ten weeks in one
premature infant fed milk by gastrostomy. Thirteen children with
coeliac disease and 9 children with cystic fibrosis mostly had
normal salivary amylase concentrations. Six out of 12 malnourished
children with jejunal villous atrophy of uncertain aetiology had low
levels which rose to normal as recovery began.</ABSTRACT>
<REFERENCES>
<CITE num="001" author="ANDERSEN DH" publication="J PEDIATR" d1="30" d2="564" d3="947"/>
</REFERENCES>
</RECORD>

<QUERY>
<QueryNumber>00001</QueryNumber>
<QueryText>What are the effects of calcium on the physical properties of mucus
from CF patients?
</QueryText>
<Results>00034</Results>
<Records><Item score="1222">139</Item><Item score="2211">151</Item>
<Item score=" 0001">166</Item><Item score="0001">311</Item><Item score="1010">
370</Item><Item score="0001">392</Item><Item score="0001">439</Item>
<Item score="0011">440</Item ><Item score="2122">441</Item>
</Records>
</QUERY>

```

Figure 3.2: A sample document extract and query. Note that the query contains relevance judgements. Both extracts have been cut to reduce space.

5. If the word has not been previously encountered, it is added to the word list. Otherwise, the collection and document/query wide word counts for that word are incremented appropriately.

The removal of stop words and non-alphabetic words, as well as the practice of stemming are standard in most IR systems. The words removed by such actions are normally those that would contribute very little to the IR process due to their frequency. Stemming also reduces the vocabulary, and allows the system to easily handle many related words.

To utilise a genetic programming environment, a number of parameters must be fixed:

- A *metric* to rate the fitness of the generated classifiers.
- *Terminal nodes* that can compute various values from the collection and its documents.
- *Operators* that can be manipulated.
- The manner in which *breeding operations* take place.

The metric

Precision and recall are common evaluators of an IR system, and can therefore be used to answer the first question posed in the research hypothesis. It has been shown that an inverse relationship normally exists between these two values [39]. This fact means that a simplistic fitness function, consisting of averaging the sum of precision and recall over all queries and documents, does not function as desired. As seen in Section 2.2.5, other, more complex single value measures have been proposed. Most however, were not considered, as they operate well when evaluated over one query only. The approach finally taken was to define the metric as the precision value obtained at a recall of 50%, averaged over the entire query collection.

As is shown in the results, the individuals computed using this metric perform badly. The reasons for this is discussed in Section 4.2.1. The poor results initially obtained led to fitness computation using a new metric. This metric was computed by averaging the precision of an individual at 11 standard recall points.

Terminals and operators

Many alternatives exist for terminal and operator nodes. Some obvious operators include the standard arithmetic functions, while obvious terminal nodes include term frequency and inter-document frequency. Other terminals, such as the distance between specific word pairs could also be used, and would be useful in handling word correlations, but were not included due to their computational overhead. Additional operators are discussed in Section 5.3.4

Table 3.1 describes the operators and terminals used in this research. These terminals and operators were chosen for their ability to generate a large number of *tf.idf*-like schemes, as well as for their low computational overhead.

Terminals	Description
i	Inverse document frequency
r	Raw term frequency
t	Term frequency as calculated in Equation 2.1
N	Number of documents in collection
n	Number of documents in which word appears
M	Term frequency of most common word
l	Document length
a	Average document length
c	A constant node
Operators	
+ - × / log √	Standard mathematical operations

Table 3.1: Nodes used to generate programs

Breeding operations

Before the environment is usable, parameters must be set for the creation of a new generation of individuals from a previous generation. Within the context of these experiments, this was done as follows:

1. The fittest 10% of individuals were copied to the new generation unchanged. This was done as some of the breeding operations were very likely to otherwise remove “good” existing solutions.
2. The rest of the individuals in the new generation were created using mutation, copying, and crossover. Selection of an individual for each of these operations was fitness based, with the chance of an individual being picked for the next generation being directly proportional to its fitness.

Table 3.2 shows the likelihood of selecting a specific breeding operation when creating new individuals. Note that two mutation operations exist, one of which replaces a single node with another node, while the other replaces the entire subtree starting at the selected random node with a new subtree. The tree replacement operation substitutes a new tree for the old tree. The latter two operations were used in an attempt to increase the space searched.

The specific breeding method, and the likelihoods chosen for the various breeding operations were chosen with very little theoretical basis. It was hoped that the values chosen would provide a balance between searching a broad section of the solution space and refining promising solutions. As discussed in Section 5.3.1 further investigation of the effects of different breeding parameters may be useful.

Other considerations

Before a run can occur, the number of individuals within a generation and the number of generations must be fixed. Unless otherwise stated, each generation contained 100 individuals, with one individual of the initial generation consisting of a standard *tf.idf* evaluator, and the rest created as described in Figure 3.3.

Operation	Likelihood
Copy	0.1
Crossover	0.35
Single Node Mutation	0.25
Subtree Mutation	0.2
Tree replacement	0.1

Table 3.2: Breeding operations together with the likelihood of their being used.

The number of generations evaluated differed between experiments. Reasons for this are discussed in the individual experiment descriptions.

Within the context of the experiments described in this report, the subset of queries and documents used for training and testing must also be selected.

3.5 The experiments

Three experiments were run in order to evaluate the effectiveness of our approach:

- Training on a subset of queries, with all documents present, after which evaluation was done on the remainder of the queries.
- Training with a subset of queries and documents, with evaluation on the remainder of the queries and the full document set.
- Training on a full document collection, using all queries, and evaluating the genetic programs on another document collection.

The remainder of this section examines each experiment in more detail. Finally, the data our experimental setup is able to gather is discussed.

3.5.1 Experiment 1: Evaluation using new queries

This experiment achieved two goals. First, by comparing the recall and precision of the resulting evaluators to *tf.idf*, we can answer our first research question, at least for a specialised domain. Second, it allows us to determine whether either of the two metrics under consideration is clearly superior to the other. If one metric can be shown to be better, the other does not need to be utilised when performing the remainder of the experiments.

In this experiment, 70% of the queries, as well as the entire document collection, were used in training, with the remaining 30% utilised during the evaluation phase. 150 generations were evaluated during each run of this experiment.

This experiment simulates a common real world situation where users pose queries to a static document collection. Positive results in this experiment would thus indicate at least some real world benefits when utilising our approach.

```

OperatorDepth:=3.
MinTerminalDepth:=2.
MaxTerminalDepth:=6.

function selectNode(depth)
  if (depth<OperatorDepth)
    return random operator.
  if (depth>=MaxTerminalDepth)
    return random terminal.
  return random (operator or terminal)
end.

function generateSubtree(depth)
  currentNode:=selectNode(depth)
  if currentNode has a possibly unlimited number of children,
    set the number of children to between 1 and 3
  for each child of currentNode
    generateTree(depth+1).
  return currentNode
end

function generateTree
  return generateSubTree(0).
end

```

Figure 3.3: Algorithm for initial individual creation

3.5.2 Experiment 2: Evaluation with new queries and documents

This experiment increased the complexity of the domain in which the system operated. It aimed to continue answering the research question targeted by the previous experiment. By running in a more complex environment, it was hoped that this experiment would provide more insight into the functioning of our approach.

As in the previous experiment, 70% of queries were used in the training phase, together with half of the documents. The remaining documents and queries were used in evaluating the experiment. Only 50 generations were evaluated during each experimental run due to the early appearance of overtraining.

This experiment simulated another commonly occurring real world situation where a document collection consisting of related documents has more documents added to it, after which queries are posed to the complete collection.

3.5.3 Experiment 3: Evaluation on a new document collection

This experiment's goal was to determine the ability of our approach to operate over general domains. This was achieved by examining whether training on one dataset can create evaluators that will operate well on a second (unrelated) dataset. It is argued that if one can perform well on an unrelated dataset, one can operate on most datasets, leading to good performance in a general domain.

The third experiment utilised the entire Cystic Fibrosis dataset for training. Evaluation for this experiment took place on the CISI dataset. As in the second experiment, 50 generations were evaluated during each experimental run.

