Intelligent Fusion of Evidence from Multiple Sources for Text Classification

Baoping Zhang

Dissertation submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Computer Science and Applications

Advisory Committee:

Edward A. Fox, Chair
Weiguo Fan
Chang-Tien Lu
Dan Spitzner
Pável Calado

June 20, 2006
Blacksburg, Virginia

Keywords: digital libraries, classification, Genetic Programming, experimentation.

Intelligent Fusion of Evidence from Multiple Sources for Text Classification

Baoping Zhang

Abstract

Automatic text classification using current approaches is known to perform poorly when documents are noisy or when limited amounts of textual content is available. Yet, many users need access to such documents, which are found in large numbers in digital libraries and in the WWW. If documents are not classified, they are difficult to find when browsing. Further, searching precision suffers when categories cannot be checked, since many documents may be retrieved that would fail to meet category constraints. In this work, we study how different types of evidence from multiple sources can be intelligently fused to improve classification of text documents into predefined categories. We present a classification framework based on an inductive learning method – Genetic Programming (GP) – to fuse evidence from multiple sources. We show that good classification is possible with documents which are noisy or which have small amounts of text (e.g., short metadata records) – if multiple sources of evidence are fused in an intelligent way. The framework is validated through experiments performed on documents in two testbeds. One is the ACM Digital Library (using a subset available in connection with CITIDEL, part of NSF's National Science Digital Library). The other is Web data, in particular that portion associated with the Cadê Web directory. Our studies have shown that improvement can be achieved relative to other machine learning approaches if genetic programming methods are combined with classifiers such as *kNN*. Extensive analysis was performed to study the results generated through the GP-based fusion approach and to understand key factors that promote good classification.

Acknowledgements

First and foremost, I sincerely thank my advisor, Professor Edward A. Fox, for his encouragement, guidance, and warm support. The completion of this work would not be possible without his inspiration and generous support.

I also thank my other committee members. In specific, I'd like to thank Dr. Weiguo Fan for his enthusiasm and support, as well as his guidance and thoughtful comments on my work. I thank Dr. Chang-Tien Lu for his thoughtful ideas and constructive comments. I also thank Dr. Dan Spitzner for his valuable ideas, discussions, and important references provided. Special thanks go to Dr. Pável Calado, who always contributed valuable ideas and discussions throughout the course of this research.

In addition, I thank members of the Digital Library Research Laboratory for their encouragement and helpful suggestions. Special thanks go to Marcos André Gonçalves, Raghavendra Nyamagoudar, and Yi Ma for their great help with the work. I also acknowledge the collaborators of this work, Marco Cristo, Ricardo Torres, and Samantha Chandrasekar, for their support and help.

Lastly, I would like to thank my parents and my husband, Yuxin Chen. Their constant support was and will be essential to my endeavors.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Text classification is a task to automatically assign semantic categories to natural language text. In recent years, automated classification of text into predefined categories has attracted considerable interest, due to the increasing volume of documents in digital form and the ensuing need to organize them.

1.1 Text Classification

Text classification (TC – a.k.a. text categorization, or topic spotting), is the activity of labeling natural language texts with thematic categories from a predefined set [100]. TC dates back to the early '60s and became a major subfield of the information systems discipline in the early '90s due to increased applicability together with the availability of more powerful hardware. Initially, the approach to TC was to manually define a set of rules encoding expert knowledge on how to classify documents under the given categories, known as the knowledge engineering (KE) approach. Nowadays the dominant approach is one of building text classifiers automatically by learning the characteristics of the categories from a training set of pre-classified documents. State-of-the-art machine learning methods have recently been applied to the task, leading to systems of increased sophistication and effectiveness, and stably placing automatic text classification (ATC) at the crossroads of information retrieval and machine learning. This has encouraged the application of TC techniques to many contexts, ranging from document indexing based on a controlled vocabulary, to document filtering, automated metadata generation, word sense disambiguation, population of hierarchical catalogues of Web resources, and in general any application requiring document orga-

nization or selective and adaptive document dispatching.

Text classification is the task of assigning a Boolean value to each pair $\langle d_j, c_i \rangle \in D \times C$, where D is a set of documents and $C = \{c_1, \ldots, c_{|c|}\}$ is a set of predefined categories. A value of T assigned to $< d_j, c_i >$ indicates a decision to classify document d_j under category c_i, while a value of F indicates a decision not to classify document d_j under category c_i. More formally, text classification is the task to approximate the unknown target function $\breve{\Phi} : D \times C \rightarrow \{T, F\}$, which describes how documents ought to be classified, by means of a function $\Phi : D \times C \rightarrow \{T, F\}$ called the classifier such that $\breve{\Phi}$ and Φ "coincide as much as possible". The degree of match indicates effectiveness; how to measure this effectiveness will be discussed in Section 4.4.

1.2 Text Classification in Web-based Digital Libraries

Text classification has witnessed a booming interest in recent times, due to the increasing number of large document collections and the need to organize them for easier use. However, traditional content-based classifiers are known to perform poorly in Web-based digital libraries (DLs) [14, 50]. In the Web-based DLs, documents are usually noisy and contain much erroneous information, e.g., digitized through speech recognition or OCR. Documents in the Web-based DLs often contain scarce textual content, e.g., metadata records in DL catalogs or in databases of abstracts. Furthermore, documents in Web-based DLs might contain images, scripts, and other types of data unusable by text classifiers. For those reasons, we have to utilize other sources of evidence, other than textual content, to improve classification. In the ever-increasing number of DLs accessible through the Web, there are many opportunities, needs, and challenges for classification. In DLs, information is explicitly organized, described, and managed – and community-oriented services are built to support specific information needs and tasks. DLs can be thought of as being in the middle of a large spectrum between databases and the Web (see Figure 1.1). In the Web, we have a very unstructured environment and very little is assumed about users. On the other hand, databases assume very rigid structures/standards and very specialized users.

Accordingly, DLs offer both (1) the opportunity to explore the rich and complex internal (semi-) structured nature of documents and metadata records in the classification task; and (2) the social networks occuring in specific communities, as expressed for example by citation patterns in the research literature.

Many DLs, especially those created by aggregation of other sub-collections/catalogs (e.g., those

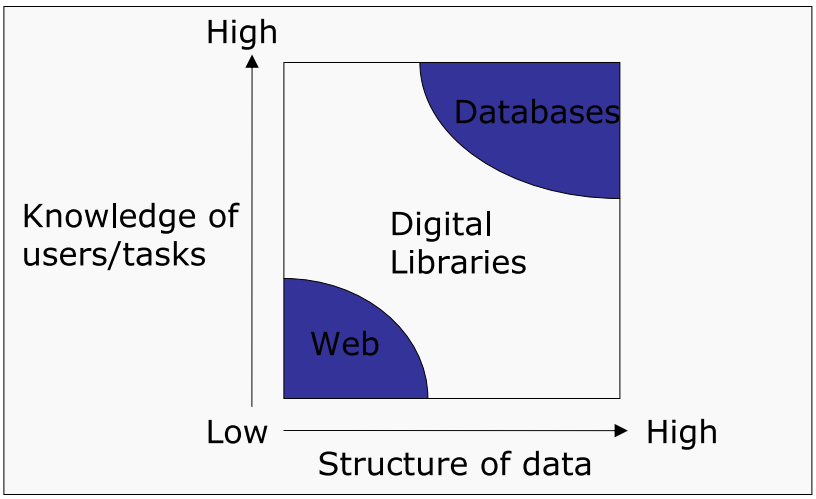


Figure 1.1: Information Spectrum (adapted from [91]).

assembled by publishers – like the ACM Digital Library [1], or by service providers that build upon the success of the Open Archives Initiative [80] – like the Networked Digital Library of Theses and Dissertations [79]), suffer from problems of quality of information. One such problem is incompleteness (e.g., missing information). This makes it very hard to classify documents using traditional content-based classifiers like Support Vector Machines (SVMs) [56] or Naive Bayes [70]. Another problem is imprecision. For example, citation-based information (e.g., as found in some entries in the ACM Digital Library) is often obtained through OCR of references, a process which leads to a significant number of errors.

Another motivation of this work is to enhance teaching and learning in the computing field, by improving the CITIDEL system [20], part of the National Science Digital Library (NSDL). Many of the records in CITIDEL lack classification information. That makes it difficult for users to browse, even with the support of our advanced multischeming approach [66]. It also limits the scope of advanced interfaces to CITIDEL, which make use of category information [58]. Further, the lack of classification information reduces broader downstream impact that could result from use of records harvested from CITIDEL and similar systems, such as into NSDL or NDLTD (e.g., to support browsing of computing dissertations [124]).

The work presented here aims to overcome these problems with classification by applying automatically discovered fusion techniques to the available evidence. In particular, we investigate an inductive learning method – Genetic Programming – for the discovery of better fused similarity functions to be used in the classifiers, and explore how this combination can be used to improve

classification effectiveness.

1.3 Objectives, Broad Impacts, and Contributions

This work combines evidence, through Genetic Programming (GP), to improve text classification, overcoming the problems discussed above. Evidence can come from different sources, like structural content information and citation-based information. The purpose is to research whether and how GP can effectively combine multiple sources of evidence and produce better results than other competitive baselines such as content-based SVM classifiers, combination-based SVM classifiers, or other fusion techniques.

1.3.1 Objectives

Our goal is to create a solution that is extensible, flexible [47], integrated, efficient, and effective. Our objectives include:

1. Discovering the most effective methods of machine learning that solve difficult but common problems in text classification, i.e., confirming through extensive experiments that GP is the most effective technique to combine different sources of evidence to tackle those problems.
2. Discovering why the solutions unveiled by GP work so well, thereby gaining a much better understanding of key factors to successful text classification.
3. Discovering the best sampling methods that address scalability issues.
4. Enhancing the effectiveness of CITIDEL (especially with regard to content from the ACM Digital Library), thence of NSDL, and of other systems like NDLTD, by classifying hitherto unclassified texts.
5. Enhancing the effectiveness of Web services that are based on categories and classification, like Cadê or DMOZ (i.e., the Open Directory Project) [10, 29].

1.3.2 Broader Impacts and Contributions

Scenario

Imagine a student or researcher, with focus on human-computer interaction, seeking to explore the topic "metasearch". She goes to Google and finds there are over 1.5 million results. Scanning through a number of screens of results, she finds nothing that seems relevant – indeed, she is angered by seeing mostly advertisements/announcements of metasearch systems. Thinking that a DL might be more helpful, she goes to the ACM Digital Library, and finds 151 records. But she really does not understand the jargon found in articles from *ACM Transactions on Information Systems* or ACM SIGIR conference proceedings. She tries the query "metasearch interface" and finds 200 records, but they are concerned with protocols and/or architectures and seem even less useful. Hoping that a category-based system might help, she goes to DMOZ, but then finds 88 sites, organized in accordance with 5 Open Directory Categories, none of which seems to match her interest. Frustrated, she realizes she does not know enough to formulate an effective search, and faces the reality that classification systems she knows don't seem helpful. Fortunately, she knows a friend who uses CITIDEL and who suggests using the *ACM Computing Classification System* (CCS) for browsing. Selecting the hierarchical category "Information Management: Information storage and retrieval", they note there are 5307 records. Searching in that category for "metasearch" they find 9 items, where the 2nd and 5th are clearly of interest. While happy to find these two helpful papers, she learns from her friend that: many available papers in the ACM Digital Library lack CCS or equivalent category codes, more than half of the entries lack abstracts, and little work is being done to accomodate these deficiencies. She wishes that the type of multi-scheme browsing found in CITIDEL are now found in other systems too, and that advanced, effective, and efficient methods of text classification are commonly employed.

This research addresses common scenarios, like that mentioned above, wherein categories are helpful. We produce basic findings concerning the effective and efficient integration of information from multiple sources. The algorithms developed in this work should be generally understandable and capable of being implemented in a wide variety of information applications. This research would apply to diverse information processing areas like: document classification, searching, teaching, and learning. The contribution of our work includes:

- Basic findings concerning the effective and efficient integration of information from multiple sources.

- Discovery of what factors lead to improved text classification, and how they can be combined most effectively.
- Uncovering solutions to ensure efficient implementation, such as through sampling.
- Delivery of open source code and educational materials, as well as testbeds (available at http://www.dlib.vt.edu/GP/), to stimulate teaching, learning, and research in this area.

The techniques we develop should be applicable to other content collections where data is noisy or where only small amounts of text are available. More generally, our methods of combining multiple sources of evidence should be applicable in other environments, where such evidence can help improve accuracy. Thus, there should be broad impact and benefit to users of all types of information systems, where accurate classification is of value. Even beyond that, our use of genetic programming, combined with sampling techniques, should lead to a new, efficient, and effective methodology wherein diverse sources of information can be usefully integrated into systems so that they will evidence more robust intelligence. Published results of our work can cause beneficial impacts on the general public, as well as the scholarly community. There is definitely potential for this research to have far-reaching impacts that will benefit multiple institutions and constituencies, since virtually all libraries are today confronted with the need for effective tools to organize digital objects, and this research will inform the implementation of such systems, as well as related Web systems.

1.4 Organization of this Work

In this chapter, we give some background on text classification and introduce the problems and motivation of this work. Chapter 2 discusses research related to our work. Chapter 3 introduces basic concepts related to Genetic Programming and then formally defines our GP based classification framework.

Chapters 4 and 5 present experimental designs and results. In particular, Chapter 4 discusses our testbeds, sampling strategy to ensure efficiency, experimental data set design, and baselines. Chapter 5 presents the experimental results on the two testbeds and the comparison between our approach and other text classification technologies. In Chapter 6, we try to analyze the experimental results to understand why and how GP based classification works. We try to analyze the

statistical property of the results as well as to search for key factors that contribute to the good classification results.

Finally, in Chapter 7 we give conclusions and suggestions regarding future steps to be taken to solve the problems left open by this research.

Chapter 2

Related Work

In this chapter, we review works related to text classification combining citation information in bibliometric science and link information in the Web environment. Section 2.1 reviews how different researchers have used link information to improve classification results. Support Vector Machines have been extensively evaluated for text classification and serve as one baseline to compare against our approach, so we describe SVMs in Section 2.2. Since our classification framework uses Genetic Programming to combine different sources of evidence, important works related to the use of Genetic Programming in information retrieval are described in Section 2.3.

2.1 Web Document Classification

Citation information among documents was first used in bibliometric science. There has been research to use citations to find papers on related topics as well as to measure the importance of a publication. Kessler [59] introduced the measure of bibliographic coupling in 1963. Two documents share one unit of bibliographic coupling if both cite a same document. So documents with similar topics would have a high value of bibliographic coupling. Thus bibliographic coupling was used to cluster scientific journals. Later, co-citation was proposed by Small [104], as a measure of similarity between scientific papers. Two papers are co-cited if a third paper has citations to both of them. Both bibliographic coupling and co-citation have been used as complementary sources of information for document retrieval and classification [2, 6]. There also were works that use citations in hypertext retrieval systems [7, 24, 42]. For example, in [25], Croft et al. used spreading activation techniques for hypertext retrieval.

The ideas of citations among documents can be adapted to the Web environment. Taking the ideas of citations in bibliometric science, Brin and Page [8] proposed an algorithm, PageRank, that uses the Web link structure to derive a measure of popularity for Web pages. Later, Kleinberg proposed the HITS (Hypertext Induced Topic Selection) algorithm to rank pages based on the link information among a set of pages. The algorithm presumes that a good authority is a page that many pages point to, and a good hub is a page that links to many authority pages. Hubs and authorities exhibit a mutually reinforcing relationship: a better hub points to many good authorities, and a better authority is pointed to by many good hubs.

In the World Wide Web environment, several works have successfully used link information to improve classification performance. Different information about links, such as anchor text describing the links, text from the paragraphs surrounding the links, and terms extracted from linked documents, has been used to classify documents. For example, Furnkranz et al. [43], Glover et al. [45], and Sun et al. [108] show that use of anchor text and the paragraphs and headlines that surround the links helps improve the classification result. Yang et al. [122] claimed that the use of terms from linked documents works better when neighboring documents are all in the same class. Other researchers applied learning algorithms to handle both the text components of the Web pages and the links between them. For example, Joachims et al. [57] studied the combination of support vector machine kernel functions representing co-citation and content information. Cohn et al. [23] showed that a combination of link-based and content-based probabilistic methods improved classification performance. Fisher and Everson [40] extended this work by showing that link information is useful when the document collection has a sufficiently high density in the linkage matrix and the links are of high quality. Chakrabarti et al. [14] estimated the category of test documents by studying the known classes of neighboring training documents. Oh et al. [81] improved on this work by using a filtering process to further refine the set of linked documents to be used. Calado et al. [11] proposed a Bayesian network model to combine the output of a content-based classifier and the information provided by the documents' link structure. In our classification framework, we combine document content information and citation information to improve classification performance.

2.2 Support Vector Machines

Support Vector Machines have been extensively evaluated for text classification on reference collections and reportedly have been the best classifier in some text classification experiments [56].

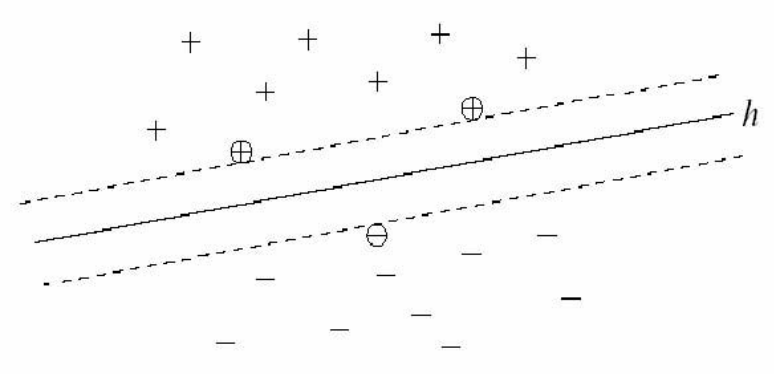


Figure 2.1: Support Vector Machines (SVMs) find the hyperplane, h, which separates the positive and negative training examples with maximum distance. The examples closest to the hyperplane are called Support Vectors (marked with circles).

A content-based SVM classifier was first used in text classification by Joachims [56]. It works over a vector space, where the problem is to find a hyperplane with the maximal margin of separation between two classes. This hyperplane can be uniquely constructed by solving a constrained quadratic optimization problem, by means of quadratic programming techniques. The optimal separating hyperplane will minimize the risk of mis-classifying the training samples and unseen test samples. Figure 2.1 illustrates this idea.