3.5.4 Remaining research questions

The results of the experiments described above answer two of the three research questions: we can determine whether this approach is more effective than *tf.idf* in specific scenarios, as well as find out if the approach is effective for general collections. The answer to the third question, i.e. whether the amount of resources required for this approach is comparable to that required for *tf.idf*, is somewhat involved:

The result of a trained system is a single procedure that will produce weightings for a specific word. Since no conditional or looping constructs appear in the terminals and operations used to generate the procedure, the time complexity of the procedure is linear in the number of operators. The time complexity of computing the terminals is terminal dependent, but no worse than $O(m * n)$ per terminal where m is the number of documents in the collection, and n the number of unique tokens. The complexity of a *tf.idf* based system is identical.

Space complexity is linear with respect to the number of terminals, assuming that the value of each terminal is computed ahead of time and then cached.

From the above, it appears that the resources used in the answering of queries are identical to *tf.idf* based approaches. The only other aspect of the system that must be examined for resource usage is the training of the weighting scheme. Since this is a once-off cost, almost any time cost is acceptable here. The space costs for training are proportional to the number of individuals within a generation.

3.6 Evaluation criteria

The experimental setup used in this research provides a wealth of data each time an experiment is run:

- The queries used during training are recorded.
- For all generations, the list of individuals used is kept, together with information about the fitness ordering of these individuals within the generation.
- For each individual, the following is recorded:
 - Its total fitness for the queries posed to it.
 - The program it represents.
 - 11 standard precision–recall measurements over the posed queries and available documents.

After training is completed evaluation is performed on the fittest individuals within each generation, as described in the previous section. The results of evaluation are recorded in the same way as results obtained during training.

A graph of training fitness against the generation number, as well as evaluation fitness against the generation number can provide a rough indication of the results of an experiment. As stated previously, fitness represents only one point on the recall precision curve. A better way of evaluating an individual therefore involves comparing its recall-precision curve to that of a baseline measure (in this case, that of *tf.idf* evaluation).

The training and evaluation fitness graphs can be used to spot overtraining: initially, both evaluation and training fitness are expected to rise as the number of generations increase. When this trend ceases, overtraining has probably set in. Since evaluation fitness is not monotonic some judgement is required in determining when overtraining occurs, a momentary drop in evaluation fitness may occur due to chance. Cross-validation was not used to detect overtraining due to the small size of the dataset and the additional implementation complexity it would have introduced.

Chapter 4

Results and evaluation

4.1 Method of evaluation

Due to the random nature of genetic programming, each experiment was run at least three times. For each experiment, fitness vs. generation curves were generated for both training and evaluation data. Where necessary, precision–recall curves were plotted for specific individuals.

4.2 Results

4.2.1 Experiment 1: Evaluation using new queries

As described in the previous chapter, this experiment is used to determine whether the approach presented here is superior to simple *tf.idf* in the simplest of cases. The experiment also aims to determine whether one of the two metrics performs better than the other. This experiment was stopped after 150 generations had been evaluated.

Experimental observations made when running this experiment include:

- The maximum-fitness individual from each generation computed on the training data. This allows one to verify that the genetic programming method is functioning, as well as indicating that some improvement over *tf.idf* can be made.
- The average fitness for each generation can be examined to further ensure that the system is functioning as desired; a sharp initial rise is expected, after which average fitness should slowly (but not monotonically) rise.
- The fitness of the best individual from each generation on the testing data. Some benefit may be obtained by examining every individual from each generation.
- Precision–recall diagrams for *tf.idf* and any individuals of interest for both training and testing data. An examination of the precision–recall curves should allow one to determine whether one fitness function is preferable to another.

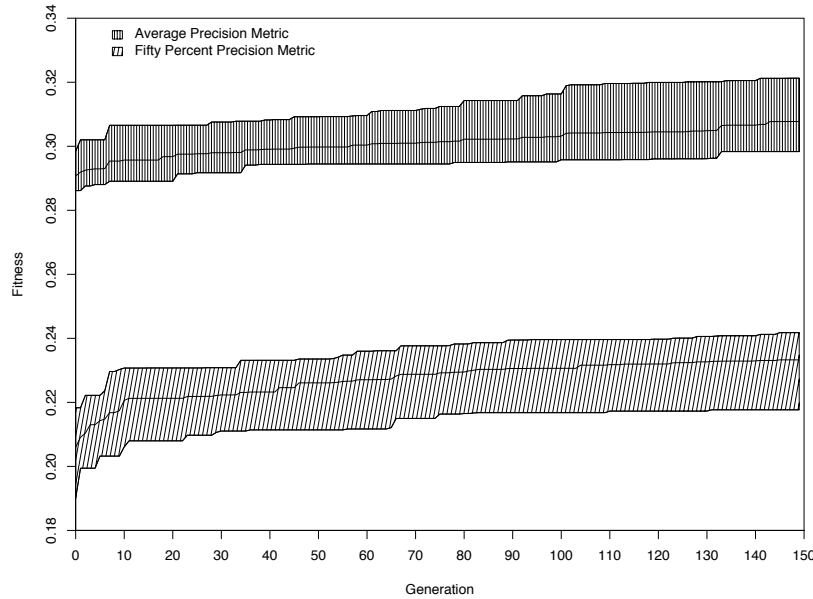


Figure 4.1: The maximum, mean and minimum fitness of the best individual within each generation during training, as evaluated using both fitness measures.

Figure 4.1 illustrates the improving fitness of the fittest individual during training for both fitness functions. *Tf.idf* is normally the fittest individual during the first generation, but evolved individuals soon overtake it. As expected, both fitness functions increase in a non-strictly monotonic manner as training progresses.

Figure 4.2 shows how the average fitness within each generation improves with training, together with an exponential curve fitted to the average fitnesses. Within the first generation, many of the generated individuals yield only nonsensical weights, giving them a fitness of zero, thus leading to a very poor average fitness. These individuals are quickly bred out, resulting in the dramatic improvements in average fitness for the first few generations. Once the system stabilises, average fitness rises very slowly over the course of a large number of generations.

Figure 4.3 shows the manner in which fitness changes against generation number when evaluated on test data. As expected, fitness increases (albeit not smoothly), for approximately 10 generations, after which a sharp drop occurs. This sharp drop is most probably caused by the appearance of overtraining.

One of the aims of this experiment was to determine whether one of the proposed fitness metrics would outperform the other. If one of the metrics was found to be superior, only it would be used in the rest of the experiments. It is clearly impossible to determine which of the two metrics is superior based on their respective fitnesses alone. One possible approach is to examine the percentage improvement in fitness of the individuals created with the metric. Another approach is to examine the precision–recall curves generated

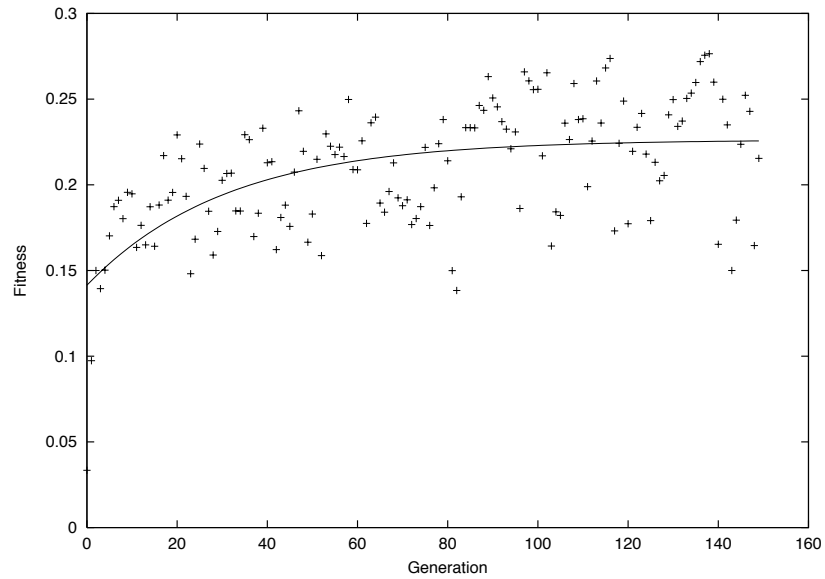


Figure 4.2: A typical average fitness curve obtained by computing the average fitness for each generation against the training data.

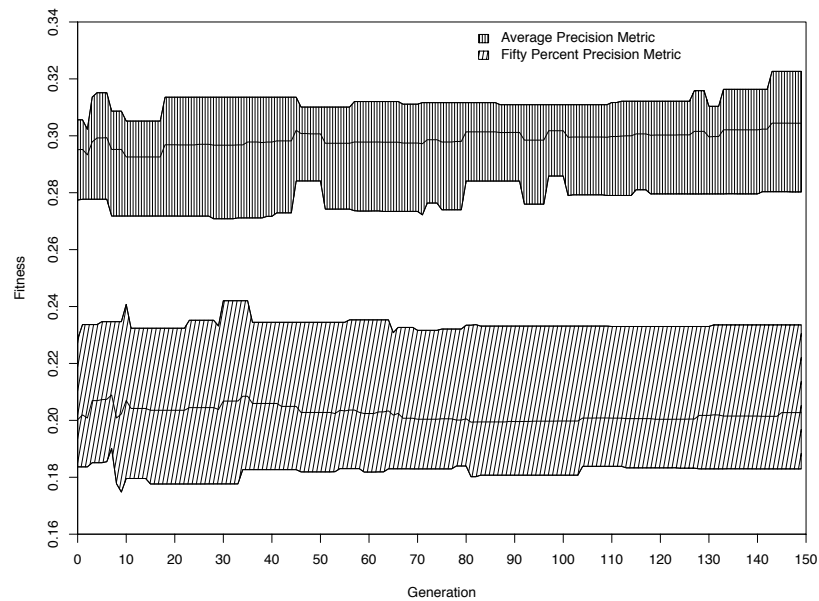


Figure 4.3: Testing fitness of the individuals deemed fittest for each generation during training, evaluated with both fitness metrics.

by the fittest members of a population. This latter approach was utilised here as one can directly relate its results to an improvement in retrieval performance.

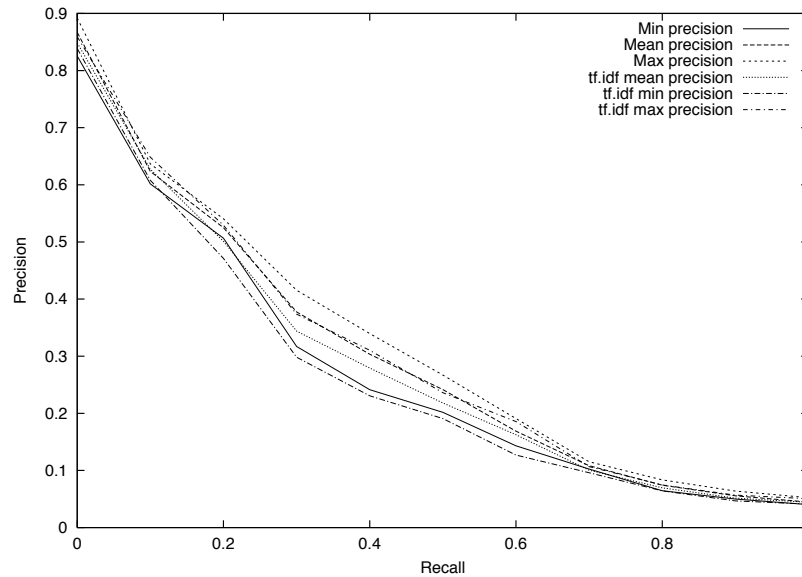


Figure 4.4: Minimum, mean and maximum precision–recall curves for the best individuals obtained using the average precision metric run on the testing dataset, as well as minimum, mean and maximum precision–recall curves for *tf.idf* run on the same test data.

Since testing was performed on different parts of the dataset during each run, precision–recall curves evaluating only testing data cannot be used to choose between metrics. These curves can however be used to provide some indication of the effectiveness of this approach in handling unseen data. Figure 4.4 compares the performance of the average precision metric to *tf.idf*, while Figure 4.5 does the same for the other metric. For clarity, these results are also shown in Table 4.1 and 4.2. These results indicate that the genetic programming approach can outperform basic *tf.idf*. Furthermore, these results hint at the possibility that the average precision metric outperforms the metric computed by using the 50% precision figure. Since different queries were used in each experimental run, further experiments were required to verify this result.

In order to better compare the two metrics, precision–recall curves can be generated by combining the test and training query sets. Precision–recall curves generated for this combined set allow for fair comparisons to be made between the two metrics. Since most of the contents of the combined set were utilised in training, comparisons of the results obtained here with those for *tf.idf* will not be representative of results that would be obtained in most real world scenarios. Figure 4.6 shows the performance of both fitness functions when evaluated over the combined dataset. For clarity, this is shown in tabular form in Table 4.3

This experiment did not attempt to investigate the relationship between the number of queries in the training set and the ability of the final evolved evaluators.

Figure 4.7 shows a simplified form of the best evolved generator. The form of the evolved evaluators in this and later experiments is further discussed in Section 4.4.

Recall	Average precision metric			<i>tf.idf</i>		
	Maximum	Mean	Minimum	Maximum	Mean	Minimum
0	0.891	0.866	0.825	0.859	0.848	0.838
0.1	0.636	0.623	0.601	0.648	0.627	0.607
0.2	0.540	0.524	0.506	0.529	0.501	0.471
0.3	0.415	0.377	0.316	0.373	0.343	0.298
0.4	0.339	0.303	0.241	0.310	0.279	0.230
0.5	0.266	0.241	0.201	0.236	0.218	0.190
0.6	0.190	0.168	0.143	0.185	0.162	0.127
0.7	0.115	0.108	0.102	0.106	0.100	0.096
0.8	0.083	0.074	0.064	0.074	0.069	0.064
0.9	0.063	0.055	0.049	0.056	0.051	0.046
1	0.052	0.044	0.040	0.050	0.044	0.040

Table 4.1: Minimum, mean and maximum precision values at standard recall values for the average precision metric and *tf.idf*, evaluated on the testing dataset.

Recall	Precision at 50% recall metric			<i>tf.idf</i>		
	Maximum	Mean	Minimum	Maximum	Mean	Minimum
0	0.825	0.818	0.804	0.855	0.849	0.845
0.1	0.665	0.612	0.571	0.692	0.641	0.607
0.2	0.599	0.532	0.492	0.576	0.515	0.471
0.3	0.375	0.333	0.288	0.371	0.333	0.298
0.4	0.302	0.258	0.219	0.303	0.260	0.230
0.5	0.243	0.213	0.191	0.229	0.201	0.184
0.6	0.185	0.160	0.132	0.171	0.150	0.127
0.7	0.129	0.111	0.096	0.121	0.102	0.090
0.8	0.080	0.074	0.066	0.081	0.073	0.064
0.9	0.059	0.054	0.048	0.057	0.053	0.047
1	0.049	0.045	0.037	0.050	0.045	0.037

Table 4.2: Minimum, mean and maximum precision values at standard recall values for the precision at 50% recall metric and *tf.idf*, evaluated on the testing dataset.

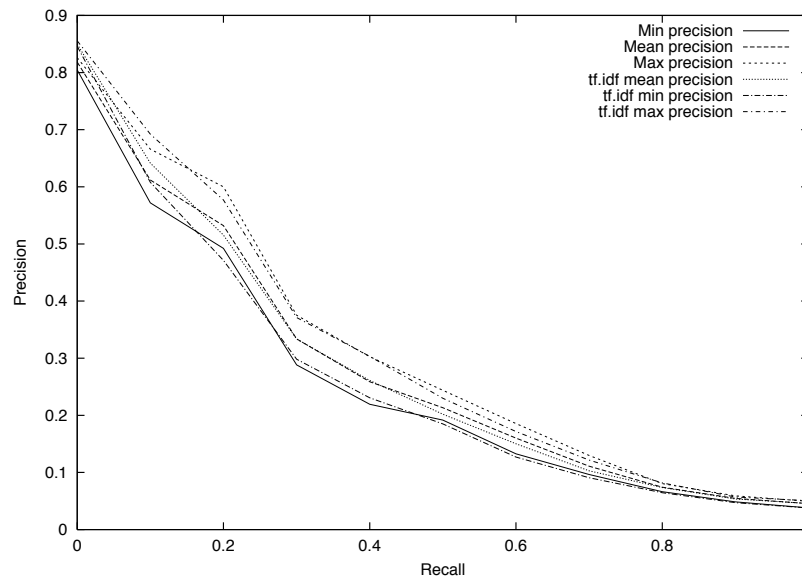


Figure 4.5: Minimum, mean and maximum precision–recall curves for the best individuals obtained using the precision at 50% recall metric run on the testing dataset, as well as minimum, mean and maximum precision–recall curves for *tf.idf* run on the same test data.