2.3 GA/P in Information Retrieval

Genetic algorithms have been used to support intelligent information retrieval for quite a long time. Gordon [48] used a genetic algorithm based approach to index documents. Competing document descriptions (sets of keywords) were associated with a document and altered over time to improve retrieval. Following a similar approach, Gordon [49] also studied the problem of document clustering, showing that, as document descriptions were changed, those relevant to similar queries began to form clusters. Raghavan and Agarwal [89] tried to directly generate clusters of documents, without trying to adjust the descriptions of individual documents. In Petry et al. [84], a weighted Boolean query was modified by a genetic algorithm in order to improve recall and precision. Yang et al. [118, 119] adapted weights assigned to query terms based on user feedback and reported the effect of adopting genetic algorithms in large-scale databases. Chen et al. [17] attempted to learn a user's information need by combining query-by-selected-samples and machine learning

techniques. Their results indicate that a learning system based on a genetic algorithm consistently outperforms other approaches. Chen et al. compared the performance of Web "spiders" governed by either a genetic algorithm or a best-first search [16]. When given the same initial set of Web pages provided by a user, a GA-based spider uncovered more new, relevant documents than the best-first search spider. A similar Internet agent/spider system based on a genetic algorithm was developed by Martin-Bautista et al. [74]. Fan et al. [33, 35, 34, 37, 38] have successfully applied GP to discover personalized ranking functions by effective combination of different weighting features. GP has been applied to data classification as well. In [15], Cheang et al. proposed a Genetic Parallel Programming classifier to evolve parallel programs for data classification problems. Eggermont et al. [32] proposed several methods using techniques from the field of machine learning to refine and reduce search space sizes for evolving decision trees for data classification. They showed how classification performance improves when shrinking the space in which a tree-based Genetic Programming algorithm searches. Kishore et al. [61] proposed a methodology for GP-based n-class pattern classification. They modeled the given n-class problem as n two-class problems; further, a GP classifier expression was evolved as a discriminant function for each class. In [60], they integrated the GP classifier with feature space partitioning for localized learning to improve pattern classification. Genetic Programming also has been applied to text classification through the use of a parse-tree. Clack et al. [21] used GP to route in-bound documents to a central classifier which autonomously sent documents to interested research groups within a large organization. The central classifier used a parse tree to match the aspects of a document to nodes of the tree, which ultimately leads to a single numerical value – the classification or "confidence value" – during evaluation. Castillo et al. [28] developed a multistrategy classifier system for document classification. They applied different types of classifiers (e.g., Naive Bayes, Decision Trees) to different parts of the document (e.g., titles, references). Genetic algorithms are applied for feature selection as well as for combining the output of the different classifiers.

2.3.1 Analysis of GP Trees

As discussed in Chapter 1, one objective of this work is to understand why the solutions unveiled by GP work so well. The GP discovered solutions are represented as trees, discussed in detail in Chapter 3. There have been different works to analyze tree properties like edit distance, subtree mining, and statistical properties of a population of trees.

Some research has involved comparing two trees based on various distance measures. [101, 109,

126] recently generalized one of the most commonly used distance measures, namely the edit distance, for both rooted and unrooted unordered trees. Here a rooted tree is defined as a free tree with a distinguished vertex that is called the root. In a rooted tree, if vertex v is on the path from the root to vertex w then v is an ancestor of w and w is a descendent of v. If in addition v and w are adjacent, then v is the parent of w and w is a child of v. A rooted ordered tree is a rooted tree that has a predefined left-to-right order among children of each vertex. In [111], Valiente proposed a bottom-up approach for calculating distance between trees in linear time of the number of nodes. Though methods of calculating distances between trees have been known for a long time, we are not aware of any published statistical analysis of the population of trees using such distance measures.

C. R. Rao [90] proposed methods to calculate the apportionment of diversity which is used to study the variance in the population. These methods are used to study the dissimilarity between populations and diversity within a population, with application in genetic studies. Motivated by Rao's work, Feigin and Alvo [39] proposed a general approach to comparing populations of rankings. We adapt this framework to analyze populations of trees by studying the dissimilarity among trees belonging to different classifiers. On the other hand, we try to find the mean tree of a population of trees. Related to this, Wang [115] proposed a method to generate a mean tree for a population of trees. This method is a top-down approach and uses the set of frequent nodes in the population of trees to derive a mean tree.

The GP trees are large, complex, and of variable size and structure, making them hard to interpret. There is a need for methods to address the interpretability issues. We can essentially 'probe' the trees to analyze which of the similarity functions (represented as trees) occur most frequently, which would provide an insight into "why only certain similarity functions work while others fail to produce good results". This is one of the analysis techniques employed, discussed in detail in Section 6.2, called common subtree searching. Though studying algorithms for mining subtrees is a relatively new field of research, yet quite a lot of work has been done to mine data from rooted and unrooted trees, and from forests. Chi et al. [18] proposed an algorithm to mine subtrees from free trees and summarized the characteristics of a tree-mining algorithm and provided a comparative analysis of such algorithms. Zaki [123] proposed a method to mine embedded subtrees from trees (rooted and unrooted). Asai et al. [5] provided a tool for mining induced subtrees from a population of trees. The mining algorithms are used for various applications like XML document mining, association rule mining, etc. These algorithms have not been used to analyze the subtrees to understand the workings of GP based techniques for text classification. Loveard [71] provided

a method for analysis of building blocks (common subtrees) on an artificial data set constrained to trees of maximum depth of four, and also provided analysis of the contribution of such common subtrees towards success or failure of a tree. Earlier works have attempted to identify if building blocks exist [67], and to track the frequency of smaller components in trees for classification [71], but none of them have analyzed these components or blocks for large trees like in our text classification problem.

Chapter 3

Approach

In this research we apply automatically discovered fusion techniques to available evidence as we solve key problems in text classification. Particularly, we investigate an inductive learning method – Genetic Programming – for the discovery of better fused similarity functions to be used in classifiers. We explore how such combination methods can be used to improve classification effectiveness. Finding the best combination requires explicit representation, use of machine-learning techniques, and use of different sampling strategies in implementation.

Three different content based similarity measures applied to the content of documents will be used: bag-of-words, cosine, and Okapi. Five different citation-based similarity measures also will be used: bibliographic coupling, co-citation, Amsler [3], and Companion (authority and hub) [27]. The new similarity functions, discovered through GP, will be applied to *kNN* classifiers [120] to get the final classification results.

3.1 Similarity Measures

To obtain these similarity measures, the documents are represented as vectors in the Vector Space Model [97]. Suppose we have a collection with t distinct index terms t_j. A document d_i can be represented as follows: $d_i = (w_{i1}, w_{i2}, ..., w_{it})$, where w_{ij} represents the weight assigned to term t_j in document d_i. For the bag-of-words measure, the similarity between two documents d_1 and d_2 can be calculated as:

$$\textit{bag-of-words}(d_1, d_2) = \frac{|\{d_1\} \cap \{d_2\}|}{|d_1|} \tag{3.1}$$

where $\{d_i\}$ corresponds to the set of terms occuring in document d_i. For the cosine measure, the similarity between two documents can be calculated [98] as:

$$cosine(d_1, d_2) = \frac{\sum_{i=1}^{t} w_{1i} * w_{2i}}{\sqrt{(\sum_{i=1}^{t} w_{1i}^2) * (\sum_{i=1}^{t} w_{2i}^2)}} \tag{3.2}$$

For the Okapi measure, the similarity between two documents can be calculated as:

$$okapi(d_1, d_2) = \sum_{t \in d_1 \cap d_2} \frac{3 + tf_{d_2}}{0.5 + 1.5 * \frac{len_{d2}}{len_{avg}} + tf_{d_2}} * \log \frac{N - df + 0.5}{df + 0.5} * tf_{d_1} \tag{3.3}$$

where tf is the term frequency in a document, df is the document frequency of the term in the whole collection, N is the number of documents in the whole collection, len is the length of a document, and len_{avg} is the average length of all documents in the collection.

From Eqs. (3.1), (3.2), and (3.3), we can see that the cosine similarity matrix is symmetric while the bag-of-words and Okapi similarity matrices are not.

Co-citation [104] measures similarity between scientific papers. It reflects the assumption that the author of a scientific paper will cite only papers related to his own work. To further refine this idea, let d be a document and let P_d be the set of documents that cite d, called the *parents* of d. The co-citation similarity between two documents d_1 and d_2 is defined as:

$$cocitation(d_1, d_2) = \frac{P_{d_1} \cap P_{d_2}}{|P_{d_1} \cup P_{d_2}|} \tag{3.4}$$

Eq. (3.4) tells us that the more parents d_1 and d_2 have in common, the more related they are. This value is normalized by the total set of parents, so that the co-citation similarity varies between 0 and 1.

Also with the goal of determining the similarity between papers, the idea of bibliographic coupling [59] is based on the notion that authors who work on the same subject tend to cite the same papers. More formally, let d be a document. We define C_d as the set of documents that d cites, also called

the *children* of d. Bibliographic coupling between two documents d_1 and d_2 is defined as:

$$bibcoupling(d_1, d_2) = \frac{C_{d_1} \cap C_{d_2}}{|C_{d_1} \cup C_{d_2}|} \tag{3.5}$$

Thus, according to Eq. (3.5), the more children document d_1 has in common with document d_2, the more related they are. This value is normalized by the total set of children, to fit between 0 and 1. In an attempt to take the most advantage of the information available in citations between papers, Amsler [3] proposed a measure of similarity that combines both co-citation and bibliographic coupling. According to Amsler, two papers A and B are related if (1) A and B are cited by the same paper, (2) A and B cite the same paper, or (3) A cites a third paper C that cites B. Thus, let d be a document, let P_d be the set of parents of d, and let C_d be the set of children of d. The Amsler similarity between two documents d_1 and d_2 is defined as:

$$amsler(d_1, d_2) = \frac{(P_{d_1} \cup C_{d_1}) \cap (P_{d_2} \cup C_{d_2})}{|(P_{d_1} \cup C_{d_1}) \cup (P_{d_2} \cup C_{d_2})|} \tag{3.6}$$

Eq. (3.6) tells us that the more documents (either parents or children) d_1 and d_2 have in common, the more they are related.

Finally, taking a different approach, Dean and Henzinger proposed the Companion algorithm [27] for Web pages. Given a Web page d, the algorithm finds a set of pages related to d by examining its links. Companion is able to return a degree of how related the topic of each page in this set is to the topic of page d. This degree can be used as a similarity measure between d and other pages. We use a similar approach, where web pages correspond to documents, and links correspond to citations. To find a set of documents related to a document d, the Companion algorithm has two main steps. In step 1, we build the set $\mathcal{V}$, the vicinity of d, that contains the parents of d, the children of the parents of d, the children of d, and the parents of the children of d. This is the set of documents related to d. In step 2 we compute the degree to which the documents in $\mathcal{V}$ are related to d. To do this, we consider the documents in $\mathcal{V}$ and the citations among them as a graph, called the vicinity graph of d, which is illustrated in Figure 3.1. This graph is then processed by the HITS algorithm [62], which returns a degree of *authority* and *hubness* for each document in $\mathcal{V}$. Intuitively, a good authority is a document with important information on a given subject. A good hub is a document that cites many good authorities. Companion uses the degree of authority as a measure of similarity between d and each document in $\mathcal{V}$. More detailed descriptions of the Companion and HITS algorithms can be found in [27] and [62], respectively.

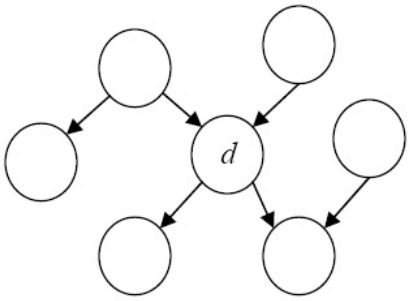


Figure 3.1: Vicinity graph of document d.

3.2 Combining Evidence

Table 3.1 lists each abovementioned type of evidence. For the ACM Digital Library collection (see Section 4.1.1 for more information about the collection), we applied bag-of-words, cosine, and Okapi to the content of the abstract, title, and abstract-plus-title (denoted as full) fields. And for the Cadê Web directory, bag-of-words, cosine, and Okapi were applied to the title-plus-body field (denoted as content).

Table 3.1: Types of Evidence.

Structural content-based evidence	ACM Digital Library	Abs_BOW	Bag of words similarity measure using abstract as content
		Abs_Cos	Cosine similarity measure using abstract as content
		Abs_Okapi	Okapi similarity measure using abstract as content
		Title_BOW	Bag of words similarity measure using title as content
		Title_Cos	Cosine similarity measure using title as content
		Title_Okapi	Okapi similarity measure using title as content
		Full_BOW	Bag of words similarity measure using title + abstract as content
		Full_Cos	Cosine similarity measure using title + abstract as content
		Full_Okapi	Okapi similarity measure using title + abstract as content
	Cadê	Content_BOW	Bag of words similarity measure using title + body as content
		Content_Cos	Cosine similarity measure using title + body as content
		Content_Okapi	Okapi similarity measure using title + body as content
Citation Evidence	ACM Digital Library & Cadê	Amsler	A measure of similarity that combines both co-citation and bibliographic coupling
		Bib_Coup	Two documents share one unit of bibliographic coupling if both cite a same paper
		Co-Cit	Two papers are co-cited if a third paper has citations to both of them
		Comp_Aut	Given a paper d, a set $\mathcal{V}$ of papers related to d can be obtained by examining its citations. Companion returns a degree of how related is the topic of each paper in this set $\mathcal{V}$ to the topic of paper d. The HITS algorithm processes documents in this set $\mathcal{V}$ and the citations among them, and returns a degree of authority and hubness for each document in $\mathcal{V}$.
		Comp_Hub	

3.2.1 Formal Definition of the GP-based Classification Problem

Each type of evidence discussed in the previous section will be represented as a document $\times$ document matrix and serves as the input to the GP-based classification framework. The matrix is defined as:

$$M_i = \begin{pmatrix} a_{11} & \dots & a_{1N} \\ \vdots & \ddots & \vdots \\ a_{N1} & \dots & a_{NN} \end{pmatrix}$$

where a_{ij} is the similarity between two documents d_i and d_j based on one type of similarity measure discussed in the previous section. GP will try to find a suitable non-linear function f to combine the matrices $M_1, M_2, \dots, M_n$, where n is the number of types of evidence. The computational output of the combination through such a non-linear function f is an output matrix defined as M_{GP}:

$$M_{GP} = f(M_1, M_2, \dots, M_n) \tag{3.7}$$

Figure 3.2 illustrates the expected role and discovery of the non-linear function f.

But M_{GP} is a matrix of similarities between pairs of documents. In order to use information represented in M_{GP} to predict the class label for a document in the classification task, we use a strategy based on a nearest neighbor classifier – *kNN* [120]. More about this will be discussed in Section 3.4.3. Classification-based measures like macro $F1$ (see Section 4.4) will be used to measure the effectiveness of the GP discovered non-linear function f. Relative to M_i, and in comparison to other results of fusion, M_{GP} is more accurate, denser, and when applied to *kNN*, it will produce better classification results for a given class C. The classification problem is illustrated in Figure 3.3.

3.2.2 Approach

Figure 3.3 provides a simplified example for our approach to combining different types of evidence in support of classification. We begin with a matrix giving document-to-document similarity values, for each type of evidence. In order to develop a classifier that will identify class C documents,

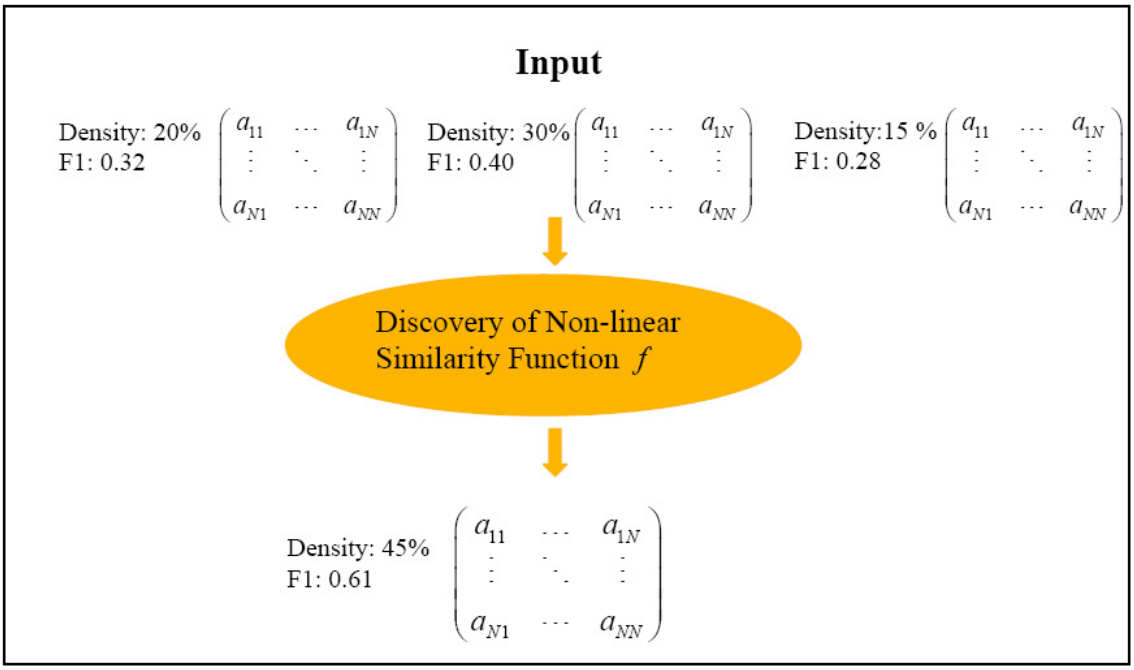


Figure 3.2: Illustration of matrix refinement through discovery of non-linear function f.

we must focus on documents in those matrices that have been identified in a training set, especially those in class C. Using training matrices, the GP algorithm allows us to discover a non-linear similarity function f that works with the matrices associated with each type of evidence. This function f, when applied to a document d_i of class C, ranks many documents from class C as similar to d_i and documents from non-class C as non-similar to d_i. As illustrated in this figure, document d_i is a class C document, as well as documents d_r, d_s, and d_t. d_i is not that similar to d_r, d_s, and d_t in each type of evidence, as shown in matrices M_1, M_2, and M_{14}. But in the new matrix generated by applying that function f, documents d_r, d_s, and d_t are much more similar to document d_i. On the other hand, document d_j is a class C document, while documents d_x, d_y, and d_z are non-class C documents. Document d_j is somewhat similar to documents d_x, d_y, and d_z in each type of evidence. But the new matrix represents the relationship between document d_j and documents d_x, d_y, and d_z much more accurately. So applying that function yields a new matrix, that can be used to develop a suitable kNN classifier. By evaluating the results of that classifier, we can gauge the quality of f, the function discovered through use of GP. Then, f and the kNN classifier that results from it can be used in the future to ascertain if new documents are in class C.

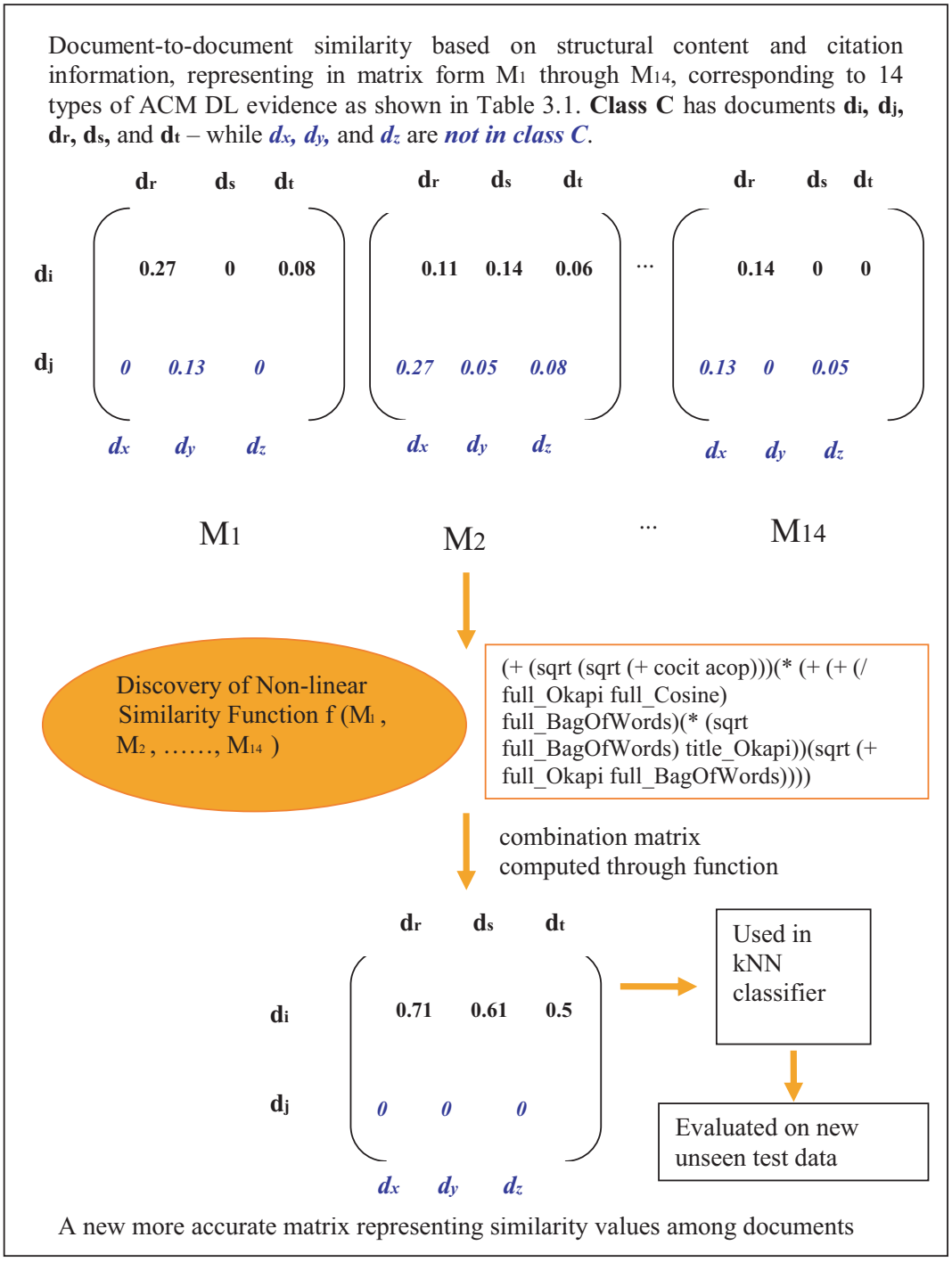


Figure 3.3: Example illustrating computations for combining evidence and classifying information.

3.3 Inductive Learning Method – GP

We have investigated machine learning [64] techniques as described below.

3.3.1 Machine Learning: Genetic Programming

Genetic Programming (GP), an extension of Genetic Algorithms (GAs), is an inductive learning technique designed following the principles of biological inheritance and evolution [64]. Genetic Programming has been widely used and proved to be effective in solving optimization problems, such as financial forecasting, engineering design, data mining, and operations management. GP makes it possible to solve complex problems for which conventional methods can not find an answer easily.

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

GP searches for an "optimal" solution by evolving the population generation after generation. It works by iteratively applying genetic transformations, such as reproduction, crossover, and mutation, to a population of individuals to create more diverse and better performing individuals in subsequent generations. The reproduction operator directly copies or, using a more appropriate term, clones some individuals into the next generation. The probability for an individual to be selected for reproduction should be proportional to its fitness. Therefore the better a solution solves the problem, the higher the probability it has to enter the next generation. While reproduction keeps the best individuals in the population, crossover and mutation introduce transformations, and so provide variations to enter into the new generation. The crossover operator randomly picks two groups of individuals, selects the best (according to the fitness) individual in each of the two groups as parent, exchanges a randomly selected gene fragment of each parent, and produces two "children". Thus, a "child" may obtain the best fragments of its excellent parents and so may surpass them, providing a better solution to the problem. Since parents are selected from a "competition",

good individuals are more likely to be used to generate offspring. The mutation operator randomly changes a gene code of an individual. Using these genetic operators, subsequent generations keep individuals with the best fitness in the last generation and take in “fresher air”, providing creative solutions to the target problem. Better solutions are obtained either by inheriting and reorganizing old ones or by lucky mutation, simulating Darwinian Evolution.

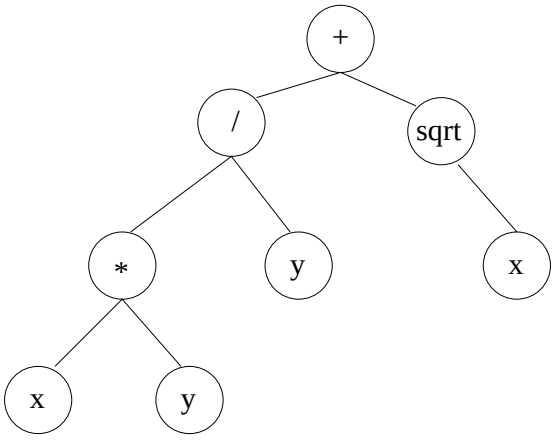


Figure 3.4: A sample tree representation.

In order to apply GP to the problem of classification, several required key components of a GP system need to be defined. Table 3.2 lists these essential components along with their descriptions.

3.3.2 Rationale of using GP for Text Classification

To fuse multiple sources of evidence for text classification, we could use techniques like manual selection and majority voting. However, manual selection and combing is very tedious and time-consuming, while simple majority voting may not work in all contexts. A better way is to use machine learning techniques to intelligently combine evidence, by optimizing a given objective function.

The decision to use GP for fusing multiple sources of evidence to discover similarity functions for text classification was motivated by four factors:

1. The large size of the search space:

Table 3.2: Essential GP Components.

Components	Meaning
Terminals	Leaf nodes in the tree structure, for example, x, y as in Figure 3.4.
Functions	Non-leaf nodes used to combine the leaf nodes. Commonly are numerical operations: +, -, *, /, log.
Fitness Function	The objective function GP aims to optimize.
Reproduction	A genetic operator that copies the individuals with the best fitness values directly into the population of the next generation without going through the crossover operation.
Crossover	A genetic operator that exchanges subtrees from two parents to form two new children. Its aim is to improve the diversity as well as the genetic fitness of the population. This process is shown in Figure 3.5.
Mutation	A genetic operator that replaces a selected individual's subtree whose root is a picked mutation point with a randomly generated subtree.

A similarity function can be represented in a tree structure. For example, to discover trees for the ACM Digital Library collection, we have used 14 terminals (see Section 3.2), four functions, trees of depth up to 16, and real constants which can vary between 0 and 1. Langdon et al. [68] show that for these parameters the search space for a tree is very large – a needle-in-a-haystack problem. Other search mechanisms like random search and exhaustive search would take inordinate amounts of time. GP has been shown to perform well under such conditions.

2. The characteristics of the objective function:

   Many performance measures in IR are discrete. GP does not require that objective functions be continuous in nature as long as it can distinguish good solutions from bad ones [64].

3. Previous success on the application of GP in the IR field as well as in data classification:

   Fan et al. previously reported success in applying GP to discover ranking functions for both individual queries (information routing tasks) [34, 35] and multiple queries (ad-hoc retrieval tasks) [37]. The success of GP in discovering nonlinear functions and structures in other data mining domains [64] also motivated us to use GP for similarity function discovery.

4. Little prior work on applying GP to large scale text classification:

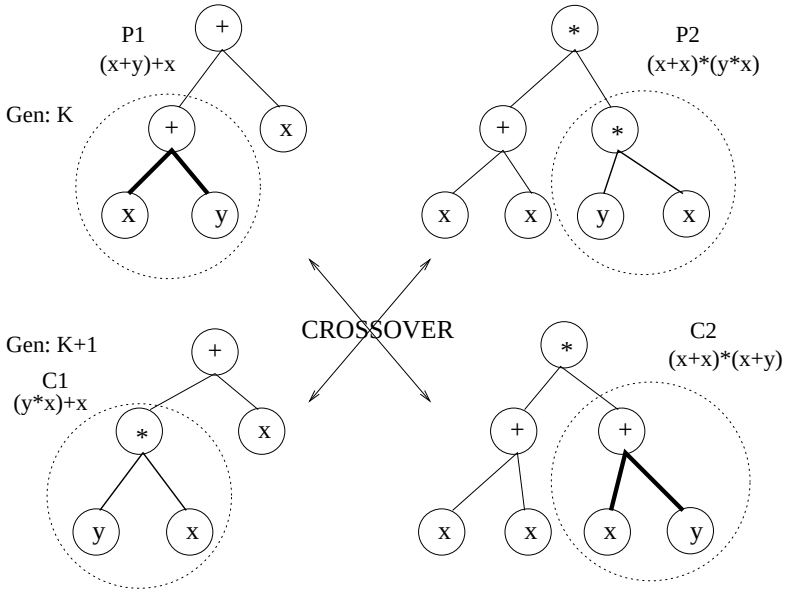


Figure 3.5: A graphical illustration of the crossover operation.

We seek to fill the near-void that exists in this area; our approach appears to have promise for broad impact.

3.4 Classification Framework

3.4.1 GP System Configurations

We set up the configurations of the GP system used for similarity function discovery as shown in Table 3.3.

Table 3.3: Modeling setup for classification function discovery by GP. Refer to Table 3.2 for explanations of the various components.

Terminals	We use features discussed in Section 3.2 as terminals.
Functions	+, *, /, sqrt
Fitness Function	We explore three different fitness functions: class recall, Macro $F1$, and $FFP4$. These fitness functions will be discussed in detail in Section 3.4.2.
Genetic Operators	Reproduction, Crossover, Mutation

3.4.2 Fitness Functions

The choice of fitness function can have a huge impact on the final classification performance [33]. So we perform experiments with different fitness functions. A good similarity function, i.e., a similarity function with a high fitness value, is one that, when applied to a document d_i of class C, ranks documents from class C in such a way that those with greater similarities to d_i are top-ranked. The higher the fitness value, the better the function.

Class Recall

Algorithm 1 below details a fitness evaluation function which GP should optimize within a particular class. This algorithm uses *kNN* to predict the class label for a document.

Algorithm 1 Class Recall Fitness Function Algorithm

```
Let R = 0
for each document D in class C do
    Find the k documents most similar to D
    Predict a class for D according to the kNN algorithm (Section 3.4.3) using the k documents
    as the k nearest neighbors
    if this is a correct prediction then
        R = R + 1
    end if
end for
Let F = R/|C|
```

Macro $F1$

Algorithm 2 details a fitness function where class labels of the most similar $|C|$ documents to a document from class C are checked.

FFP4

We also perform experiments using a utility-theoretic based fitness function called FFP4 [33]. According to utility theory, there exists a utility function (a user's preference function) that assigns a utility value (the gained value from a user's perspective) for each document. These values vary

Algorithm 2 Macro $F1$ Fitness Function Algorithm

Let L_p, L_r be empty lists
for each document D in class C **do**
  Let $L_p = L_p \cup$ (the set of $|C|$ documents most similar to D)
  Let $L_r = L_r \cup$ (the set of $|C|$ documents most similar to D and also not already in L_r)
end for
Let $p =$(no. of documents in L_p that are of class C)$/|L_p|$
Let $r =$(no. of documents in L_r that are of class C)$/|L_r|$
$F = \frac{2pr}{p+r}$

from document to document. In general, we assume the utility of a similar document decreases with its ranking order. More formally, given a utility function $U(x)$, and two ranks x_1, x_2, with $x_1 < x_2$, according to this assumption, we expect the following condition to hold:

$$U(x_1) > U(x_2) \tag{3.8}$$

Eq. (3.8) is called the order preserving condition. For example, suppose the top two similar documents to a document D belonging to class C are both from class C. A relevant document to a document D is one that comes from the same class as document D. According to condition (3.8), the first relevant document in the rank list should have greater utility than the second one. The further down the rank list, the less utility a relevant document gives. The following equation defines $FFP4$:

$$Fitness_{FFP4} = \sum_{i=1}^{|D|} r(d_i) * k_8 * {k_9}^i \tag{3.9}$$

where $r(d) \in (0, 1)$ is the relevance score assigned to a document, it being 1 if the document is relevant and 0 otherwise. $|D|$ is the total number of similar documents to a document D. For each document in a class C, the fitness value is calculated based on Eq. (3.9), and then the final fitness value of a solution is obtained through averaging the fitness value for each document. k_8 and k_9 are scaling factors and will be decided through exploratory analysis. Algorithm 3 below details the $FFP4$ fitness function.

Algorithm 3 $FFP4$ Fitness Function Algorithm

Let $F = 0$
for each document D in class C **do**
$Fitness\_D$ = $FFP4$ fitness value calculated based on the set of $|D|$ documents that are similar to D
$F+ = Fitness\_D$
end for
$F = F/|C|$

3.4.3 Framework

Along with the settings discussed in the previous section, the overall classification framework has 5 steps, as follows:

1. For each class, generate an initial population of random trees, each representing a similarity function.
2. For each class, perform the following sub-steps on training documents for N_{gen} generations.
   (a) Calculate the fitness of each similarity tree.
   (b) Record the top N_{top} similarity trees.
   (c) Create a new population by: reproduction, crossover, and mutation.
3. Apply the recorded (N_{gen} * N_{top}) candidate "similarity trees" to a set of validation documents, and select the best performing tree b_C as the unique best discovered similarity tree for each class C.
4. For each class C, use b_C as similarity function in a *kNN* classifier (see below) and apply this resulting classifier to a set of testing documents.
5. Combine the output of each classifier through a simple majority voting.

Steps 1, 2, and 3 concern the training process within GP which intends to discover better similarity functions for each class. However, the discovered functions only can be used to calculate the similarity between a pair of documents. In order to evaluate the performance of those functions in the classification task, we used a strategy based on a nearest neighbor classifier – *kNN* [120]. This classifier assigns a category label to a test document, based on the categories attributed to the

k most similar documents in the training set. The *kNN* algorithm was chosen since it is simple and makes direct use of similarity information. Figure 3.5 shows a flow-chart representation of the classification framework.

In the *kNN* algorithm, to a given test document d is assigned a relevance score $s_{c_i,d}$ associating d with each candidate category c_i. This score is defined as:

$$s_{c_i,d} = \sum_{d' \in \mathcal{N}_k(d) \wedge class(d') = c_i} similarity(d, d') \tag{3.10}$$

where $\mathcal{N}_k(d)$ are the k nearest neighbors (the most similar documents) of d in the training set. In Step 4 of our framework the generic similarity function of *kNN* is replaced by the functions discovered by GP for each class.

In multi-classification problems with n classes, we effectively end up with n *kNN* classifiers using the described framework. In order to produce a final classification result, we combine the output of all n classifiers using a simple *majority voting* scheme, whereby the class of a document d_i is decided by the most common class assigned by all the n classifiers. In case of ties, we assign d_i to the larger class. Because of its simplicity we chose to use majority voting in our framework (Step 5) to: 1) help alleviate the common problem of overfitting found in GP training [35] and; 2) help boost performance by allowing *kNN* classifiers to apply different similarity functions which explore and optimize the characteristics of each particular class in different ways.

3.4.4 Theoretical and Practical Properties of the Framework

The classification framework suffers from scalability, a common problem with GP. That is, it takes a relatively long time to find a good solution. In fact, the time complexity of an experiment based on the above framework is O(N_{gen} * N_{ind} * T_{eval}), where N_{gen} is the number of generations for evolution, N_{ind} is the number of individuals in a population pool, and T_{eval} is the fitness evaluation time for an individual. Since T_{eval} is determined by the complexity of an individual (Size) and the number of training samples ($N_{samples}$), then the total time complexity of an experiment is O(N_{gen} * N_{ind} * Size * $N_{samples}$) [37, 33]. Scalability and computing efficiency are not the main issue in our experiments; we focus more on the effectiveness of the classifiers. Yet, the speed of the learning process can improve through parallel processing as well as through optimization, e.g., as described in Section 4.2. Further, in a practical application, the learning process need occur only occasionally; the frequent operation will be the actual classification of incoming new documents.

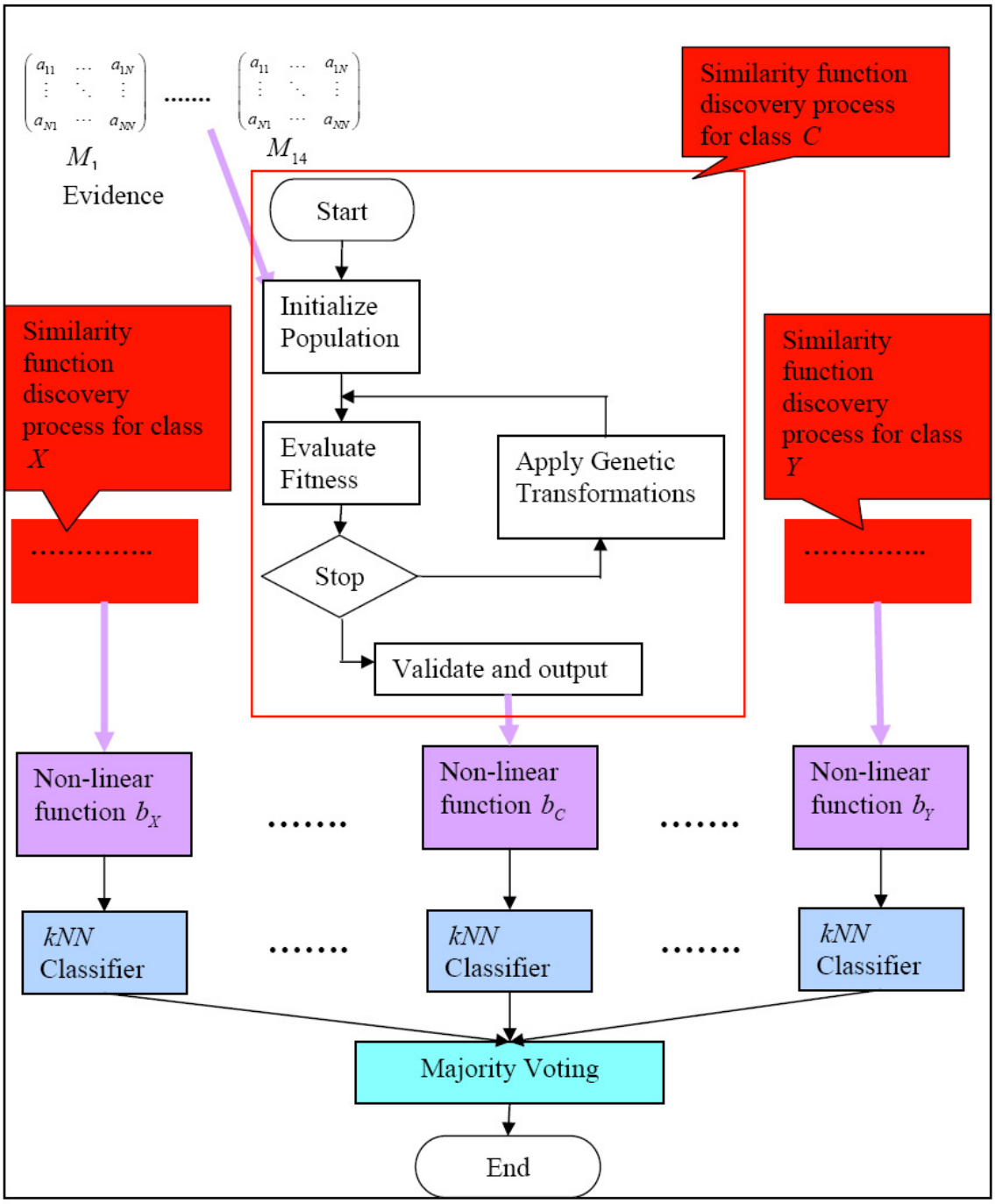


Figure 3.6: Flow-chart representation of the classification framework.