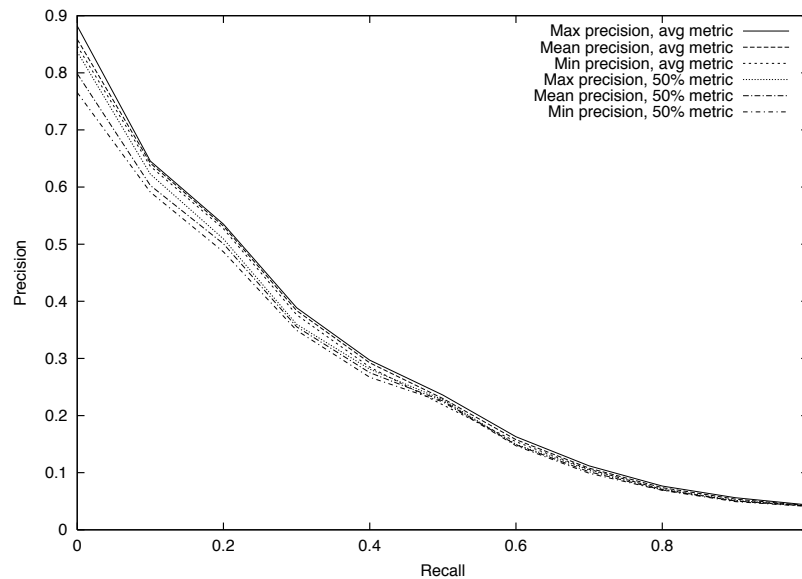


Figure 4.6: Precision–recall curves for both metrics, evaluated over the full query set.

Analysis

This experiment indicates that when evaluated over a static document set, genetic programming can yield evaluators that perform better than simple *tf.idf*. The best evaluators appear to emerge within 40 genera-

Recall	Average precision metric			Precision at 50% recall metric		
	Maximum	Mean	Minimum	Maximum	Mean	Minimum
0	0.881	0.858	0.847	0.836	0.798	0.765
0.1	0.645	0.641	0.636	0.622	0.602	0.591
0.2	0.535	0.531	0.527	0.509	0.500	0.486
0.3	0.388	0.383	0.376	0.359	0.355	0.349
0.4	0.296	0.292	0.284	0.281	0.275	0.266
0.5	0.236	0.229	0.218	0.227	0.225	0.224
0.6	0.162	0.157	0.154	0.149	0.148	0.147
0.7	0.112	0.108	0.105	0.104	0.101	0.098
0.8	0.076	0.073	0.070	0.071	0.069	0.069
0.9	0.056	0.053	0.050	0.051	0.050	0.049
1	0.043	0.042	0.041	0.041	0.041	0.040

Table 4.3: Minimum, mean and maximum precision values at standard recall values for both metrics, evaluated over the full query set.

$$\frac{M}{i(i + \ln(M))} - i + (rt - ir) \left(2a - 3M - 2i + \ln(l) + \frac{a}{M} + \frac{l}{M} + \ln(M + 0.293) + \ln \left(n + \frac{\ln(n)(M + N + t)}{N - 0.366l + n} \right) \right)$$

Figure 4.7: The simplified form of the best evaluator obtained during Experiment 1. Letters represent terminals as shown in Table 3.1.

tions, after which overtraining sets in. Improvements in performance, as judged by an increase in precision averaged over all recall values is of the order of five percent.

As expected, the average precision fitness function outperforms the fitness function computed by evaluating precision at 50% recall. It is interesting to note that even at 50% recall, performance of the former metric was better than that of the latter. This may simply be due to chance, or due to the fact that good performance at a certain recall level also requires improved performance at other levels. The precision at 50% recall metric may thus be inclined to move towards a local optimum, from which it cannot escape.

Savoy and Vrajitoru [43] state that improvements in average precision of 5% between different IR techniques are considered significant, with increases of greater than 10% seen as very significant. The best improvements seen using our approach are approximately 4.4%, falling a little short of these figures. On average, our approach exceeds standard *tf.idf* by only 3.2%. While these results show that the approach holds promise, further enhancements, suggested in Section 5.3 have the potential to yield greater improvements.

4.2.2 Experiment 2: Evaluation with new queries and documents

This experiment consisted of training the evaluators on a subset of both the document collection and the queries. Evaluation then took place on a combination of the original and remaining documents and queries. Only 50 generations were evaluated due to the early appearance of overtraining. Primary experimental

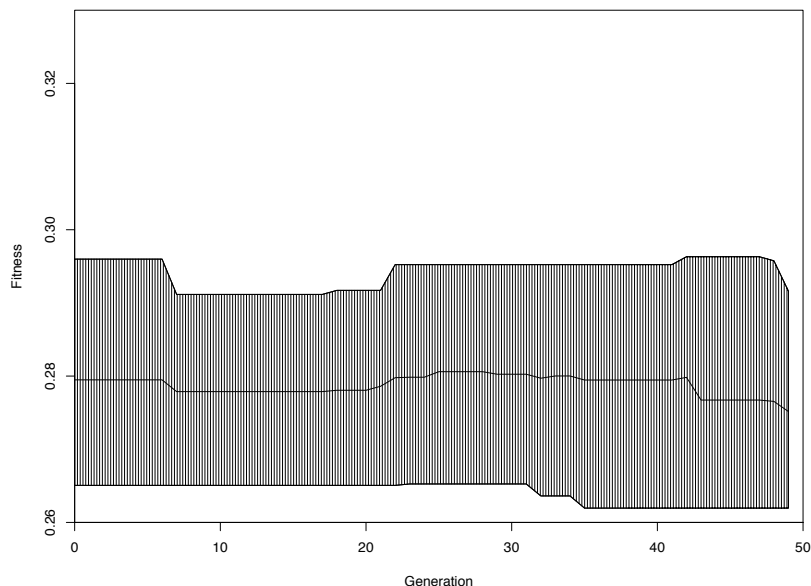


Figure 4.8: Maximum, mean and minimum fitnesses of the best evaluators for each generation, tested on testing documents and testing data.

observations consisted of:

- Evaluator fitness computed on the testing documents and testing queries.
- Evaluator fitness calculated using the training document set and testing queries.
- Evaluator fitness computed using the testing document set and training queries.
- Evaluator fitness computed on the full document set using the testing queries.

Secondary experimental observations, examined in an attempt to get more insight into the underlying behaviour of the system, included:

- Evaluator fitness calculated using the training document set and training queries.
- Fitness calculated using all the queries on the new document set.

Due to the better performance of the average-precision metric seen in Experiment 1, only this metric was evaluated when running this experiment.

Figure 4.8 shows how the evaluators performed when evaluated using test documents and queries. Figure 4.9 was generated using test data and training queries, while Figure 4.10 evaluates the individuals with training data and testing queries. Figure 4.11 shows the fittest individual evolved in this experiment, as judged when evaluating over testing queries and documents.

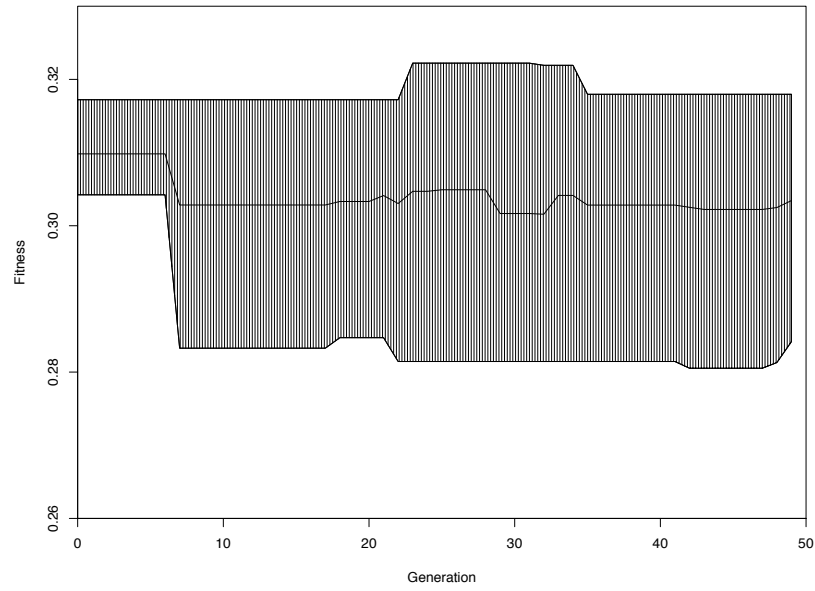


Figure 4.9: Maximum, mean and minimum fitnesses of the best evaluators for each generation, tested on testing documents and training queries.