Chapter 4

Experimental Design

In this chapter, we discuss the data sets that are used to test the hypothesis that GP is able to find the best similarity functions. We also describe the design of the experiments that validates the effectiveness and generalizability of our classification framework.

4.1 Test Data Collection

We use learning resources and the Web as test data collections. The rationale for using these two collections as our test data is the generality and wide applicability of these data and diverse relationships found in these cases.

4.1.1 Test data from learning resources (ACM Digital Library)

The ACM Digital Library is a sub-collection of CITIDEL that was used as one of our testing document collections. This collection offers a number of challenges and opportunities in classification [66, 124]. The complex structure, including multiple fields in each metadata record, allows using structural information (e.g., citation-based similarities) for classification. On the other hand, the ACM Digital Library suffers from most of the problems mentioned in Section 1.2. For example, considering the ACM Computing Classification System (CCS, http://www.acm.org/class/1998/), only 42.3% of the records in the first level and only 41.7% of the records in the second level have abstracts; this makes it very hard to classify them using traditional content-based classifiers. For

these records, the only available textual content is title, and titles normally contain only 5 to 10 words. Citation information was created with OCR and has a significant number of errors. A very imprecise process of matching between the citation text and the documents, using adaptations of techniques described in [69], had to be performed. This introduced noise and incompleteness in the citation-based similarity matrices computed with measures such as co-citation or bibliographic coupling. Other concerns were the large search space and skewed distributions observed for some categories.

4.1.2 Test data from the Web

We also test our classification framework using a Web category system. This helps to make sure the obtained classifiers work not only on one specific collection but also on other collections [13] like the hierarchical Web documents collection. In particular, we performed experiments using a set of classified Web pages extracted from the Cadê Web directory (http://www.cade.com.br). This directory points to Brazilian Web pages that were classified by human experts. It is a subset of the WBR-99 collection, which was built from a set of documents collected from the Brazilian Web in November 1999. The WBR-99 collection is available to scholars at no cost for research in Information Retrieval problems. Experiments with this collection have been used in several doctoral and masters studies, and are published in works such as [12] and [102]. We constructed two sub-collections using the data available in Cadê: Cade12 and Cade188. Cade12 contains 44,099 documents belonging to the 12 first level categories (Computers, Culture, Education, Health, Internet, News, Recreation, Science, Services, Shopping, Society, and Sports). Cade188 is a subset of Cade12, without the documents classified only in the first level category. Cade188 contains 42,004 documents belonging to 188 second level categories of Cadê (Biology, Chemistry, Dance, Music, Schools, Universities, etc.). Similar to the ACM collection, this Cadê collection also has skewed distributions as shown later in Section 4.2.

Table 4.1 summarizes the characteristics for the two test-bed collections.

4.2 Sampling Strategy to Ensure Efficiency

A common problem with GP is scalability. We explore different sampling strategies to speed up the learning process.

Table 4.1: Testbed Collection Characteristics.

Testbed Collection	Category	No. of Documents	No. of Categories
ACM Digital Library	First level	30,022	11
	Second level	18,664	44[a]
Cadê	First level	44,099	12
	Second level	42,004	188

[a]Some extra-small CCS second level categories with less than 50 documents are dropped from the experiments.

4.2.1 Stratified Random Sampling

The ACM Digital Library collection used in our experiments has 30,022 documents belonging to 11 categories for the first level and 18,664 documents belonging to 44 categories for the second level. Figure 4.1 shows the distributions for the first level and second level categories, respectively.

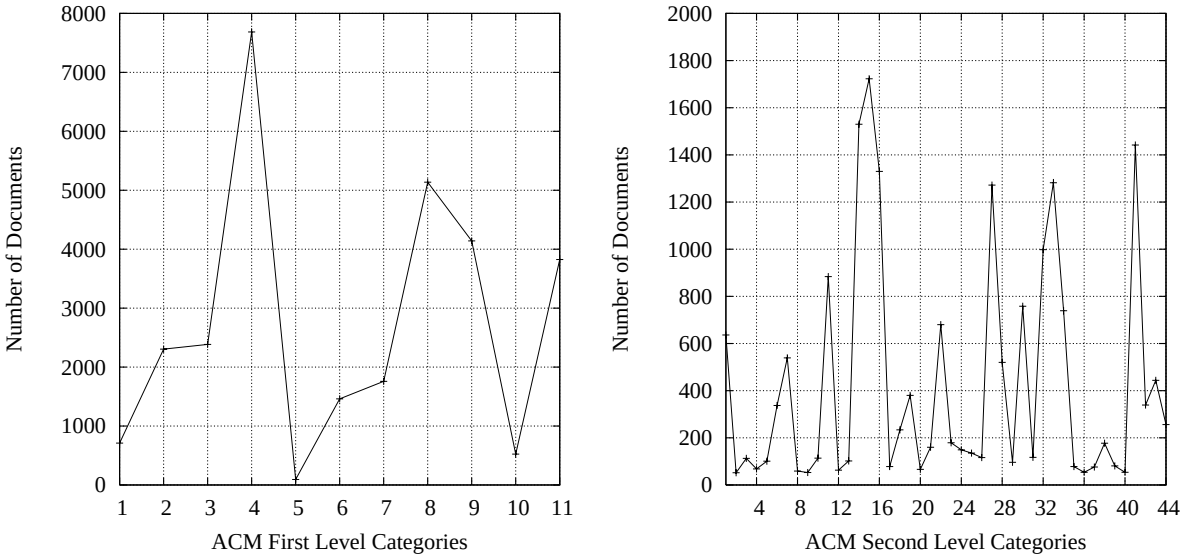
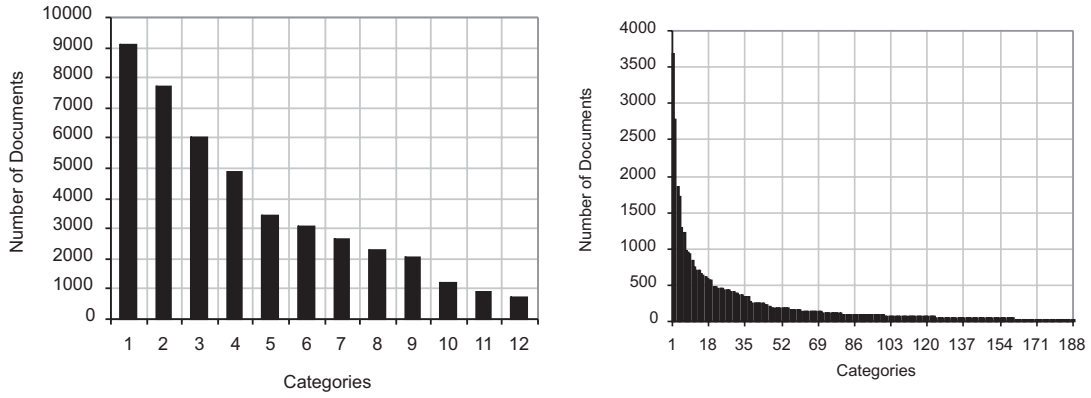


Figure 4.1: Compared distributions for the first and the second level ACM collections.

The Cadê Web collection also has a skewed distribution of categories for both levels. From Figure 4.2 we can see that more than 50% of all documents come from the three most popular categories for Cade12. The most popular category, *Services*, has 9,081 documents while the least popular, *Shopping*, has only 715 documents. For Cade188, 50% of the documents are in just 10% of the categories. The most popular category, *Society: People*, has 3,675 documents while the least popular, *Internet: Tutorials*, has only 24 documents.

Each terminal or feature described yields a similarity matrix which contains the similarity between



(a) Cade12 (b) Cade188

Figure 4.2: Compared distributions for the Cade12 and Cade188 collections.

each pair of documents. Using half or more of the whole collection as our training data, the required resources – in CPU time and amount of memory – would be enormous. The time required to discover a proper classification framework also would be significant. To reduce the high cost of resources and at the same time improve efficiency, sampling was used. So, for the ACM collection, we start by considering the documents belonging to each category of the ACM CCS as different population strata. We then randomly select from each stratum a given number of units based on a proportion, like 15%, for each category. However, special attention was paid to skewed categories. For example, category E in the first level only has 94 documents while the average size of the other categories is in the range of thousands. 15% of 94 only gives us 14 documents, which is too small to serve as a sample to discover the whole category's characteristics. The number of documents for some second level categories is too small (< 50) for classification purposes so those categories were dropped from the experiments. Also, because of its size, category E was not subdivided in the second level and all of its ACM sub-categories were considered as a unique second-level entry (denoted E.x). Classification statistics for both the first level and the second level collection were used to control the sampling procedure. That is, baselines from the samples for each level were compared with baselines for that level's corresponding whole collection to ensure that the samples mirror that level's whole collection as well as possible. The same random stratified sampling strategies was applied to the Web category test data set as well.

4.2.2 Active Sampling

Textual data are often very noisy, and the documents selected by random sampling often deviate from the true class distribution. In the active sampling strategy, based on the active learning literature [75, 105, 96], training data samples are actively selected from a large pool of data. The idea is that we create a class profile using the centroid vector of all training documents in a particular class J. Then, based on the cosine similarity measure, we compute the similarity of the class profile with all out-class documents (not belonging to class J) in the pool data and the top N out-class documents are selected for classification purposes. These N documents are the most similar out-class documents, thus are assumed to be the most difficult ones to classify (and so should be most helpful, as occurs with the Ide dec-hi algorithm used in relevance feedback [55]). This sampling strategy is conceptually very similar to the "query zone" idea where only the top N documents based on the match with a given query are selected for learning experiments [103]. This technique has been shown to be effective in query learning [103] and ranking function optimization [33, 36, 35, 37]. We apply active sampling to the GP machine learning experiments.

4.3 Experimental Data Set Design

We follow a three data-sets design [35, 36, 78] in our experiments. We randomly split the data into training, validation, and test parts. The introduction of the validation data set is to help alleviate the problem of overfitting of GP on the training data and select the best generalizable similarity function. Overfitting is a common problem encountered in many machine learning and data mining techniques. It occurs when the learned or evolved model fits the particulars of the training data overly well and consequently does not generalize to new unseen data.

We generate two sets of training samples using a stratified sample strategy for both the first level and the second level in the ACM CCS. The first set uses a random 15% sample for large classes and 50% for skewed classes. The second set uses a random 30% sample for large classes and 50% for skewed classes. For the 15% (50% for skewed classes) training sample, a random 10% (25% for skewed classes) sample of the collection is used for validation and the rest of the samples are used for testing. Correspondingly, for the 30% (50% for skewed classes) training sample, a random 20% (25% for skewed classes) sample of the collection is used for validation and the rest of the samples are used for testing. All approaches reported in later sections use the same training and test sets. Results will be evaluated based on test data sets only. In the remainder, we

will use 15% to refer to the first sample set and 30% to refer to the second sample set for each level, respectively. Recently there has been research investigating the use of hierarchies for text classification [26, 31, 63, 51, 116]. But we perform the ACM second level experiments to further verify that GP-based classification works on different levels in a single collection. Hierarchical classification is not the focus of this work.

In order to test if the GP-based classification framework works on different collections, we perform experiments in a Web test collection. Similar to what we did for the ACM DL collection, we generate two sets of training samples using the stratified sample strategy for both the first level and the second level in the Cadê Web directory. Following the same design as above, the first set uses a random 15% sample for large classes and 50% for skewed classes. One slight difference for the second set is that instead of using a random 30% sample for large classes, we use 20% in order to make the GP training process more time efficient.

4.4 Baselines

In order to demonstrate that the combination of different features by GP is able to provide better classification results, we need to compare it with the classification statistics of each feature in isolation (baselines). We use the commonly used $F1$ measure as the comparison criteria. The $F1$ measure was first introduced by van Rijsbergen [94]. It is a combination of precision and recall. Precision p is defined as the proportion of correctly classified records in the set of all records assigned to the target class. Recall r is defined as the proportion of correctly classified records out of all the records having the target class. $F1 = \frac{2pr}{p+r}$. It is worth noticing that $F1$ is a balanced combination of precision and recall. It reduces the risk that you can get perfect precision by always assigning zero categories, or perfect recall by always assigning every category. The result we want is to assign the correct categories and only the correct categories, maximizing precision and recall at the same time, thereby maximizing $F1$.

To get a single performance value for the entire classification problem, there are two ways to compute the overall $F1$ measure by two different types of averages, *micro-average* and *macro-average*. In micro-averaging, the $F1$ measure is computed globally over all categories. It gives equal weight to each document and therefore is considered as an average over all the document/category pairs. In [121], Yang and Liu discussed that micro-averaged $F1$ tends to be dominated by the classifiers' performance on common categories. In macro-averaging, the $F1$ measure is computed locally on

a per-category basis, then averaged over categories. Macro-averaged $F1$ gives equal weight to each category. Yang and Liu [121] discussed that macro-averaged $F1$ is influenced more by the classifiers' performance on rare categories.

4.5 Experimental Set Up

We ran several experiments on the training samples using different parameters. Particularly, we noticed that a larger population size and different rate for the genetic transformation operations like crossover, reproduction, and mutation produce better results. On the other hand, they have a huge effect on the training time. Based on exploration in some preliminary experiments, we use the settings shown in Table 4.2 as our GP system experimental setting. And we only report performance of the best tree in the validation sets applied to the test sets.

Table 4.2: GP system experimental settings.

Population size	400
Crossover rate	0.65
Mutation rate	0.05
Reproduction rate	0.30
Generations	30
No. of seeds	5 (maximum)

Chapter 5

Experimental Results

We performed experiments on the two test bed collections introduced in Chapter 4. Different fitness functions and different sampling strategies were tested. We report detailed experimental results in this chapter. Same training sets are used for different approaches under comparison. Results reported in this chapter are based on both test data sets.

We demonstrate the effectiveness of our classification framework in several ways: 1) by comparing its performance against the best baselines per class in isolation; 2) by comparing it against a majority voting of classifiers using those best baselines as similarity functions; 3) by comparing our experimental results with the results achieved through a linear combination of both the content-based and structure-based information through SVM; and 4) by comparing our experimental results with the results achieved through a content-based SVM classifier[1]. While the last comparison may seem inappropriate since the classifiers are trained and applied to different types of content, it does provide a good idea of the core performance of our method, clearly showing it as a valid alternative in classification tasks similar to the ones used in this work.

The SVM classifier has been extensively evaluated for text classification on reference collections, thus offering a strong baseline for comparison. Joachims et al. [57] also have shown how to combine different similarity measures by means of composite kernels. Their approach consists of combining simple and well-understood kernels by a series of "kernel preserving" operations, hence constructing an increasingly matching feature space S. In order to apply their method, we started by noticing that each one of our types of evidence can be represented as positive document-by-

[1]For content we used a concatenation of title + abstract for the ACM DL collection, and a concatenation of title + body for the Web collection.

document matrices, like those presented in Section 3.2. So, we can represent each type of evidence as a kernel matrix $\mathcal{K}_{evidence}$ with elements $\mathcal{K}_{evidence}(d_i, d_j)$, where d_i and d_j are document vectors. The kernel matrix for our final feature space S is obtained by means of a linear combination of our initial kernel matrices, as described in Eq. (5.1).

$$\mathcal{K}_{combined}(d_i, d_j) = \frac{1}{N} \sum_{\forall evidence} \mathcal{K}_{evidence}(d_i, d_j) \tag{5.1}$$

where N is the number of distinct kinds of evidence used. Finally, the classification decisions for the combined kinds of evidence are obtained by applying a linear SVM classifier in our final feature space S.

For completeness, we also compare our classification framework against a classifier using a simple linear fusion of evidence as the similarity function and against a classifier using the similarity function discovered through GA. The similarity function used in linear fusion is simply represented through a summation of all the evidence, while the similarity function discovered through GA is represented by a weighted linear combination of all evidence, where the weights are assigned by the GA training process.

5.1 Experiments on ACM Digital Library

Experiments were performed on the ACM DL using CCS under both the first and the second level.

5.1.1 Baselines

Tables 5.1 and 5.2 show the type of evidence that performs the best, according to the $F1$ measure, when applied to a *kNN* algorithm for a specific category in isolation in the test collections for the first level and the second level, respectively. Tables 5.3 and 5.4 show the macro-averaged F1 and micro-averaged F1, respectively, over all categories for both the first level and the second level for each similarity evidence, also in isolation.

From Tables 5.1 and 5.2 it can be seen that full-based evidence is most important for the majority of the classes. For those classes whose best performer was citation-based evidence, 1) Amsler had the highest value for category I.3 for both the 15% and 30% samples; 2) Amsler, Bib_Coup, and

Table 5.1: Best baseline for each category (first level).

Class	F1/ class(15%)	Best Evidence (15%)	F1/ class(30%)	Best Evidence (30%)
A	27.76	Full_BOW	37.31	Title_BOW
B	70.34	Full_Okapi	73.55	Full_Cos
C	67.15	Full_Okapi	67.53	Full_Okapi
D	73.97	Full_Okapi	77.38	Full_Okapi
E	43.75	Full_Okapi	27.59	Full_Cos
F	51.59	Full_Okapi	56.96	Full_Cos
G	67.75	Full_Cos	70.88	Full_Cos
H	72.83	Full_Okapi	72.59	Full_Okapi
I	65.97	Full_Okapi	67.86	Full_Okapi
J	22.34	Full_Cos	20.12	Full_Okapi
K	73.19	Full_Okapi	73.74	Full_Okapi

Comp_Hub were the most frequent evidence for the 15% sample; and 3) Amsler and Comp_Aut were the most frequent evidence for the 30% sample. One interesting result we noticed from Table 5.1 is that for class A, the best type of evidence is bag-of-words based ones while for the other classes, it is cosine or Okapi based ones. This is because the incomplete information problem of the ACM digital library collection is the worst in this class. Recall from Section 4.1.1 that only 42.3% of the records in the first level have abstracts. However, in class A, only 12.8% of the records in the first level have abstracts. Thus, given the much more scarce textual content in this class, bag-of-words similarity measure might give the best performance.

From Tables 5.3 and 5.4, it can be seen that the best types of evidence are the full-based ones, followed by title-based, citation-based, and abstract-based evidence, respectively. This should be expected since: full is the evidence which appears in all the documents and contains the most complete information, the titles are relatively short, many documents are missing abstracts, and the information provided by the citation structure is very incomplete and imprecise.

5.1.2 Results

The results reported in this subsection were based on the stratified random sampling strategy and class recall fitness function. Experiments also were performed using active sampling and other fitness functions and those results are reported in the next two subsections.

In a comparison, class by class, for the first level, between the majority GP (Table 5.5 column 5 and

Table 5.2: Best baseline for each category (second level).

Class	F1/ class(15%)	Best Evidence (15%)	F1/ class(30%)	Best Evidence (30%)
A.0	48.05	Full_Okapi	55.57	Full_Okapi
A.1	***9.52***	Full_BOW	***0.00***	-[a]
B.1	0.00	-[a]	44.44	Full_Okapi
B.3	28.13	Abs_Okapi	39.13	Full_Okapi
B.5	24.00	Bib_Coup	31.25	Abs_Cos
B.6	53.53	Full_Cos	55.56	Full_Cos
B.7	56.29	Full_Okapi	56.19	Full_Okapi
B.8	***15.38***	Bib_Coup	***9.52***	Full_Okapi
C.0	***0.00***	-[a]	***12.50***	Full_Okapi
C.1	21.05	Full_Okapi	27.78	Full_Cos
C.2	68.29	Full_Okapi	70.91	Full_Okapi
C.4	***5.88***	Comp_Hub	***6.67***	Comp_Hub
D.1	***17.54***	Full_Okapi	***17.78***	Full_Cos
D.2	59.10	Full_Okapi	60.91	Full_Okapi
D.3	65.37	Full_Okapi	66.37	Full_Okapi
D.4	64.19	Full_Okapi	65.57	Full_Okapi
E.x	24.24	Full_Okapi	44.44	Title_Cos
F.1	37.59	Bib_Coup	38.33	Comp_Aut
F.2	24.29	Amsler	37.26	Amsler
F.3	***14.29***	Comp_Aut	***22.86***	Comp_Aut
F.4	32.14	Amsler	43.18	Full_Okapi
G.1	66.94	Full_Cos	69.46	Full_Cos
G.2	23.30	Abs_BOW	44.04	Full_Cos
G.3	37.50	Abs_BOW	45.16	Comp_Aut
G.4	17.65	Comp_Hub	32.50	Full_Okapi
H.1	12.99	Full_Cos	8.33	Amsler
H.2	71.44	Full_Cos	75.07	Full_Cos
H.3	56.01	Full_Okapi	61.07	Full_Okapi
H.4	26.67	Full_Okapi	32.43	Title_Okapi
H.5	50.30	Full_Cos	54.88	Full_Okapi
I.1	40.00	Full_Cos	43.33	Full_Cos
I.2	59.77	Full_Okapi	66.44	Full_Okapi
I.3	69.42	Amsler	70.58	Amsler
I.6	72.33	Full_Okapi	72.97	Full_Okapi
I.7	32.65	Full_Okapi	25.00	Title_Cos
J.1	***10.00***	Comp_Hub	***0.00***	-[a]
J.3	45.00	Full_Cos	60.61	Full_Cos
J.5	18.35	Full_Cos	13.51	Title_Cos
J.6	32.00	Full_Cos	55.81	Full_Okapi
K.1	***33.33***	Title_BOW	***21.05***	Title_BOW
K.3	75.37	Full_Cos	76.03	Full_Cos
K.4	19.94	Full_Cos	27.15	Title_BOW
K.6	23.42	Full_Okapi	24.92	Full_Okapi
K.7	20.00	Full_BOW	38.98	Full_Okapi

[a]Here each evidence has the 0 F1 value for this category thus there is no best baseline.

Table 5.3: Macro F1 on individual evidence.

Evidence	First Level		Second Level	
	(15%)	(30%)	(15%)	(30%)
Abs_BOW	18.31	24.22	11.36	9.50
Abs_Cos	32.41	34.97	18.80	20.82
Abs_Okapi	33.21	34.14	18.81	19.49
Full_BOW	40.01	45.76	22.87	23.31
Full_Cos	53.40	55.80	32.31	36.39
Full_Okapi	**57.17**	**57.37**	**32.63**	**36.64**
Title_BOW	44.87	49.34	26.27	30.39
Title_Cos	49.10	51.58	28.45	34.01
Title_Okapi	48.36	52.08	28.40	33.98
Bib_Coup	30.85	33.96	17.84	20.49
Amsler	36.90	41.09	21.21	24.41
Co-Cit	21.73	26.88	12.71	15.06
Comp_Aut	31.43	35.77	17.39	21.33
Comp_Hub	33.40	36.85	19.02	22.24

Table 5.4: Micro F1 on individual evidence.

Evidence	First Level		Second Level	
	(15%)	(30%)	(15%)	(30%)
Abs_BOW	29.35	33.94	24.86	23.76
Abs_Cos	42.09	44.95	35.32	37.20
Abs_Okapi	43.39	44.57	35.99	37.37
Full_BOW	54.48	59.04	44.88	45.61
Full_Cos	65.25	69.67	55.57	59.51
Full_Okapi	**69.25**	**71.24**	**57.47**	**61.42**
Title_BOW	58.72	62.68	48.30	52.31
Title_Cos	60.84	64.96	51.42	55.56
Title_Okapi	61.42	65.55	50.99	55.93
Bib_Coup	42.28	46.13	31.55	36.55
Amsler	50.27	54.47	38.43	43.29
Co-Cit	30.45	35.89	22.82	27.59
Comp_Aut	42.85	47.94	31.84	37.21
Comp_Hub	46.03	49.80	34.12	39.17

Table 5.5: Macro F1 comparison between Majority Voting using the best evidence per class in isolation, Linear Fusion, GA, combined majority GP, content-based and combination-based SVM for the first level on the 15% sample.

Class	Majority Best Evidence	Linear Fusion	GA	Majority GP	Content-based SVM	Combination-based SVM
A	24.54	22.01	20.70	25.36	32.22	26.36
B	69.48	66.99	70.44	78.48	68.40	69.80
C	67.95	66.96	68.34	72.18	65.30	64.93
D	68.14	70.73	72.98	76.55	72.59	74.04
E	34.48	24.24	29.41	37.84	14.81	12.34
F	48.32	46.81	50.50	58.80	50.39	54.11
G	66.81	68.38	69.89	73.55	65.69	68.21
H	71.37	69.37	71.65	77.56	71.73	71.22
I	60.25	62.45	65.14	71.62	64.59	64.75
J	16.39	24.19	24.22	29.54	11.59	13.55
K	72.63	71.22	73.15	76.09	70.50	72.55
Macro F1	54.58	53.94	56.04	**61.60**	53.44	53.80

Table 5.6: Macro F1 comparison between Majority Voting using the best evidence per class in isolation, Linear Fusion, GA, combined majority GP, content-based and combination-based SVM for the first level on the 30% sample.

Class	Majority Best Evidence	Linear Fusion	GA	Majority GP	Content-based SVM	Combination-based SVM
A	34.68	31.81	31.13	35.51	43.43	35.52
B	74.66	72.35	74.27	79.90	71.78	71.79
C	69.22	68.11	69.04	71.45	66.55	65.74
D	78.04	77.66	78.87	81.93	78.34	77.95
E	22.22	20.00	20.00	21.62	0.00	15.39
F	57.42	55.89	58.23	64.58	56.95	55.30
G	72.77	72.53	75.03	77.29	71.19	70.05
H	73.10	70.48	72.78	77.86	72.54	72.49
I	67.86	66.95	69.83	74.27	67.17	65.42
J	14.01	21.71	21.84	29.03	24.86	24.65
K	74.73	72.20	73.36	75.81	73.19	72.14
Macro F1	58.07	57.24	58.58	**62.66**	56.91	56.95

Table 5.7: Micro F1 comparison between Majority Voting using the best evidence per class in isolation, Linear Fusion, GA, combined majority GP, content-based and combination-based SVM for the first level.

	15%	30%
Majority Best Evidence	66.56	72.54
Linear Fusion	66.67	70.95
GA	68.77	72.66
Majority GP	**73.78**	**76.49**
Content-based SVM	67.98	71.88
Combination-based SVM	67.83	70.76

Table 5.6 column 5) and the best evidence in isolation (Table 5.1), the majority GP outperforms the best evidence in most of the classes in both samples (only the performance for class A and E are worse). It also can be seen from Tables 5.8 and 5.9 (column 5) and 5.2 that this is true for most of the classes in the second level. There are two major reasons that the majority GP didn't excel in a few classes (like classes A.1, B.8, C.0, C.4, D.1, F.3, J.1, and K.1) in the second level. One is that the numbers of documents belonging to these classes are too small, so GP could not discover enough characteristics of those classes for classification purposes. For example, class A.1 only has 52 documents while class C.0 only has 53 documents. The other reason is that the best baselines for those classes (see Table 5.2) are poor, as compared with other classes. This means that we do not have good types of evidence for those classes.

Using macro-averaged F1 as the comparison measure, when comparing the majority GP against the majority using the best evidence on the first level (Tables 5.5 and 5.6, between columns 2 and 5, respectively), it is clear that majority GP presents better performance: we obtain a gain of 12.86% in the 15% sample and 7.90% in the 30% sample. This is also true for the second level (Tables 5.8 and 5.9, between columns 2 and 5, respectively): we obtain a gain of 10.05% in the 15% sample and 10.42% in the 30% sample. If we compare the micro-averaged F1 of the majority GP with that of the majority best evidence (Table 5.7 between row 2 and row 5), we also can see that majority GP shows better performance: we obtain a gain of 10.85% in the 15% sample and 5.45% in the 30% sample for the first level. And we obtain a gain of 4.73% in the 15% sample and 3.76% in the 30% sample for the second level (Table 5.10 between row 2 and row 5).

When majority GP is compared with the combination-based SVM classifiers, the gains are even higher: Table 5.11 shows the percentage of gains based on macro-averaged $F1$ (Tables 5.5 and 5.6 between columns 5 and 7) and micro-averaged $F1$ (Table 5.7 between rows 5 and 7).

Table 5.8: Macro F1 comparison between Majority Voting using the best evidence per class in isolation, Linear Fusion, GA, combined majority GP, content-based and combination-based SVM for the second level on the 15% sample.

Class	Majority Best Evidence	Linear Fusion	GA	Majority GP	Content-based SVM	Combination-based SVM
A.0	47.43	46.56	46.27	48.55	57.41	57.43
A.1	0.00	0.00	0.00	***0.00***	11.11	0.00
B.1	36.84	36.11	39.44	44.19	21.62	21.33
B.3	28.17	26.47	26.47	30.00	20.00	26.47
B.5	26.97	19.07	18.71	25.64	16.13	20.69
B.6	45.91	42.16	40.73	45.09	26.84	42.54
B.7	57.31	52.86	50.21	59.56	45.29	51.18
B.8	5.41	0.00	0.00	***5.56***	4.16	0.00
C.0	0.00	0.00	0.00	***0.00***	0.00	0.00
C.1	28.57	19.28	24.39	24.24	14.71	16.92
C.2	71.43	68.48	66.52	73.62	69.02	70.70
C.4	0.00	0.00	0.00	***0.00***	0.00	0.00
D.1	10.34	11.76	8.20	***12.35***	14.63	13.74
D.2	57.95	57.32	54.60	63.58	55.08	56.90
D.3	63.90	63.20	61.73	70.72	59.35	64.20
D.4	66.38	62.65	61.07	68.65	59.85	61.11
E.x	17.14	25.64	16.22	35.00	7.14	13.12
F.1	34.89	29.25	26.47	33.73	24.78	27.35
F.2	21.36	15.05	16.36	26.01	10.17	18.84
F.3	5.26	0.00	0.00	***8.51***	0.00	0.00
F.4	35.94	29.27	30.40	30.88	37.33	28.77
G.1	67.57	67.88	65.76	73.54	62.33	66.38
G.2	22.03	20.47	19.05	23.18	23.12	21.10
G.3	40.00	30.87	31.79	31.37	45.98	35.18
G.4	23.14	16.67	13.79	17.39	13.43	21.66
H.1	10.53	9.52	6.45	13.70	6.78	4.21
H.2	66.24	66.21	68.06	78.40	76.62	75.18
H.3	59.81	58.07	55.43	63.89	51.26	59.11
H.4	15.69	24.24	26.67	31.43	5.27	9.68
H.5	49.61	41.33	42.08	54.79	45.78	41.91
I.1	39.18	28.32	28.57	43.37	29.63	23.64
I.2	61.93	55.91	54.84	65.19	54.59	52.27
I.3	74.74	68.92	63.79	76.20	59.83	58.75
I.6	74.73	74.34	72.44	70.73	72.62	75.67
I.7	16.67	21.05	21.28	32.65	14.29	18.19
J.1	0.00	0.00	0.00	***0.00***	0.00	7.40
J.3	36.36	38.89	44.44	56.41	18.19	13.34
J.5	14.04	23.53	10.67	22.47	22.59	10.42
J.6	24.56	29.27	24.10	36.62	16.67	17.30
K.1	23.53	25.00	30.30	***24.24***	24.00	10.52
K.3	76.25	73.82	71.49	76.45	74.38	77.64
K.4	20.25	8.42	12.16	25.51	17.95	12.46
K.6	18.32	21.13	21.43	21.63	10.76	23.61
K.7	19.16	22.01	22.43	22.43	26.74	20.82
Macro F1	34.44	32.52	31.70	**37.90**	30.17	30.63

Table 5.9: Macro F1 comparison between Majority Voting using the best evidence per class in isolation, Linear Fusion, GA, combined majority GP, content-based and combination-based SVM for the second level on the 30% sample.

Class	Majority Best Evidence	Linear Fusion	GA	Majority GP	Content-based SVM	Combination-based SVM
A.0	53.50	53.52	52.89	55.50	59.33	57.43
A.1	0.00	0.00	0.00	***0.00***	10.52	11.11
B.1	42.55	44.44	40.91	50.85	27.78	38.46
B.3	40.00	46.67	42.42	42.86	38.71	39.02
B.5	24.56	14.04	13.11	30.30	12.12	19.45
B.6	53.07	49.33	45.82	57.59	38.89	46.62
B.7	59.75	53.88	49.46	62.22	58.89	55.35
B.8	0.00	0.00	0.00	***9.52***	0.00	6.90
C.0	0.00	9.52	19.05	***11.76***	0.00	11.11
C.1	23.81	18.60	19.05	21.74	8.89	20.93
C.2	74.57	70.22	68.27	77.11	71.53	71.74
C.4	0.00	0.00	0.00	***0.00***	0.00	6.25
D.1	6.90	12.12	12.12	***15.79***	26.09	17.91
D.2	61.43	54.95	57.65	65.19	57.41	60.72
D.3	66.76	65.53	64.37	69.33	61.05	65.48
D.4	68.75	65.07	63.51	71.65	61.21	64.25
E.x	53.33	38.89	43.75	47.06	10.52	10.26
F.1	44.76	38.46	35.84	40.99	35.82	40.25
F.2	37.74	28.46	29.96	45.99	21.88	31.19
F.3	0.00	11.11	11.11	***17.39***	0.00	0.00
F.4	46.38	46.58	44.74	50.63	39.44	43.18
G.1	74.06	72.33	69.21	75.59	67.96	69.12
G.2	46.32	42.59	43.56	41.58	27.69	33.59
G.3	49.38	34.67	34.57	46.34	33.34	34.48
G.4	34.48	31.82	28.85	37.21	36.36	34.41
H.1	0.00	0.00	0.00	5.41	5.72	4.88
H.2	78.63	71.11	70.85	81.26	76.48	75.81
H.3	63.16	59.83	57.14	67.92	55.96	61.84
H.4	7.14	18.75	22.86	23.53	0.00	8.89
H.5	57.82	48.18	49.85	61.30	51.45	49.59
I.1	46.15	39.29	40.68	49.12	19.51	38.81
I.2	70.51	64.60	64.31	73.06	56.43	62.27
I.3	75.02	69.24	67.51	77.10	64.73	63.76
I.6	76.41	71.47	70.71	79.63	73.96	74.86
I.7	14.29	23.08	14.29	32.26	8.00	23.53
J.1	0.00	0.00	0.00	***0.00***	0.00	0.00
J.3	37.04	40.00	29.63	48.28	18.19	4.17
J.5	9.38	9.52	9.84	19.18	11.77	18.92
J.6	62.50	54.05	45.45	60.00	14.81	36.73
K.1	0.00	11.76	0.00	***12.50***	26.09	11.11
K.3	75.88	74.70	72.87	78.42	75.41	77.68
K.4	31.09	22.42	28.46	33.20	21.82	32.86
K.6	25.83	20.07	21.62	29.83	19.52	37.58
K.7	43.10	39.64	40.65	40.68	51.67	41.25
Macro F1	39.46	37.28	36.29	**43.57**	33.11	36.68

Table 5.10: Micro F1 comparison between Majority Voting using the best evidence per class in isolation, Linear Fusion, GA, combined majority GP, content-based and combination-based SVM for the second level.

	15%	30%
Majority Best Evidence	60.00	64.87
Linear Fusion	56.85	60.45
GA	55.05	59.53
Majority GP	**62.84**	**67.31**
Content-based SVM	56.54	59.43
Combination-based SVM	55.41	60.36

Table 5.11: Gains of majority GP over combination-based SVM.

Performance Measure	First Level		Second Level	
	(15%)	(30%)	(15%)	(30%)
Macro F1	14.50%	10.03%	23.73%	18.78%
Micro F1	8.77%	8.10%	13.41%	11.51%

The performance of content-based SVM is also worse than that of the majority GP, which suggests that we have a better classification method.

From Tables 5.8 and 5.9, we also can notice that for classes A.1, C.4 and J.1, the F1 value given by GP was 0 but either content-based SVM or combination-based SVM achieved a non-zero F1 value. This is because all these three classes are very small (class A.1 has 13 test documents, class C.4 has 17 test documents, and class J.1 has 14 test documents) and content-based or combination-based SVM sometimes was able to classify 1 document correctly.