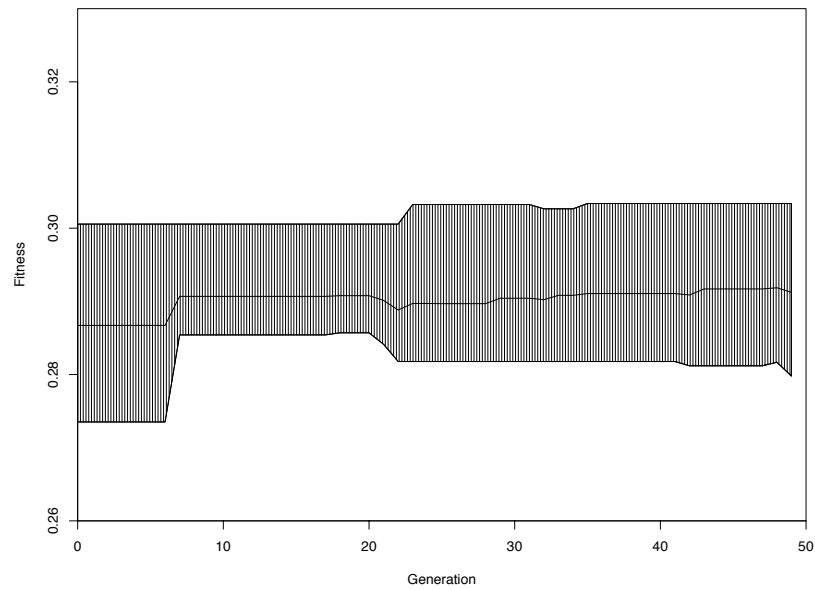


Figure 4.10: Maximum, mean and minimum fitnesses of the best evaluators for each generation, tested on training set documents and test set queries.

$$\frac{r}{l+n+r+0.436Mt+0.436t \ln(M)}$$

Figure 4.11: The simplified form of the best evaluator for Experiment 2.

Analysis

The most striking result of this experiment is the relatively poor performance achieved by this approach. Fitness, when evaluated over all documents and queries only improves by around one percent over standard *tf.idf*. Since this scenario is more realistic than the one described in the previous experiment, this result indicates that our approach may not perform well in a general information retrieval setting.

Looking at the experimental results until the seventh generation, where overtraining begins, an ordering of fitness levels between the various experiments emerges: The evaluation of training data with training queries yields the highest fitness results, followed by evaluation of the testing document set with training queries, testing queries and training documents, and finally, testing documents and queries.

The fact that the fitnesses for the first and last cases respectively achieve the highest and lowest fitnesses is not unexpected, as the former represents the training of the evaluators, while the latter represents the testing of the individuals on data and queries which they had not previously encountered. The remaining cases represent the middle ground between the two extremes, with evaluation taking place on either previously encountered documents or queries.

The ordering illustrated by the two intermediate cases can be explained in a number of ways:

1. A bias exists in that 70% of the queries were used in training, while only 50% of the documents were utilised for this purpose. Thus, evaluation occurs on the remaining 30% of queries and 50% of documents. This means that individuals needed to successfully adapt to only the small number of queries to perform well in the testing queries – training data case. In the training queries – testing data case, it is more difficult to perform well due to the variety of documents that must be returned. If this explanation is valid, two observations can be made:
 - (a) It is more difficult for the GP to adapt to varied queries than to varied documents.
 - (b) Thus, performance with new, unseen queries will be quite poor. The remainder of this experiment appears to validate this result. The results of Experiment 1 do however dispute this to some extent, and indicate that perhaps if a larger training set were available, better performance would be observed.
2. These results could occur due to an unlucky choice of training/testing documents and queries. If this is the case, rerunning the experiments a sufficient number of times using different subsets of data for training and testing will yield different results.

In an attempt to investigate further, the experiment was repeated six times with using identical training queries and documents (and hence identical testing queries and documents). Figure 4.12 shows a summary of the results.

Recall	Best evaluator	<i>tf.idf</i>
0	0.662	0.658
0.1	0.474	0.446
0.2	0.377	0.358
0.3	0.297	0.290
0.4	0.241	0.230
0.5	0.205	0.191
0.6	0.164	0.154
0.7	0.129	0.119
0.8	0.095	0.084
0.9	0.066	0.058
1	0.051	0.043

Table 4.4: Precision at the standard recall values for *tf.idf* and the best evaluator, trained on the CF dataset and evaluated on the CISI dataset.

While evaluation using testing queries and training documents still outperforms evaluation with training queries and testing documents, a large amount of overlap exists between the two cases. Since identical training/testing data was used for all cases, these results hint that our original results were not simply due to chance, but rather due to the behaviour described in case 1a above.

As seen previously, the performance of the evaluators is quite poor. On average, almost no improvement is seen in both of these cases. When used in the manner described previously, it appears as if the GP approach cannot generalise sufficiently well to be useful for information retrieval. It may be possible to salvage the approach by using larger (and more general) training sets, together with a requisite increase in computer power to handle the greater processing times.

4.2.3 Experiment 3: Evaluation on a new document collection

This experiment was designed to determine the general effectiveness of the approach, by training evaluators on one dataset, and evaluating the results on a different dataset. Figure 4.13 shows that evaluation fitness peaks within the first few generations, after which overtraining sets in. Figure 4.15 shows the fittest individual evolved during this experiment.

Precision–recall curves for *tf.idf* and the best performing evolved evaluator are shown in Figure 4.14. The precision values at the standard recall points are shown in Table 4.4.

Analysis

After the poor results obtained in experiment two, it was expected that the evaluators created in this experiment would be inferior to standard *tf.idf*. Surprisingly, this was not the case. While overtraining appeared to set in after approximately 8 generations, improvements in evaluation fitness of up to 4.96% were seen. The most probable reason for this improvement is the increased size of the dataset used in training as com-

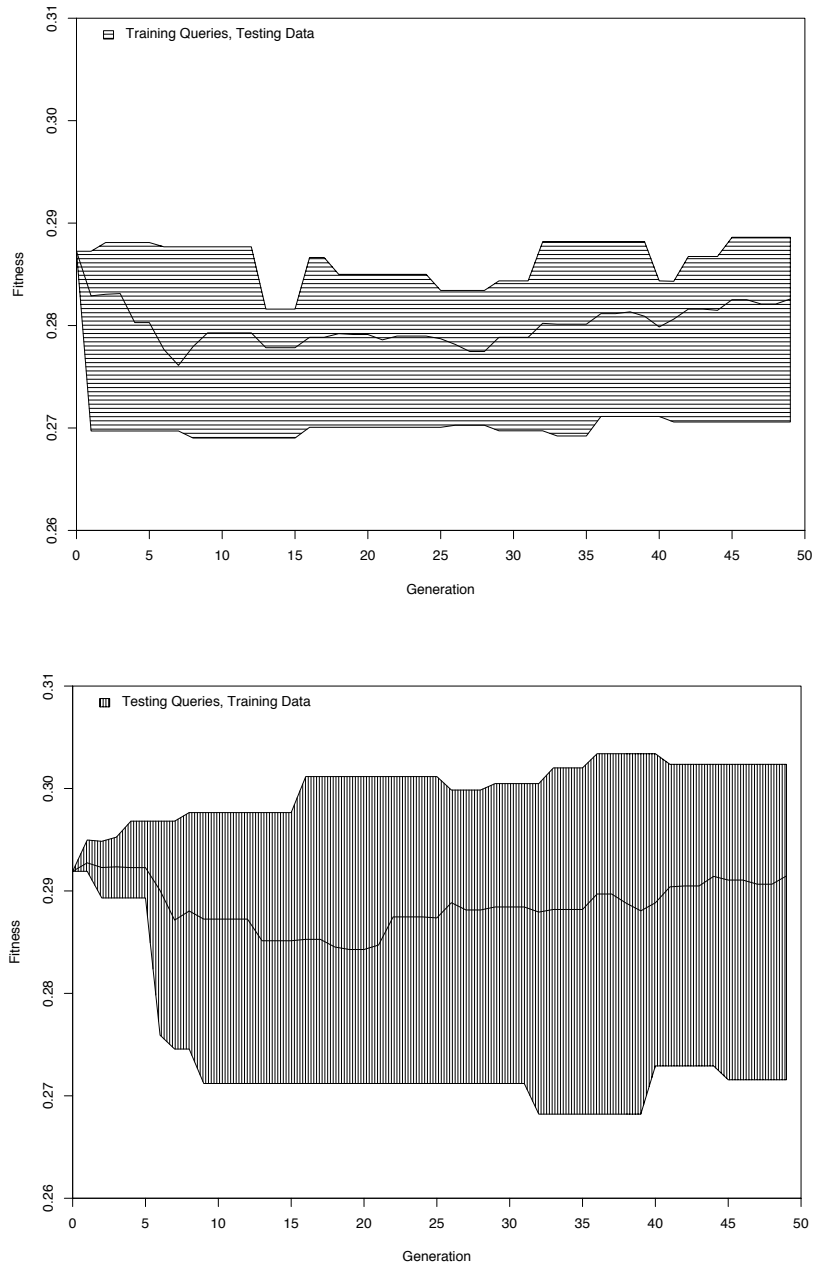


Figure 4.12: Minimum, maximum and mean fitness of the best evaluators for six experiments with two training set combinations, using the same identical training and testing datasets.