Finally, when we compare majority GP against both linear fusion and GA, we see that GP is able to discover better similarity functions.

We wanted to compare GP with other approaches to decide if GP produces measurably better results than the others. The most common approach to this problem is to apply the pair-wise t-test to these differences [54]. The t-test compares the magnitude of the difference between the two approaches under comparison to the variation among the differences. If the average difference is large compared to its standard error, then the approaches are significantly different. According to [54], t-test is a valid test for the measures we are trying to compare here. We did a pair-wise t-test comparing GP with all other columns in Tables 5.5, 5.6, 5.8, and 5.9, respectively. Majority GP is statistically significantly different from all the others, with $p<0.05$. So we can conclude

that on average, majority GP outperforms the other approaches. However, we can notice that GP's performance is worse than content-based and combination-based SVM in category A. Category A is a very special category and has some systematic aspects which keep GP from doing well in this category. As mentioned in Section 5.1.1, the incomplete information problem is much worse in this category compared with the whole ACM digital library collection. In addition, the citation-based types of evidence for this category have worse baselines compared with the other categories.

5.1.3 Effects of Fitness Functions

In order to test the effects of different fitness functions, experiments were performed on the first level of CCS using the three fitness functions discussed in Section 3.4.2. Results based on the class recall fitness function were reported in the previous subsection. Here we compare the results using different fitness functions.

Table 5.12: Macro F1 comparison between different fitness functions for the first level.

Fitness Functions	15%	30%
GP + Class Recall	**61.60**	**62.66**
GP + Macro F1	58.79	60.72
GP + FFP4	56.72	59.28

Table 5.13: Micro F1 comparison between different fitness functions for the first level.

Fitness Functions	15%	30%
GP + Class Recall	**73.78**	**76.49**
GP + Macro F1	71.99	75.01
GP + FFP4	71.09	74.14

From Tables 5.12 and 5.13, we can see that the class recall fitness function gives the best result comparing with the other two fitness functions. One possible reason of the class recall fitness function being the best is due to the fact that we are using the GP discovered similarity functions in a *kNN* classifier. Recall from Section 3.4.2, the class recall fitness function uses a similar strategy to predict categories for documents. So when this function is used, GP is trained to maximize the accuracy of predicting categories for documents using the nearest k neighbors. Another possible reason is that the class recall fitness function only uses the nearest k neighbors to calculate the fitness value while the other two fitness functions use a larger set of more documents to calculate the fitness value. Thus, this larger set may introduce more noise and is more difficult for GP to

discover a similarity function with higher fitness value. The fitness function plays an important role in guiding GP to discover a good solution within a large search space. Good fitness functions will help GP to explore the search space more effectively and efficiently. Thus choosing a proper fitness function is very important for the effectiveness and efficiency of evolutionary algorithms. Here the results indicate that the design of fitness functions is instrumental in performance improvement.

5.1.4 Effects of Active Sampling

To test the active sampling strategy, we actively select training data samples from the pool of data generated by the random sampling strategy. Specifically, we used the 15% training data generated by the random sampling strategy as the pool of data and calculated a class profile for a class C. Then, in order to actively generate the training data for class C, we computed the similarity between class C's profile and all out-class documents (not belonging to class C) in the pool data. And then the top N out-class documents are selected as the out-class (negative) samples of the active training data for classification purposes. The $|C|$ documents belonging to class C itself serve as the positive samples of the active training data. We performed different exploration experiments to find a good value of the number of out-class samples, N. For example, N could be $|C|$, $2 \times |C|$, or $3 \times |C|$. We expect the performance would increase as N increases, and then stabilize or maybe drop a little bit when N gets larger. This is because those N documents are likely to be the most difficult ones to classify and thus would be most helpful. If we keep increasing N until N includes all the negative documents which are originally in the full training set generated by the random sampling, we lose the motivation of using those N difficult documents to make GP learn harder. Thus we might see a peak in performance as N increases. In our experimental results shown in Tables 5.14 and 5.15, $N = 2 \times |C|$ gives the best results. We believe this might not be the most appropriate setting since the results might be biased by the data. In order to discover the most appropriate setting and to verify the pattern in performance as N increases, we will need to run the experiments multiple times for each setting of N. This way the bias could be minimized and the true pattern could be discovered.

Tables 5.14 and 5.15 shows the macro-averaged and micro-averaged results, respectively, of using these different settings of N with two fitness functions, $MacroF1$ and $FFP4$. We were not able to test the class recall fitness function due to the fact that the number of positive training samples in the active training set is large so that almost all the nearest k neighbors for a document d would be documents from the same class as d. This results in a maximum fitness value of 1. So there is

no room for GP to learn or to improve the similarity functions during the training process.

Table 5.14: Macro F1 comparison between active sampling and random sampling for the first level 15% training set.

Fitness Functions	Active Sampling			Full Training Set
	$N = \|C\|$	$N = 2\times\|C\|$	$N = 3\times\|C\|$	
GP + Macro F1	56.29	**57.10**	55.78	**58.79**
GP + FFP4	50.88	50.80	51.94	56.72

Table 5.15: Micro F1 comparison between active sampling and random sampling for the first level 15% training set.

Fitness Functions	Active Sampling			Full Training Set
	$N = \|C\|$	$N = 2\times\|C\|$	$N = 3\times\|C\|$	
GP + Macro F1	69.77	**70.59**	69.16	**71.99**
GP + FFP4	66.12	64.54	66.12	71.09

Notice that the active sampling strategy is used to construct the training sets. The validation set and test set are still the same as compared with the random sampling strategy. So the results reported in Tables 5.14 and 5.15 are comparisons based on the same test sets. When comparing with the random sampling strategy, we can see that random sampling still gives better results. We think this is due to the relatively small size of the data pool. Thus we were not able to construct the active samples effectively. From Tables 5.14 and 5.15, we also can notice that when $N = 2 \times |C|$, active sampling achieved relatively close results compared with the full training set when Macro F1 fitness function was used. This suggests active sampling might be a good sampling strategy to achieve good results and to save training time since active samples are much smaller than the full training set generated by the random sampling.

5.2 Experiments on Cadê Web Directory

As with the ACM DL collection, experiments were performed on the Cadê Web directory under both the first and the second level.

5.2.1 Baselines

Table 5.16 shows the type of evidence that performs the best, according to the $F1$ measure, when applied to a *kNN* algorithm for a specific category in isolation in the Cade12 subcollection. Here we do not show a similar table for each category in the Cade188 subcollection since there are 188 categories and it would be too much data to show in a single table. Tables 5.17 and 5.18 show the macro-averaged and micro-averaged F1, respectively, over all categories for both the first level (Cade12) and the second level (Cade188) for each similarity evidence, also in isolation.

Table 5.16: Best baseline for each category of Cadê (first level).

Class	F1/ class(15%)	Best Evidence (15%)	F1/ class(20%)	Best Evidence (20%)
1	**77.18**	Amsler	**78.30**	Amsler
2	**74.53**	Co-Cit	**74.97**	Amsler
3	89.17	Co-Cit	89.52	Co-Cit
4	87.63	Co-Cit	88.12	Co-Cit
5	95.72	Amsler	95.96	Co-Cit
6	91.68	Co-Cit	91.79	Amsler
7	88.11	Co-Cit	89.01	Co-Cit
8	85.15	Co-Cit	86.55	Co-Cit
9	89.36	Amsler	90.86	Co-Cit
10	**64.92**	Co-Cit	**77.07**	Co-Cit
11	72.14	Co-Cit	77.56	Co-Cit
12	**77.86**	Co-Cit	**79.07**	Co-Cit

Table 5.17: Macro F1 on individual evidence on the Cadê Web directory.

Evidence	First Level		Second Level	
	(15%)	(20%)	(15%)	(20%)
Content_BOW	23.37	24.18	9.54	9.12
Content_Cos	33.94	36.49	22.58	23.46
Content_Okapi	35.60	38.70	23.02	23.73
Bib_Coup	4.86	5.05	0.40	0.42
Amsler	**82.52**	**84.64**	**84.07**	**85.58**
Co-Cit	**82.77**	**84.87**	**84.33**	**85.66**
Comp_Aut	66.66	72.25	75.85	79.41
Comp_Hub	10.43	11.32	4.15	4.08

From Table 5.16, we can see that citation based evidence, Co-Cit and Amsler, are the most important evidence for all the Cade12 classes. From Tables 5.17 and 5.18, we also can see that the best

Table 5.18: Micro F1 on individual evidence on the Cadê Web directory.

Evidence	First Level		Second Level	
	(15%)	(20%)	(15%)	(20%)
Content_BOW	32.15	34.71	24.74	25.05
Content_Cos	42.51	45.18	32.61	35.76
Content_Okapi	44.79	47.28	36.87	39.45
Bib_Coup	22.73	22.70	14.88	14.19
Amsler	**83.28**	**84.30**	**82.62**	**83.63**
Co-Cit	**83.39**	**84.33**	**82.89**	**83.81**
Comp_Aut	69.51	74.53	71.07	76.21
Comp_Hub	24.45	25.68	16.20	15.31

two types of evidence are Co-Cit and Amsler. We also can notice that the content-based types of evidence are much worse compared with the good citation-based evidence.

5.2.2 Results

In this subsection, we report the experimental results based on the stratified random sampling strategy and the class recall fitness function. We also performed experiments to test the effect of other fitness functions and active sampling on this Cadê Web directory. The results of these experiments are reported in the next two subsections.

In a comparison, class by class, for the first level, between the majority GP (Table 5.19 column 5 and Table 5.20 column 5) and the best evidence in isolation (Table 5.16), the majority GP outperforms the best evidence in half of the classes in the 15% sample. Majority GP didn't improve upon class 3, 5, 6, 7, 8 or 9 in the 15% sample. For the 20% sample, majority GP gives better performance except for classes 6, 7, 8, and 9. From Table 5.16, we can see that the best baselines for these few classes are already pretty good, so this does not give GP much room to improve. But for classes with relatively worse evidence, e.g., classes 1, 2, 10, and 12, majority GP was able to show more improvement. This verifies that our approach is able to reinforce document relationship information through effectively combining multiple sources of evidence, and so can produce better results. Notice that we do not show the results class by class for the second level since we have 188 categories and that's too much to fit in a table. Instead, we only show the macro-averaged and micro-averaged F1 value over all the 188 categories.

When comparing the majority GP against the majority using the best evidence on the first level

Table 5.19: Macro F1 comparison between Majority Voting using the best evidence per class in isolation, Linear Fusion, GA, combined majority GP, content-based SVM, and Bayesian Network for Cadê first level on the 15% sample.

Class	Majority Best Evidence	Linear Fusion	GA	Majority GP	Content-based SVM	Bayesian Network
1	77.18	78.77	87.76	**84.48**	54.67	76.96
2	74.52	71.92	81.83	**79.43**	43.27	74.55
3	89.14	74.52	87.81	85.61	50.85	89.20
4	87.64	81.68	88.71	88.81	50.72	87.62
5	95.70	86.66	95.50	94.93	42.52	95.76
6	91.59	79.24	91.41	90.19	47.66	91.66
7	87.09	71.99	85.22	85.65	38.90	88.11
8	85.04	69.25	85.89	82.67	19.49	85.21
9	89.36	68.52	88.82	85.23	20.60	89.33
10	64.76	37.38	60.98	**68.14**	20.40	65.00
11	72.53	51.32	67.33	72.46	16.09	71.63
12	78.04	72.68	78.59	**82.74**	22.40	77.85
Macro F1	82.72	70.33	83.32	**83.36**	35.63	82.74

(Tables 5.19 and 5.20 between columns 2 and 5, respectively) we find that majority GP presents slightly better performance. This also is the case for the second level as we can see from Table 5.22. If we look at the results given by Micro F1 in Table 5.21, between rows 2 and 5, GP also shows slightly better performance over the majority using the best evidence. GP did not yield a lot of improvement because the content-based types of evidence are inaccurate, and at the same time, the few strong citation-based evidence cases already have very good performance. From Table 5.23, we can notice that for the second level, the micro-averaged $F1$ value of GP is worse than that of most other baselines. This is because the overfitting of GP results in 4 out of the 188 second level classes being over confident; thus there are many false positive documents classified to these 4 classes. This dramatically influences the micro-averaged result. However, even though macro-averaged value tends to be influenced more by rare categories, since that's only 4 out of 188, the macro average of the 188 F1 values is not affected that much compared with the micro-averaged value. This is further illustrated in the following example. Suppose the other 184 classes' $F1$ values are around value 88 (some are worse, some are better; suppose this is the average value). If the 4 over confident classes' $F1$ values are around value 5, the macro-averaged $F1$ value is still $(184 \times 88 + 4 \times 5)/188$, which is around value 86. But if among these 4 classes, one class has 2781 predicted documents, among which only 81 predictions are correct, then given 16,932 test

Table 5.20: Macro F1 comparison between Majority Voting using the best evidence per class in isolation, Linear Fusion, GA, combined majority GP, content-based SVM, and Bayesian Network for Cadê first level on the 20% sample.

Class	Majority Best Evidence	Linear Fusion	GA	Majority GP	Content-based SVM	Bayesian Network
1	78.30	78.07	88.37	**86.39**	57.22	78.02
2	75.12	72.51	82.21	**81.20**	45.36	74.95
3	89.48	80.12	90.41	90.18	54.18	89.56
4	87.85	81.77	88.26	88.76	52.32	88.11
5	95.92	85.47	96.27	96.24	47.08	95.94
6	91.70	81.06	91.46	90.97	51.29	91.76
7	88.54	75.29	87.82	88.26	43.57	89.07
8	86.42	71.53	86.71	85.29	23.80	86.55
9	90.73	72.59	90.57	89.03	26.27	90.85
10	77.01	51.76	77.73	**81.97**	25.37	76.97
11	77.23	67.89	76.83	79.25	21.02	77.80
12	79.07	78.39	78.52	**86.01**	20.19	78.70
Macro F1	84.78	74.70	86.26	**86.96**	38.97	84.86

documents, the micro-F1 value will drop $(2781 - 81)/16932$, which is about 16. Future work is needed to help alleviate the overfitting problem.

When we compare majority GP against linear fusion, we see that GP is able to discover better similarity functions for both sample sets of both levels. Again, due to the very good citation-based evidence, GA also is able to find a good linear similarity function.

If we compare content-based SVM with majority GP, the performance of content-based SVM is much worse than that of the majority GP. This should be expected because the content-based types of evidence are much worse than the link-based evidence.

We were not able to compare GP with combination-based SVM for this collection. The size of the combined kernel matrix is huge (about 500 million points for a symmetric sparse representation); thus we were not able to compute the multi kernel method for Cadê.

Calado [11] proposed a Bayesian network model to improve classification through combining content-based and link-based information in the Cadê collection, and reported good results. Bayesian networks provide a graphical formalism for explicitly representing relationships among variables of a probability distribution, and can derive any probability regarding such variables. They have

Table 5.21: Micro F1 comparison between Majority Voting using the best evidence per class in isolation, Linear Fusion, GA, combined majority GP, content-based SVM, and Bayesian Network for the first level.

	15%	20%
Majority Best Evidence	83.38	84.34
Linear Fusion	75.61	77.39
GA	86.92	88.12
Majority GP	85.22	87.51
Content-based SVM	45.56	48.45
Bayesian Network	83.40	84.33

Table 5.22: Macro F1 comparison between Majority Voting using the best evidence per class in isolation, Linear Fusion, GA, combined majority GP, content-based SVM, and Bayesian Network for Cadê second level.

	15%	20%
Majority Best Evidence	82.51	83.99
Linear Fusion	75.25	76.72
GA	82.75	84.20
Majority GP	85.02	85.96
Content-based SVM	17.93	18.49
Bayesian Network	84.86	86.04

been successfully applied to several information retrieval tasks [92, 110]. Ribeiro-Neto et al. [93] have shown that Bayesian networks are especially useful when combining different sources of evidence. So Bayesian networks offer a strong baseline with which to compare the GP results.

Calado et al. [11] adopted the Bayesian framework proposed by Ribeiro-Neto and Muntz in [92] for modeling distinct Web IR problems and for combining content-based and link-based information. The model they proposed is shown in Figure 5.1. In the network of this figure, nodes D_1 through D_N are the root nodes and represent prior knowledge about the problem, i.e., a set of classified documents (the training set). Node C represents a category. A category is composed of a set of classified documents. Thus, the edges from nodes D_i to node C represent the fact that observing a set of classified documents will influence the observation of a category. Nodes T_1 through T_K and L_1 through L_K represent the documents to be classified (the test set) under the content-based and link-based contexts, respectively. More specifically, each node T_i represents evidence from the content-based classifier indicating that test document i belongs to category C, and each node L_i represents evidence from the link-based classifier indicating that document i belongs to category C. The edges from a node D_j to a node T_i represent the fact that observing a set of training

Table 5.23: Micro F1 comparison between Majority Voting using the best evidence per class in isolation, Linear Fusion, GA, combined majority GP, content-based SVM, and Bayesian Network for the second level.

	15%	20%
Majority Best Evidence	83.49	83.81
Linear Fusion	74.80	76.02
GA	82.45	83.99
Majority GP	73.46	77.86
Content-based SVM	35.43	37.55
Bayesian Network	82.93	83.84

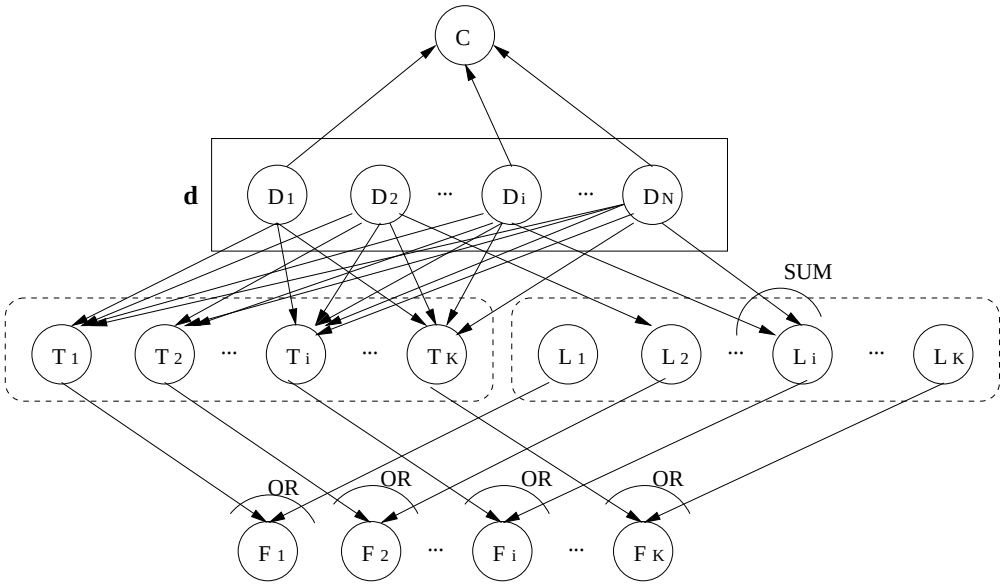


Figure 5.1: Bayesian network model to combine evidence from the content-based classifier with evidence from the link structure.

documents will influence observing test document i belonging to category C. Similarly, an edge from a node D_j to a node L_i means the training document j is related to test document i. Thus, the observation of test document i as belonging to category C is influenced by the observation of the training documents that are related to document i. Finally, nodes F_1 through F_K represent the final evidence that each test document belongs to category C. Both the content-based and link-based evidence will influence this final evidence, as shown by the incoming edges to the final evidence.