pared with the one used in the previous experiment. One cannot however rule out the possibility of “dumb luck”, i.e. that the random process used to evolve the individuals happened to proceed in such a way that an effective individual was created.

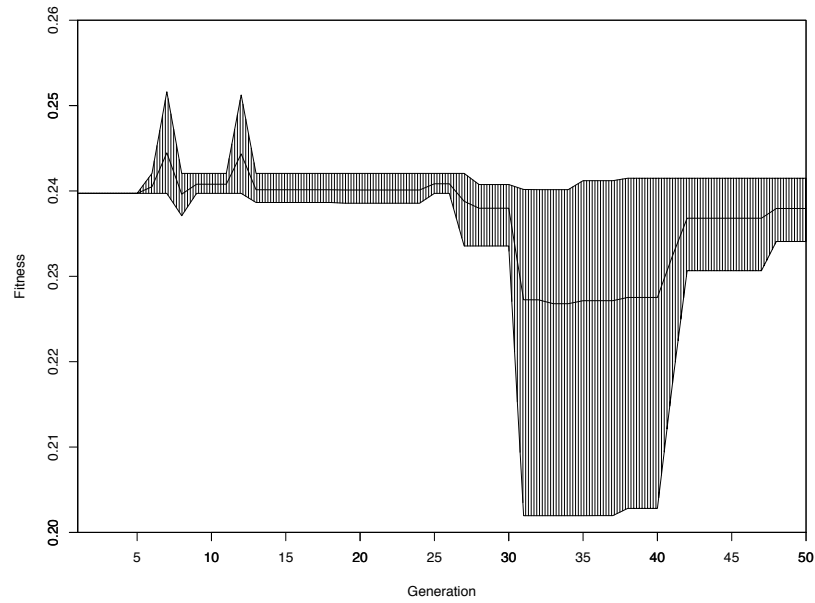


Figure 4.13: Minimum, maximum and mean evaluation fitness of the best evaluators when trained on the Cystic Fibrosis dataset and tested on the CISI dataset.

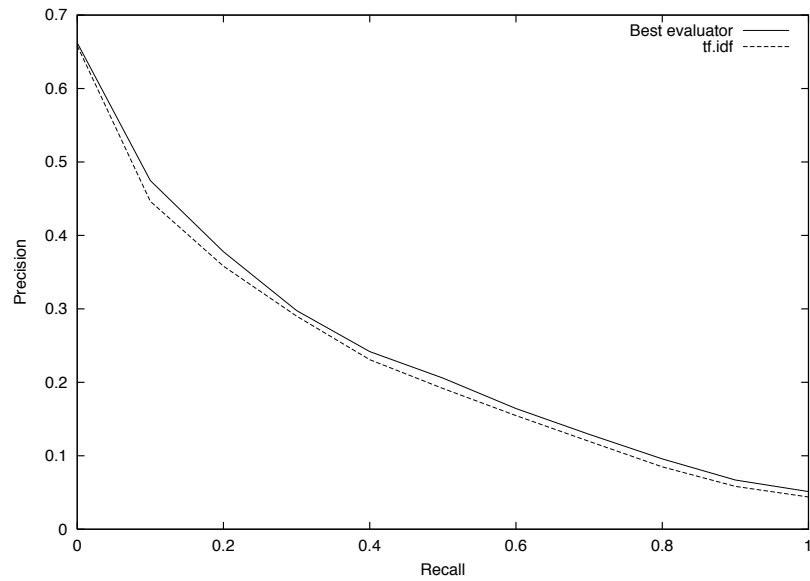


Figure 4.14: Precision recall curve for the best evaluator and *tf.idf* as trained on the standard data set and evaluated on the CISI dataset.

$$it(M + \ln(l + n + MNr))$$

Figure 4.15: The simplified form of the fittest individual evolved during experiment 3, as judged during the evaluation phase.

Coupled with the results of the previous experiment, it appears that given a sufficiently large dataset on which to train, it may be possible to evolve a general evaluator with performance superior to that of a standard *tf.idf* evaluator.

When compared to the evaluators evolved in the previous experiment, the short length of the fittest individual obtained here is striking. This individual also contains noticeable elements of standard *tf.idf*. Both of these phenomena can be explained by noticing that this individual appeared as a member of one of the early generations.

4.3 A few words about training time

An experimental run for 50 generations, evaluated on half (approximately 1000) documents and 70% of the queries, using 100 individuals, took approximately three hours to run on a cluster of 40 800MHz Pentium-III workstations.

Since overtraining set in within the first 15 or so generations (depending on experiment), it should be possible to greatly speed up the training process by reducing the number of generations evaluated.

By increasing the number of individuals within each generation, a larger area of the problem space can be searched in each iteration. This may lead to improved individuals, but may also cause overtraining to appear even earlier.

As mentioned in the third experiment, it appears as if increasing the training set size has a profound influence on the quality of the obtained solution. Unfortunately, the time complexity of training is $O(i*d*q)$ where i is the number of individuals, d is the number of documents, and q is the number of queries. Thus, increasing dataset size (consisting of both documents and queries), effectively increases the training time quadratically.

4.4 An examination of evolved individuals

Figure 4.16 shows the raw form of the fittest individual evolved during Experiment 1. A striking feature of a typical individual is the amount of redundant information it contains, consisting of expressions such as $\ln(\ln(\text{constant}))$, and the repetition of terms. Individuals evolved later on in an experimental run usually contain more such redundancies than those from earlier generations. While it is not difficult to simplify the individual and eliminate the extra information, any change in its structure would impact on the manner in which it evolves. The extraneous nodes enable the individual to be more resilient to harmful breeding operations [6]. For the same reason, the redundant nodes also slow down the rate at which improvements

$$\begin{aligned}
& (((((M) / ((\ln((M))) + (i))) / (i)) - (i)) - (((((M) - \\
& (\ln(((\ln((n)))) / (((N) - (l)) + ((l)) * (0.63462072873238)) + \\
& (n)) / ((N) + (M) + (t)))) + (n)))) - ((l) / (M)) - (((a) \\
& + (((\ln((0.29347407924639) + (M)))) - ((M))) - (((M) - \\
& (\ln((l)))) - ((a) / (M)))) - (i)) + (a)) - (i)) * ((t) - \\
& (i)) * (r)))
\end{aligned}$$

Figure 4.16: The unsimplified form of the best individual evolved during experiment 1.

appear. This effect did not hamper the experiments due to the speed at which overtraining set in. These redundant nodes are similar to “junk DNA” in biological systems, and are thus referred to by the same name: introns [6]. Since the fittest individuals appeared at close to 150 generations in the first experiment, while the best individuals appeared much earlier in the other experiments, the former individual contains many more introns than the latter.

By varying the likelihood of the breeding operations, some control may be gained over the rate of change in individual makeup.

Another interesting aspect of the evolved individuals is their dissimilarity to standard *tf.idf*, at least in the initial two experiments. Since the initial populations were all seeded with a *tf.idf* individual, and since this individual was, at least initially, amongst the fittest in the population, one would have expected a mutated version of *tf.idf* to remain the fittest individual throughout the experimental run. This result indicates that evaluators with forms very different to *tf.idf* may be suitable for IR systems.

4.5 Summary and Conclusions

The experiments described in this chapter were run to answer the research questions posed previously. The following was determined:

- In limited applications, genetic programming techniques can generate evaluators which have better recall and precision than standard *tf.idf*.
- With a sufficiently large training set, it appears as if the GP approach can generalise sufficiently well to operate over a large number of domains. With only limited training data however, the approach fails.
- While a query can be performed in a time comparable to standard techniques, the training of the system is very (and perhaps prohibitively) computationally expensive. Training need only be done once however.
- Individuals with forms very different to *tf.idf* may be suitable as evaluators in IR systems.

The first experiment validated the applicability of the approach in a very simple and limited case. Some improvements were seen over *tf.idf*. Furthermore, this experiment allowed me to discard a candidate metric in further experiments.

Experiment two evaluated the feasibility of the technique in a slightly more complex environment. This experiment yielded very poor results. I believe that this was due to the smaller data set used.

The third experiment examined how well genetic programming would function in a general domain. This was achieved by training on a dataset and testing on an unrelated dataset. Improvements of close to five percent were seen in some experimental runs. It was hypothesised that the improved performance over experiment two were achieved due to the increased size of the training dataset.

The efficiency of the approach was evaluated only in terms of training time, as it was shown in Section 3.5.4 that the runtime costs of the technique are acceptable.

The next chapter summarises the research, and provides a number of suggestions regarding future extensions to this research.

Chapter 5

Conclusions and further research

This chapter begins by summarising the research. It then provides suggestions for areas of further investigation. After examining the contributions of this research, Final conclusions are given.

5.1 Summary of research

This research evaluates the effectiveness of utilising genetic programming in creating evaluators by fitting to the query space in an attempt to provide improved precision and recall. These evaluators were tested in three separate scenarios:

1. Simulating an unchanging dataset by training and evaluating on all its documents, training on some queries, and evaluating the results using the remaining queries.
2. Training on a subset of both the queries and documents, and evaluating on the remaining queries and documents. This scenario simulates the addition of new but “similar” documents to the dataset.
3. Training on one document collection, and evaluating on a second set of documents and queries. This was done to examine the effectiveness of the approach in dealing with “general” collections.

5.2 Summary of results and conclusions

The genetic programming approach seems very effective for the first scenario, performs disappointingly in the second, and provides surprisingly good results in the third. The poor results appear to stem from very fast over-specialisation by individuals as training progresses, probably due to the small size of the training dataset.

The gain in retrieval performance shown by some of the experiments run during the course of this research approach the five percent mark suggested by Savoy and Vrajitoru as an indicator of significant performance improvement. These results indicate that the method described in this document merits further investigation. As mentioned in the previous chapter, the use of larger datasets may result in an even

larger performance gain. Furthermore, some suggestions are provided below that may also aid in yielding significant performance improvements with this approach.

5.3 Future work

While some of the results presented in this report are disappointing, a number of avenues for future work exist that may greatly improve the effectiveness of this approach.

5.3.1 Modifying breeding parameters

Changing the probability of the various breeding operations, together with the manner in which individuals are selected for breeding may greatly influence the quality of solutions obtained by this approach. The primary effect of different breeding operation probabilities is the speed at which good solutions are found. As was seen in the experimental results, the best evaluators appear very early on in an experimental run, and thus an increase in this aspect is not needed. It may however be possible to evaluate a broader range of solutions by modifying the breeding parameters, leading to improved evaluators before overtraining appears. The best way to achieve this may be by looking at other fitness functions.

5.3.2 Increased dataset size

The combined results of the second and third experiment suggest that improved performance may be obtained by increasing the size of the training set. By changing training set size, it is possible to delay the onset of overtraining, as well as forcing the individuals to adapt to more general queries. Similarly, by using cross-validation techniques, it may be possible to improve performance even when utilising a small dataset.

5.3.3 Different evaluators for queries and documents

As mentioned in the background section, *tf.idf* often utilises slightly different formulas when creating weighting vectors for documents and queries. Since using different weighting schemes appears to improve performance in the traditional approach, it should be investigated whether benefits can be gained by evolving different evaluators for each in the approach used here.

The main problem involved in using (and investigating) this approach is the increase in the number of evaluations one needed to evaluate the fitness of an individual in training. In the worst case, where two separate populations are kept, one for document evaluators and one for query evaluators, every individual within each population must be evaluated with every individual within the other population to provide a fitness measure. Issues also arise regarding how to transfer individuals to the next generation: If individual *a* provides good answers with individual *x* in the other population, but poor answers when combined with individual *y*, while individual *b* performs well with *y* but poorly with *x*, what fitness should be assigned to each individual? A number of possible solutions exist, but it is unknown which will lead to the best final evaluators. Another solution, involving less computation, is creating only one population, with each individual comprising of two evaluators, one of which evaluates documents, while the other operates on

queries. This doubles the number of calculations required for evaluation. It may be possible to come up with schemes that operate between these two extremes.

Another problem with this approach is the reduction in convergence speed. If both evaluators are approximately of length n , and the assumption is made that a query evaluator interacts in an arbitrary way with a document evaluator, it is possible to replace the two with a single evaluator of length n^2 . Effectively, the dimensionality of the space searched for a valid evaluator is thus squared.

5.3.4 Additional terminal types and operators

Only a small number of terminal nodes were used in this research. By adding new terminal nodes, individuals may be evolved that make use of more of the meaning from within documents and queries. By adding specific terminals, it may be possible to represent most current information retrieval methods. With sufficiently complex terminals, it should be possible to make use of semantic meaning from within the texts.

With the advantages provided by these new terminals come costs. Many complex terminals are very computationally expensive if used indiscriminantly. For example, a terminal that returns the distance in tokens between arbitrary pairs of words immediately causes a quadratic increase in the number of computations that must be performed. Some terminals are clearly even more expensive, while others complicate the structure of individuals by introducing the need for more complex operators, and perhaps some form of type checking.

It may be possible to create more complex evaluators by increasing the number of usable operators. It has however been shown [6] that evolved programs can very often create more complex operators by combining a number of simple operators. To obtain the greatest effect, the new operators should therefore not be easily derivable from any existing operators.

As mentioned earlier, another use for more complex operators is to support the existence of different terminals. For example, if boolean terminals are introduced (say to determine whether a certain word exists within a document), it makes sense to introduce boolean operators, as well as to add operators that enforce the closure property.

5.3.5 Multiple evaluation stages

Some of the techniques mentioned in this section may result in a prohibitive increase in training and evaluation time. By setting up a framework wherein all documents are evaluated using inexpensive techniques in early stages, and the most promising documents are then passed onto more computationally expensive later stages, it may be possible to reduce the amount of computation required to implement the techniques.

5.3.6 Relevance feedback training

The results from the previous chapter indicate that the programs evolved using GP performed poorly when documents on which they were not trained were added to the collection for evaluation purposes. It should be possible to modify the approach with a form of manual relevance feedback such that training takes place

continuously. Thus, as new documents are added to the collection, new evaluators could arise which would be better suited to the modified document set.

5.4 Contributions of this research

This research proposes a modification to the indexing function used in vector based information retrieval. Improvements in retrieval performance were seen which should directly translate to real-world IR systems. Furthermore, since only a small portion of the IR system was modified, this technique should be simple to integrate into existing systems. A full vector based IR system using this indexing function scheme would be able to leverage of other common techniques, such as relevance feedback, to further improve retrieval performance.

Due to the emergence of large electronic information stores, IR systems are becoming more and more ubiquitous. The slight improvement in retrieval performance obtained by this system can thus translate into large real world productivity gains.

5.5 Final word

While a large amount of work already exists on the utilisation of genetic programming (and other machine learning techniques) in information retrieval, much work remains to be done. Many of the commonly used techniques are empirical in nature, and an automated search (using machine learning methods) through the space of possible modifications to the base approach may yield good results.

Within the context of this research, it was seen that with operations and nodes similar to those found when performing *tf.idf*, genetic programming functions well in very simple cases. As more complex cases were presented to the system, it was found that a larger training set was required to obtain good results. With a sufficiently large training set, this approach looks promising, and further investigations should be conducted to determine whether the modifications suggested above can further improve the system's performance.