Given these definitions, the probability that a test document i belongs to category C can be determined through:

$$P(f_i|c) = \eta \sum_{\mathbf{d}} \Big(1 - \big(1 - P(t_i|\mathbf{d})\big)\big(1 - P(l_i|\mathbf{d})\big)\Big) P(c|\mathbf{d})\ P(\mathbf{d}) \tag{5.2}$$

where $\eta = 1/P(c)$ is a normalizing constant and d is a possible state of all the variables D_j where only the variables corresponding to the training documents of class C are active. $P(t_i|\mathbf{d})$ is defined as the value given by the content-based classifier of document i belonging to category C, and $P(l_i|\mathbf{d})$ is defined as the value given by the link-based classifier of document i belonging to category C. $P(c|\mathbf{d})$ is the probability that is used to select only the training documents belonging to category C. $P(\mathbf{d})$ is a constant *a priori* probability for every d. More details of this equation can be found in [11].

If use of the Bayesian network is compared with majority GP, from Tables 5.19 and 5.20 columns 5 and 7, respectively, we can see that majority GP shows slightly better performance for both the 15% and 20% sample for the first level using Macro F1 as the comparison criteria. From Table 5.21, we also can see that GP presents better Micro F1 value for both samples for the first level. For the second level, as shown in Table 5.22, GP shows comparable results with the Bayesian network based on the macro-averaged F1 value. Again, due to the overfitting problem, GP's Micro F1 performance on the second level (Table 5.23) is worse.

From all these results we can see that our GP classification framework is applicable to different collections and shows better classification results. In contrast, different Bayesian network models need to be derived for different collections. But like other evolutionary computation approaches, one disadvantage of GP is the scalability problem: increasing problem size leads to an unmanageably high increase in space and time resource requirements. The sampling strategy explored in this research partially addressed this problem. But still the experiments are time consuming as shown in Section 5.3. The Bayesian network model has the advantage of being very efficient compared

with GP. As mentioned earlier, it seems that GP was able to more effectively combine different sources of evidence to produce better classification results, especially for classes (i.e., classes 1, 2, 10, and 12 in Tables 5.19 and 5.20) with relatively worse baselines.

5.2.3 Effects of Fitness Functions

In order to test the effects of different fitness functions, experiments were performed on the first level of Cadê using the three fitness functions discussed in Section 3.4.2. We reported the results based on the class recall fitness function in the previous subsection. Here we compare the results using different fitness functions.

Table 5.24: Macro F1 comparison between different fitness functions for the first level.

Fitness Functions	15%	20%
GP + Class Recall	**83.36**	**86.96**
GP + Macro F1	81.87	84.04
GP + FFP4	79.13	83.46

Table 5.25: Micro F1 comparison between different fitness functions for the first level.

Fitness Functions	15%	20%
GP + Class Recall	85.22	**87.51**
GP + Macro F1	**86.16**	83.31
GP + FFP4	84.40	85.69

From Table 5.24, we can see that, like what we observed for the ACM Digital Library collection, the class recall fitness function gives the best macro-averaged result when compared with the other two fitness functions. If we look at the micro-averaged result shown in Table 5.25, we notice that for the 15% sample, the fitness function Macro F1 gives slightly better results than the class recall fitness function. For the 20% sample, the class recall fitness function still show the best micro-averaged result.

5.2.4 Effects of Active Sampling

From what we learned from the exploratory experiments on the ACM Digital Library collection, we used $N = 2 \times |C|$ in the active sampling experiments for the Cadê Web collection. Here C is a class, $|C|$ is the size of this class, and N is the number of out-class samples for class C. Tables 5.26

and 5.27 show the results of comparing active sampling with the full training set generated by the stratified random sample.

Table 5.26: Macro F1 comparison between active sampling and random sampling for the first level 15% training set.

Fitness Functions	Active Sampling	Full Training Set
GP + Macro F1	**82.48**	81.87
GP + FFP4	*78.92*	*79.13*

Table 5.27: Micro F1 comparison between active sampling and random sampling for the first level 15% training set.

Fitness Functions	Active Sampling	Full Training Set
GP + Macro F1	83.05	**86.16**
GP + FFP4	*84.36*	*84.40*

When compared with the random sampling strategy, we can see that active sampling achieved very comparable results with random sampling if FFP4 is used as the fitness function for both the macro-averaged and micro-averaged comparisons. But if Macro F1 is used as the fitness function, active sampling was able to give slightly better macro-averaged results. Recall from Section 5.1.4 that for the ACM Digital Library collection, random sampling gives better results compared with active sampling. We argue this is due to the relatively small size of the data pool. For the Cadê Web collection, from Table 4.1, we can see that we have a larger size of the data pool to actively construct the active samples, so we get better results, but at the same time save training time since the active samples are much smaller when compared with the full training set.

5.3 Computation Time and Space Analysis

As discussed earlier in Section 3.4.4, the classification framework suffers from scalability. Table 5.28 shows the space requirement and computation time for discovering the similarity functions for the ACM CCS first level 15% sample set, which has 4,963 samples. Figure 5.2 shows a graph of class size vs. training time. Given a large collection, to store the similarity values between each pair of documents, the space requirement is enormous. Approaches like sparse representation of the similarity matrices are feasible to save disk space. But nowadays, the hard drive price drops quickly and most servers have large disk space so space is not a problem for the framework. We

consider the computation time shown in Table 5.28 to be acceptable since we focus more on effectiveness in this research. We partially addressed the scalability issue through optimization, i.e., sampling strategies, as discussed in Section 4.2. Parallel processing is another approach to speed up the learning process. More of this is pointed out in the future work section of Chapter 7.

Table 5.28: Computation Time and Space.

Machine	Training Set Size	Class	Class Size	Training Time (Hours)	Training Matrices Size (M Bytes)
Intel Pentium 4 3.4GHz Linux Server	4,963	A	355	1.53	105
		B	345	1.60	102
		C	357	2.42	106
		D	1152	72.94	343
		E	47	0.47	14
		F	219	1.33	65
		G	263	2.02	78
		H	770	33.15	229
		I	621	2.96	185
		J	261	1.04	77
		K	573	2.39	170
		Total	4,963	**121.85**	**1,474**

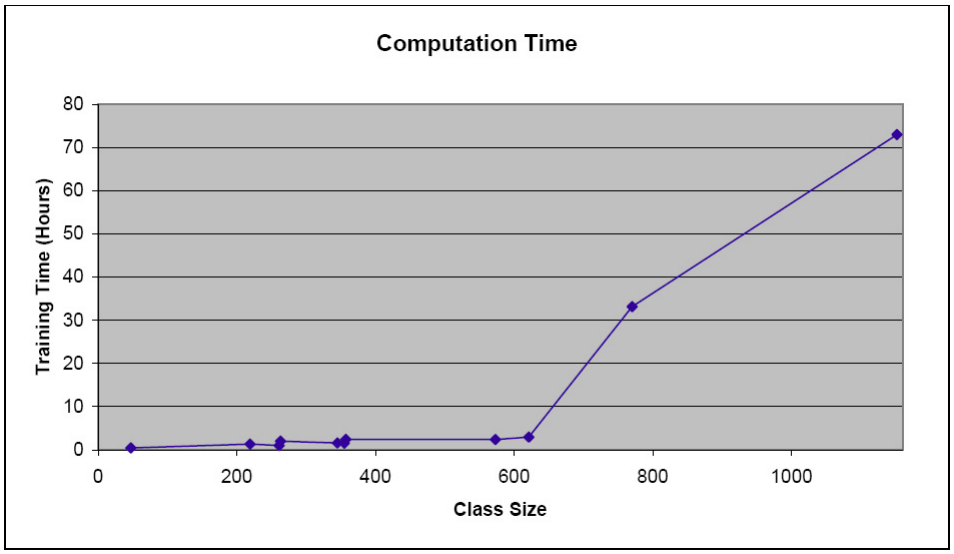


Figure 5.2: Class size vs. training time.

Chapter 6

Analysis

Our experimental results show that good text classification is possible when documents are noisy or have limited amounts of text (e.g., short metadata records) by fusing or combining multiple sources of evidence through genetic programming methods. In this chapter, we try to understand why the solutions, unveiled by GP, work so well, so we can understand the key factors that contribute to accurate text classification.

One of the advantages of the GP discovered solutions over other modeling techniques such as Neural Networks [30] or SVMs [113] is that the solutions are interpretable. In other words, one can look at the results and understand why some of the similarity functions work while others do not. Sometimes, though, the results are of a very high complexity that is beyond any meaningful interpretation by domain experts (the bloat problem in GP [6] as illustrated in Figure 6.1; this sample tree contains more than 100 nodes and, other than noting that some frequently appeared subtrees occur (like full_Cosine, as indicated in bold face), it's hard to understand why a formula like this works). Accordingly, we perform an extensive and comprehensive analysis of the reasons why GP-based fusion works and outperforms other similar techniques.

((+ (* (+ (/ acop abs_Okapi) (+ (+ **full_Cosine** amsler) (/ amsler 1))) (/ (sqrt title_Okapi) (+ (+ (+ **full_Cosine** amsler) (/ amsler (* (+ (sqrt (/ **full_Cosine** abs_Okapi)) (+ (+ 2.8408 **full_Cosine**) (+ **full_Cosine** amsler))) (* (+ (/ acop abs_Okapi) (+ **full_Cosine** comp_hub)) (/ (sqrt title_Okapi) (+ full_Okapi 2.1113)))))) (/ acop amsler)))) (+ **full_Cosine** (+ **full_Cosine** amsler))) (* (+ (/ acop (sqrt 0_5)) (+ **full_Cosine** comp_hub)) **full_Cosine**))*

Figure 6.1: Sample Discovered Tree.

6.1 Statistical Analysis of the Tree Population

We obtained a population of good trees as well as bad trees for each class during the experiments. Those trees correspond to the similarity functions that are used in *kNN* classifiers in our classification framework. We would like to know how different the trees are. A tree population can be thought of as a data space, where each tree of the population corresponds to a data point in the data space. The difference between trees are measures of how far the trees are from each other. Thus, we need a distance measure to measure distance between trees. The distance measure we adopted is called edit distance, which is explained in detail in Section 6.1.1.

Following the analysis of the distance between trees, we analyzed the structure of variability among good performing trees as well as the variability between good and bad performers.

To better understand the GP tree population, we devised a technique to derive an average "high-performer" tree for the tree population.

6.1.1 Edit Distance between Trees

The trees obtained through GP are ordered, labeled, and rooted. A rooted tree is a tree in which a particular vertex is designated as a root. An ordered tree is a rooted tree in which the order of the subtrees is significant. A labeled tree is a tree with its nodes labeled.

Before discussing the edit distance algorithm for trees, let's look at the string edit distance algorithm first. The *edit distance* of two strings, s_1 and s_2, is defined as the *minimum* number of *point mutations* required to change s_1 to s_2, where a point mutation is one of insert, delete, and replace operators. The following recurrence relations define the edit distance, $d(s_1, s_2)$, of two strings s_1 and s_2:

$d(''$,$'')= 0$, where $''$ means an empty string

$d(s$,$'')= d(''$, $s) = |s|$, where $|s|$ is the length of string s

$d(s_1 + ch_1, s_2 + ch_2) = min(d(s_1, s_2)+$ if $ch_1 = ch_2$ then 0 else 1 fi,

$$d(s_1 + ch_1, s2) + 1,$$

$$d(s_1, s_2 + ch_2) + 1)$$

The first two rules above are obvious, so let's consider the last rule. Here, each string has a last

character, ch_1 and ch_2, respectively. And we need to *edit* $s_1 + ch_1$ into $s_2 + ch_2$. If ch_1 equals ch_2, then ch_1 and ch_2 can be matched for no penalty, i.e., 0, and the overall edit distance is $d(s_1, s_2)$. If ch_1 differs from ch_2, then ch_1 could be *mutated* into ch_2, i.e., 1, giving an overall edit distance of $d(s_1, s_2) + 1$. Another option to *edit* $s_1 + ch_1$ into $s_2 + ch_2$ is to edit $s_1 + ch_1$ into s_2 and then insert ch_2, giving an overall edit distance of $d(s_1 + ch_1, s_2) + 1$. The last option is to delete ch_1 and edit s_1 into $s_2 + ch_2$, giving the edit distance of $d(s_1, s_2 + ch_2) + 1$. There are no other alternatives. In order to get the edit distance of the two strings, we take the least expensive, i.e., min, of these three alternatives.

The following algorithm details how to use a dynamic programming technique to calculate the edit distance between two strings. Here, a two-dimensional matrix $m[0..|s_1|, 0..|s_2|]$ is used to hold the edit distance values.

Algorithm 4 String Edit Distance Algorithm

Let $m[0, 0] = 0$
for $i = 1..|s_1|$ **do**
  $m[i, 0] = i$
end for
for $j = 1..|s_2|$ **do**
  $m[0, j] = j$
end for
for $i = 1..|s_1|$ **do**
  for $j = 1..|s_2|$ **do**
    $m[i, j] = min(m[i-1, j-1] +$ if $ch_1 = ch_2$ then 0 else 1 fi,
                $m[i-1, j] + 1$,
                $m[i, j-1] + 1)$
  end for
end for

The tree distance problem is harder than the above mentioned string edit distance problem. For two strings s_1 and s_2, if $s_1[i] = s_2[j]$, then the distance between $s_1[1..i]$ and $s_2[1..j]$ is the same as the distance between $s_1[1..i-1]$ and $s_2[1..j-1]$. However, to get the distance between trees, the ancestral relationship has to be taken into account and an analogous simplification is not possible. As discussed in Section 2.3.1, Zhang and Shasha [125] proposed a dynamic programming algorithm similar to the string edit distance algorithm. In fact, the string edit distance algorithm can be considered as a special case of this tree edit distance algorithm. This algorithm works well for our GP trees, which are ordered labeled trees.

The distance between two trees is defined as the number of edit operations required to change one

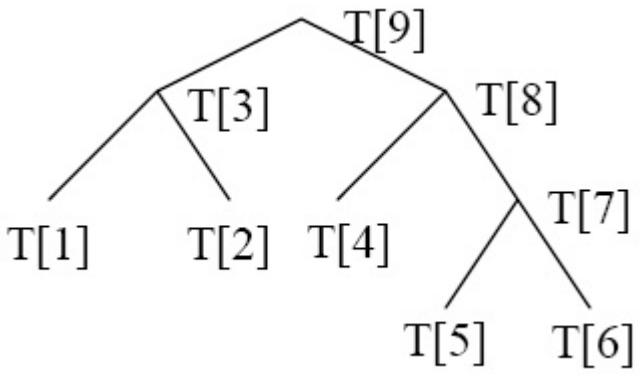


Figure 6.2: Tree Postorder Numbering.

tree to another. The edit operations can be insert, delete, and change. The insert operation inserts a new node into a tree. The delete operation deletes a node in a tree. The change operation changes the label of a node in a tree.

To identify the nodes in a tree, we use a left-to-right postorder numbering of the nodes in the trees. Figure 6.2 gives an example of the postorder numbering. $T[i]$ is the ith node in the tree according to this postorder numbering.

Given the above edit operations definition, let's look at what a *mapping* is. A mapping between two trees, T_1 and T_2, is a graphical specification of what edit operations apply to each node in the two trees. It shows a way to transform tree T_1 to tree T_2. Figure 6.3 shows an example of a mapping. A dotted line from $T_1[i]$ to $T_2[j]$ indicates that the *change* edit operation needs to be applied in order to change $T_1[i]$ to $T_2[j]$ if $T_1[i] \neq T_2[j]$, or that $T_1[i]$ remains unchanged if $T_1[i] = T_2[j]$. The nodes of T_1 not touched by a dotted line need to be *deleted* and the nodes of T_2 not touched by a dotted line need to be *inserted*. To construct the sequence of editing operations, we simply need to delete all the nodes indicated by the mapping (i.e., all the nodes in T_1 not touched by the dotted line), then do all the re-labelings, then all the inserts. In this example, the sequence of edit operations needed to transform T_1 to T_2 is: delete node with label $\surd$, which is the node in T_1 not touched by a dotted line, and insert node with label $\surd$, which is the node in T_2 not touched by a dotted line.

A mapping M from T_1 to T_2 has a cost associated with it. Let I and J be the sets of nodes in T_1 and T_2, respectively, not touched by any dotted line in M, then the cost of M can be defined as:

$Cost(M) = \sum_{(i,j)\in M} change(T_1[i], T_2[j]) + \sum_{i\in I} delete(T_1[i]) + \sum_{j\in J} insert(T_2[j])$

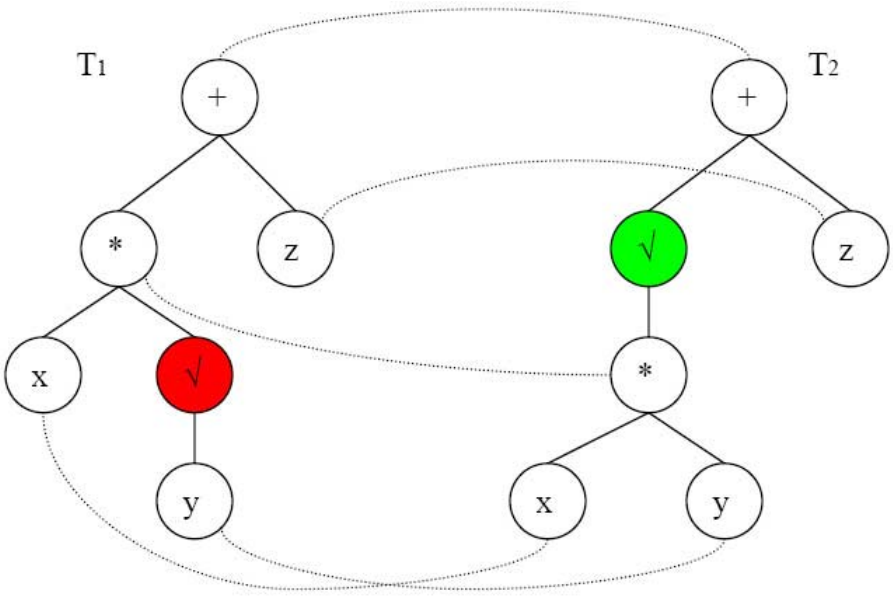


Figure 6.3: A mapping between trees T_1 and T_2.

where $change$ is the edit operation that changes the label of $T_1[i]$ to $T_2[j]$ if $T_1[i] \neq T_2[j]$, $delete$ is the edit operation that deletes node $T_1[i]$ in I, and $insert$ is the edit operation that inserts node $T_2[j]$ in J.