Bibliography

- [1] The CISI test collection, available at <http://www.cs.utk.edu/~lsi/corpa.html>.
- [2] Syed S. Ali and Susan McRoy. Links: information retrieval. *Intelligence*, 11(4):17–19, 2000.
- [3] Peter J. Angeline and Jordan B. Pollack. Evolutionary module acquisition. In *Proceedings of the 2nd Annual Conference on Evolutionary Programming*, pages 154–163, February 1992.
- [4] T. Bäck, F. Hoffmeister, and H. Schwefel. Applications of evolutionary algorithms. Technical report, University of Dortmund, Department of Computer Science, 1992.
- [5] Ricardo Baeza-Yates and Brethier Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley, 1999.
- [6] Wolfgang Banzhaf, Peter Nordin, Robert Keller, and Frank Francone. *Genetic Programming—an Introduction: on the automatic evolution of computer programs and its applications*. Morgan Kaufmann Publishers, 1998.
- [7] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, 30(1–7):107–117, 1998.
- [8] C. Buckley, G. Salton, and J. Allan. The effect of adding relevance information in a relevance feedback environment. In *Proceedings of the 17th annual international ACM-SIGIR conference on research and development in information retrieval*, pages 292–300, July 1994.
- [9] James P. Callan, W. Bruce Croft, and Stephen M. Harding. The INQUERY retrieval system. In *Proceedings of DEXA-92, 3rd International Conference on Database and Expert Systems Applications*, pages 78–83, 1992.
- [10] D. Y. Cho and B. T. Zhang. Genetic programming-based alife techniques for evolving collective robotic intelligence. In M. Sugisaka, editor, *Proceedings 4th International Symposium on Artificial Life and Robotics*, pages 236–239, B-Con Plaza, Beppu, Oita, Japan, 19–22 1999.
- [11] W. Bruce Croft and David D. Lewis. An approach to natural language processing for document retrieval. In *Research and Development in Information Retrieval*, pages 26–32, 1987.

- [12] Scott C. Deerwester, Susan T. Dumais, Thomas K. Landauer, George W. Furnas, and Richard A. Harshman. Indexing by latent semantic analysis. *Journal of the American Society of Information Science*, 41(6):391–407, 1990.
- [13] Marc Ebner. Evolution of a control architecture for a mobile robot. In Moshe Sipper, Daniel Mange, and Andre's Pe'rez-Urbe, editors, *Proceedings of the Second International Conference on Evolvable Systems: From Biology to Hardware (ICES 98)*, volume 1478, pages 303–310, Lausanne, Switzerland, 23-25 1998. Springer Verlag.
- [14] C. Faloutsos and D.W. Oard. A survey of information retrieval and filtering methods. Technical report, Department of Computer Science, University of Maryland, August 1995.
- [15] Weiguo Fan, Michael D Gordon, and Praveen Pathak. Personalization of search engine services for effective retrieval and knowledge management. In *Proceedings of the 2000 International Conference on Information Systems*, pages 20–34, December 2000.
- [16] Norbert Fuhr and Ulrich Pfeifer. Probabilistic information retrieval as combination of abstraction inductive learning and probabilistic assumptions. *ACM Transactions on Information Systems*, 12(1):92–115, 1994.
- [17] David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, 1989.
- [18] M. Gordon. Probabilistic and genetic algorithms in document retrieval. *Communications of the ACM*, 31(10):1208–1218, 1988.
- [19] William R. Hersh, Diane L. Elliot, David H. Hickam, Stephanie L. Wolf, and Anna Molnar. Towards new measures of information retrieval evaluation. In *Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 164–170, July 1995.
- [20] D. Hiemstra and A. de Vries. Relating the new language models of information retrieval to the traditional retrieval models. Technical Report TR–CTIT–00–09, Centre for Telematics and Information Technology, University of Twente, May 2000.
- [21] Djoerd Hiemstra. Term-specific smoothing for the language modeling approach to information retrieval: the importance of a query term. In *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 35–41. ACM Press, 2002.
- [22] Thomas Hofmann. Probabilistic latent semantic indexing. In *Research and Development in Information Retrieval*, pages 50–57, 1999.
- [23] John Holland. *Adaption in natural and artificial systems*. University of Michigan Press, 1975.

- [24] Yufeng Jing and W. Bruce Croft. An association thesaurus for information retrieval. In *Proceedings of RIAO 94*, pages 146–160, October 1994.
- [25] J. Koza. *Genetic Programming: On the programming of computers by means of natural selection*. MIT Press, 1992.
- [26] John Lafferty and Chengxiang Zhai. Document language models, query models, and risk minimization for information retrieval. In *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 111–119. ACM Press, 2001.
- [27] C. Langdon and A. Qureshi. Genetic programming - computers using "natural selection" to generate programs. Technical report, University College London, October 1995.
- [28] Victor Lavrenko and W. Bruce Croft. Relevance-based language models. In *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 120–127, 2001.
- [29] David Miller, Tim Leek, and Richard Schwartz. A hidden markov model information retrieval system. In *Proceedings of the 22nd annual international ACM-SIGIR conference on research and development in information retrieval*, pages 214–221, August 1999.
- [30] Julian F. Miller, Dominic Job, and Vesselin K. Vassilev. Principles in the evolutionary design of digital circuits - part II. *Genetic Programming and Evolvable Machines*, 1(3):259–288, 2000.
- [31] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.
- [32] Y. Ogawa, T. Morita, and K. Kobayashi. A fuzzy document retrieval system using the keyword connection matrix and a learning method. *Fuzzy Sets and Systems*, 39:163–179, 1991.
- [33] Praveen Pathak, Michael Gordon, and Weiguo Fan. Effective information retrieval using genetic algorithms based matching functions adaptation. In *Proceedings of the 33rd Hawaii International Conference on System Science*, 2000.
- [34] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers, 1988.
- [35] Jay M. Ponte and W. Bruce Croft. A language modeling approach to information retrieval. In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 275–281, August 1998.
- [36] M. F. Porter. An algorithm for suffix stripping. *Program*, 14:130–137, July 1980.
- [37] C.J. Van Rijsbergen. *Information Retrieval, 2nd Ed*. Butterworth & Co, 1979.
- [38] S.E Robertson and K. Sparck Jones. Relevance weighting of search terms. *Journal of the American Society for Information Science*, 1977.

- BIBLIOGRAPHY 57
- [39] G. Salton and M. E. Lesk. Relevance assessments and retrieval system evaluation. *Inf. Stor. Retrieval*, 4:343–359, December 1968.
 - [40] Gerard Salton. *Automatic Text Processing*. Addison-Wesley, 1989.
 - [41] Gerard Salton and Christopher Buckley. Term-weighting approaches in automatic text retrieval. *Information Processing & management*, 24(5):513–523, 1988.
 - [42] Gerard Salton, Edward A. Fox, and Harry Wu. Extended boolean information retrieval. *Communications of the ACM*, 26(11):1022–1036, 1983.
 - [43] J. Savoy and D. Vrajitoru. Evaluation of learning schemes used in information retrieval. Technical Report CR-I-95-02, Université de Neuchâtel, Faculté de droit et des Sciences Économiques., 1996.
 - [44] Fei Song and W. Bruce Croft. A general language model for information retrieval (poster abstract). In *Research and Development in Information Retrieval*, pages 279–280, 1999.
 - [45] Ross Wilkinson and Philip Hingston. Using the cosine measure in a neural network for document retrieval. In *Proceedings of the 14th annual international ACM/SIGIR conference on Research and development in information retrieval*, pages 202–210. ACM Press, 1991.
 - [46] B. Wondergem, P. van Bommel, and T. van der Weide. Boolean index expressions for information retrieval. Technical Report CSI-R9827., University of Nijmegen, December 1998.
 - [47] S. K. M. Wong, Wojciech Ziarko, and Patrick C. N. Wong. Generalized vector spaces model in information retrieval. In *Proceedings of the 8th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 18–25, 1985.
 - [48] Jinxi Xu and W. Bruce Croft. Query expansion using local and global document analysis. In *Proceedings of the 19th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 4–11, August 1996.
 - [49] J. Yang and R. R. Korfhage. Query improvement in information retrieval using genetic algorithms: A report on the experiments of the TREC project. In *Proceedings of the 1st Text Retrieval Conference*, pages 31–58, March 1993.