A few more notations are necessary before we can actually start the tree edit distance algorithm. As pointed out earlier, we use a postorder numbering of the nodes in the trees. $l(i)$ is defined as the number of the leftmost leaf descendant of the subtree rooted at $T[i]$, where $T[i]$ is the ith node in the tree according to the left-to-right postorder numbering. In the postordering, $T_1[1..i]$ and $T_2[1..j]$ will generally be forests as shown in Figure 6.4. $T[i..j]$ is defined as the ordered subforest of T induced by the nodes numbered i to j inclusively. If $i > j$, $T[i..j] = \emptyset$. $T[1..i]$ is referred to as $forest(i)$. $T[l(i)..i]$ is referred to as $tree(i)$. The definition of mapping for ordered forests is the same as for trees.

The distance between two forests $T_1[i'..i]$ and $T_2[j'..j]$ is denoted as $fdist(i'..i, j'..j)$. And the distance between the subtree rooted at $T_1[i]$ and the subtree rooted at $T_2[j]$ is denoted as $tdist(i, j)$. Obviously, the value of $tdist$ between the tree rooted at T_1's root and the tree rooted at T_2's root gives us the solution to our problem. Another notation used in the algorithm is called $LR\_keyroots$. The set $LR\_keyroots$ of tree T is defined as follows:

$LR_keyroots(T) = \{k|$there exists no $k' > k$ such that $l(k) = l(k')\}$

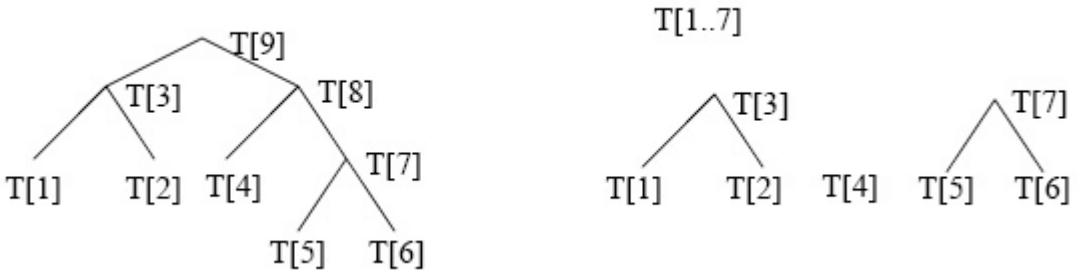


Figure 6.4: Subforests of tree T.

That is, $LR\_keyroots$ is the set consisting of the root and all nodes with left siblings. For example, for the two trees T_1 and T_2 in Figure 6.3, $LR\_keyroots(T_1) = \{3, 5, 6\}$ and $LR\_keyroots(T_2) = \{2, 5, 6\}$.

In the following algorithm that calculates the edit distance between trees, the cost matrix c is a unit cost matrix. That is, the insert, delete, and change operations have unit cost. $LR\_keyroots$ is a boolean array where a value of true indicates that a particular element is in that set. In the algorithm, the main routine EditDistanceBetweenTrees does all the preprocessing and initializations and makes the necessary calls to the routine *treedist*. Results are stored in the permanent array $tdist$ which in the end contains the distances between all possible pairs of subtrees and at position $(|T_1|, |T_2|)$ the distance between the two roots, i.e., the distance between the two trees. The routine $treedist$ computes the distances between the two forests $l(pos_1)..pos_1$ and $l(pos_2)..pos_2$ by using previously computed tree distance values and costs for edit operations. It stores the results in the temporary array $fdist$ and also fills in values for distances between subtrees into the array $tdist$.

Using the above edit distance algorithm, we analyzed GP trees in two ways: intra-class and inter-class. In intra-class analysis, we calculate the distance between: 1) top 20 performing (based on fitness value) trees from a single GP run for a class; and 2) the top performing tree of each generation of a single GP run (generation 0 to 30).

Table 6.1 shows the pair-wise distance between the top 20 trees of class D of ACM CCS. From this table, we can see that most of the trees (except trees 1, 16 and 20) are very similar, i.e., the distance between them is small. Trees 1, 16 and 20 differ from others because: tree 1 is much smaller than others, tree 16 is similar to a few other trees except it has a new mutated branch, and tree 20 has a few branches which also appear in other trees but tree 20 is generated through a few crossover and mutation operations. Table 6.2 shows the pair-wise distance between the top performing tree from

Algorithm 5 Edit Distance between Trees Algorithm

EditDistanceBetweenTrees(T_1, T_2)
{Input: Tree T_1 and T_2}
{Output: Tree Distance between T_1 and T_2 – $tdist[i][j]$}
preprocessing(); {compute l_1 and l_2, $LR\_keyroots1$ and $LR\_keyroots2$}
{init $tdist$ array and costs array c}
for $i = 1$ to $|T_1|$ **do**
  if $LR\_keyroots1[i]$ **then**
    for $j = 1$ to $|T_2|$ **do**
      if $LR\_keyroots1[j]$ **then**
        treedist(i, j, tdist, l_1, l_2, c);
      end if
    end for
  end if
end for
return ($tdist[i][j]$);

treedist($pos_1, pos_2, tdist, l_1, l_2, c$) //return the distance between trees rooted at pos_1 and pos_2
//init $fdist$ array
$fdist[0][0] = 0$; //corresponding to distance between two empty forests, no edit operations needed in this case
for $i = l_1[pos_1]$ to pos_1 **do**
  $fdist[i][0] = fdist[i-1][0] + c[T_1[i]][0]$; //deletions
end for
for $j = l_2[pos_2]$ to pos_2 **do**
  $fdist[0][j] = fdist[0][j-1] + c[0][T_2[j]]$; //insertions
end for
for $i = l_1[pos_1]$ to pos_1 **do**
  for $j = l_2[pos_2]$ to pos_2 **do**
    if $l_1[i] == l_1[pos_1]$ and $l_2[j] == l_2[pos_2]$ **then**
      $fdist[i][j] = min(fdist[i-1][j] + c[T_1[i]][0],$
      $fdist[i][j-1] + c[0][T_2[j]],$
      $fdist[i-1][j-1] + c[T_1[i]][T_2[j]]);$
      $tdist[i][j] = fdist[i][j]$; //put in permanent array
    else
      $m = l_1[i] - l_1[pos_1]; n = l_2[j] - l_2[pos_2];$
      $fdist[i][j] = min(fdist[i-1][j] + c[T_1[i]][0],$
      $fdist[i][j-1] + c[0][T_2[j]],$
      $fdist[m][n] + tdist[i][j]);$
    end if
  end for
end for

each generation of class D's run. The trees generated by GP are expected to evolve generation by generation and stabilize at some point. Our analysis results are consistent with the expectation. From Table 6.2, we can see that the trees evolve generation by generation but from generation 17, the top tree for each generation stabilizes, and remains the same during the rest of the evolution process.

Table 6.1: Distances between top 20 performing trees of class D.

Tree No.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	**0**																			
2	**72**	0																		
3	**69**	9	0																	
4	**69**	9	0	0																
5	**69**	9	0	0	0															
6	**69**	9	0	0	0	0														
7	**69**	9	0	0	0	0	0													
8	**69**	9	0	0	0	0	0	0												
9	**69**	9	0	0	0	0	0	0	0											
10	**70**	11	2	2	2	2	2	2	2	0										
11	**68**	15	6	6	6	6	6	6	6	8	0									
12	**69**	9	0	0	0	0	0	0	0	2	6	0								
13	**69**	9	0	0	0	0	0	0	0	2	6	0	0							
14	**69**	13	4	4	4	4	4	4	4	6	10	4	4	0						
15	**67**	14	5	5	5	5	5	5	5	7	11	5	5	9	0					
16	**69**	**22**	**13**	**13**	**13**	**13**	**13**	**13**	**13**	**15**	**15**	**13**	**13**	**17**	**18**	**0**				
17	**69**	9	0	0	0	0	0	0	0	2	6	0	0	4	5	13	0			
18	**69**	9	0	0	0	0	0	0	0	2	6	0	0	4	5	13	0	0		
19	**69**	9	4	4	4	4	4	4	4	6	10	4	4	8	9	17	4	4	0	
20	**57**	**80**	**74**	**74**	**74**	**74**	**74**	**74**	**74**	**76**	**68**	**74**	**74**	**70**	**73**	**73**	**74**	**74**	**78**	**0**

In the inter-class analysis, pair-wise distances between the best performing tree for each class were calculated. In order to explore and optimize the characteristics of each particular class in different ways, we run the GP experiments locally for each class to discover a similarity function for each class. So we expect that different similarity functions were discovered for different classes. Correspondingly, the distance between the best performing tree (i.e., the discovered similarity function) of each class is very large. The results shown in Table 6.3 are as per expectation.

Table 6.2: Distances between top performing trees (one for each generation) of class D.

Gen No.	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
0	0																														
1	0	0																													
2	0	0	0																												
3	9	9	9	0																											
4	9	9	9	0	0																										
5	9	9	9	0	0	0																									
6	9	9	9	0	0	0	0																								
7	20	20	20	19	19	19	19	0																							
8	20	20	20	19	19	19	19	0	0																						
9	20	20	20	19	19	19	19	0	0	0																					
10	20	20	20	19	19	19	19	0	0	0	0																				
11	20	20	20	19	19	19	19	0	0	0	0	0																			
12	20	20	20	19	19	19	19	0	0	0	0	0	0																		
13	20	20	20	19	19	19	19	0	0	0	0	0	0	0																	
14	20	20	20	19	19	19	19	0	0	0	0	0	0	0	0																
15	20	20	20	19	19	19	19	0	0	0	0	0	0	0	0	0															
16	20	20	20	19	19	19	19	0	0	0	0	0	0	0	0	0	0														
17	41	41	41	38	38	38	38	28	28	28	28	28	28	28	28	28	28	0													
18	41	41	41	38	38	38	38	28	28	28	28	28	28	28	28	28	28	0	0												
19	41	41	41	38	38	38	38	28	28	28	28	28	28	28	28	28	28	0	0	0											
20	41	41	41	38	38	38	38	28	28	28	28	28	28	28	28	28	28	0	0	0	0										
21	41	41	41	38	38	38	38	28	28	28	28	28	28	28	28	28	28	0	0	0	0	0									
22	41	41	41	38	38	38	38	28	28	28	28	28	28	28	28	28	28	0	0	0	0	0	0								
23	41	41	41	38	38	38	38	28	28	28	28	28	28	28	28	28	28	0	0	0	0	0	0	0							
24	41	41	41	38	38	38	38	28	28	28	28	28	28	28	28	28	28	0	0	0	0	0	0	0	0						
25	41	41	41	38	38	38	38	28	28	28	28	28	28	28	28	28	28	0	0	0	0	0	0	0	0	0					
26	41	41	41	38	38	38	38	28	28	28	28	28	28	28	28	28	28	0	0	0	0	0	0	0	0	0	0				
27	41	41	41	38	38	38	38	28	28	28	28	28	28	28	28	28	28	0	0	0	0	0	0	0	0	0	0	0			
28	41	41	41	38	38	38	38	28	28	28	28	28	28	28	28	28	28	0	0	0	0	0	0	0	0	0	0	0	0		
29	41	41	41	38	38	38	38	28	28	28	28	28	28	28	28	28	28	0	0	0	0	0	0	0	0	0	0	0	0	0	
30	41	41	41	38	38	38	38	28	28	28	28	28	28	28	28	28	28	0	0	0	0	0	0	0	0	0	0	0	0	0	0

6.1.2 Multi-sample Analysis of Tree Populations

The results above, in Table 6.3, show that the trees representing each class differ largely from each other. The discovered similarity functions are used in *kNN* classifiers. Thus we would like to statistically prove that different classifiers are working differently so the classifiers explore the characteristics of each particular class. In other words, we want to study the variance between the tree population of different classes. To do this, we carry out a multi-sample analysis of the tree populations, which parallels the intuition of conventional analysis of variance. The aim is to assess whether distinct trees are produced by GP for distinct classifiers.

As discussed in Section 2.3.1, Feigin and Alvo [39], building on the ideas in Rao [90], proposed a basic analysis framework for apportionment of diversity. This analysis framework requires a way to define distance between trees. The edit distance between trees, discussed in Section 6.1.1, is a relevant measure.

Table 6.3: Distances between best performing tree of each class of ACM CCS.

Class	A	B	C	D	E	F	G	H	I	J	K
A	0										
B	48	0									
C	85	71	0								
D	37	40	82	0							
E	37	44	79	27	0						
F	46	51	77	43	47	0					
G	35	38	77	23	29	45	0				
H	44	38	73	28	37	46	35	0			
I	36	37	74	16	24	43	24	26	0		
J	54	58	80	50	50	57	41	54	48	0	
K	76	76	96	79	73	78	71	76	71	77	0

Before we proceed to discuss the experiment setup to calculate diversity, it's necessary to lay out the theoretical setup for apportionment of diversity. Let $T = \{T_1, T_2, \cdots, T_n\}$ be the population of all possible trees, δ_{ij} be the distance between trees T_i and T_j, and then we can compile the distance matrix Δ having the (i, j) entry δ_{ij}. Suppose $\pi_1, \cdots, \pi_g$ are g probability measures defined on T. These measures represent g distinct sampling procedures, modeling the behavior of the g distinct classifiers. Fix $0 \leq \lambda_1, \cdots, \lambda_g \leq 1$ so that $\sum_i \lambda_i = 1$ and set $\pi = \sum_i \lambda_i \pi_i$. Here λ_i represents the sampling weight of classifier i. Based on Feigin and Alvo's framework [39], we can establish

$$\begin{aligned} H(\pi) &= \sum_{i,j} \lambda_i \lambda_j \pi_i^T \Delta \pi_j \\ &= \sum_i \lambda_i \pi_i^T \Delta \pi_i - \frac{1}{2} \sum_{i,j} \lambda_i \lambda_j (\pi_i - \pi_j)^T \Delta (\pi_i - \pi_j) \end{aligned} \tag{6.1}$$

The first quantity will be referred to as the *within-populations diversity*, and the second quantity will be referred to as *between-populations diversity*.

The experiment setup is as follows. Suppose g distinct populations of trees are of interest, and a sample of n_i trees is drawn with replacement from the i^{th} population. Denote the j^{th} tree in sample i by t_{ij}, where $j = 1, \cdots, n_i$. The *frequency vector* of the i^{th} sample is defined as $f_i = (f_i(1), f_i(2), \cdots, f_i(n))$ where $f_i(k)$ is the ratio of the number of occurrences of tree k in sample i to the total number of trees in sample i. The sampling weight w_i is defined as $w_i = n_i/n$, where $n = n_1 + n_2 + \cdots + n_g$. The *total frequency vector* is given by $f = \sum_i w_i f_i$. The sample frequency vectors $f_1, f_2, \cdots, f_g$ and the total frequency vector f record the relative frequencies of the sample

data in grouped format. For example, suppose $g = 2, n = 7$ such that $n_1 = 3, n_2 = 4$, and the sample data are observed as sample $S_1 = \{T_4, T_1, T_6\}$ and sample $S_2 = \{T_2, T_4, T_5, T_2\}$. Tree T_1 occurs once in S_1 of size $n_1 = 3$, thus $f_1(1) = 1/3$. We can calculate the $f_i(k)$ values for other trees in S_1 and S_2. The sampling weights in this example would be $w_1 = 3/7$ and $w_2 = 4/7$. A corresponding relative frequency table is shown in Table 6.4.

Table 6.4: Frequency Vectors.

k	T_k	$f_1(k)$	$f_2(k)$	$f(k)$
1	T_1	$1/3$	$0/4$	$1/7$
2	T_2	$0/3$	$2/4$	$2/7$
3	T_3	$0/3$	$0/4$	$0/7$
4	T_4	$1/3$	$1/4$	$2/7$
5	T_5	$0/3$	$1/4$	$1/7$
6	T_6	$1/3$	$0/4$	$1/7$

Substituting the population quantities π_i and λ_i with f_i and w_i in Eq. (6.1), respectively, we get the *total sample diversity* as follows:

$$\begin{aligned} H(f) &= \sum_{i,j} w_i w_j f_i^T \Delta f_j \\ &= \sum_{i} w_i f_i^T \Delta f_i - \frac{1}{2} \sum_{i,j} w_i w_j (f_i - f_j)^T \Delta (f_i - f_j) \end{aligned} \tag{6.2}$$

in which the first quantity is the *within-samples diversity (WSD)*, and the second quantity is the *between-samples diversity (BSD)*.

The technique described above parallels conventional ANOVA. The "total sum of squares (*TSS*)" is given by $H(f)$, "sum-of-squares within" is given by the first quantity (WSD) in Equation (6.2), and "sum-of-squares between (*SSB*)" is given by the second quantity (BSD) in Equation (6.2).

The basic null hypothesis of multi-sample testing is whether the probability measures for the samples are the same. This is equivalent to stating that the *between-samples diversity (BSD)* is 0. This is stated formally as:

$H_0 : f_1 = f_2 = \cdots = f_g$

which is equivalent to:

$H_0 : BSD = 0$

In conventional analysis of variance, the usual test statistic is the F-statistic given by:

$$F = \frac{[SSB/(g-1)]}{[SSW/(n-g)]} \tag{6.3}$$

The F test statistic, under assumptions of normality and independent sampling, has its null distribution of an F random variable with $v_1 = g - 1$ and $v_2 = n - g$ degrees of freedom. Here, g is the number of populations of trees under interest, and n is the total number of trees belonging to g populations.

The test statistic of an analogue for apportionment of diversity then can be defined as:

$$F = \frac{[BSD/(g-1)]}{[WSD/(n-g)]} \tag{6.4}$$

A difficulty in using this statistic is that there is no method in general to deduce its null distribution. However, a simulation approach like randomization analysis could be used to calculate a suitable cutoff.

The argument motivating randomization analysis is that if the null hypothesis were true, we can arbitrarily allocate the n sampled trees to g distinct collections of sizes $n_1, n_2, \cdots, n_g$ to represent the g classifiers. If the null hypothesis were false, then we support our experimental finding that classification is not arbitrary, and that each class has its own tree and that tree can explore that particular class's characteristics. In other words, distinct trees represent distinct classifiers.

We carry out the randomization analysis as follows:

1. Take a sample of n trees from g populations of trees.
2. Calculate WSD, BSD, and *total sample diversity*, as given by Equation (6.2). Calculate the corresponding F-statistic given by Equation (6.4). Refer to this value as $F_{unshuffled}$.
3. Randomly shuffle all n sampled trees. Then assign the first n_1 trees to collection 1, the next n_2 trees to collection 2, and so on, till the last n_g trees are assigned to collection g.
4. Using the shuffled data, calculate the corresponding F-statistic given by Equation (6.4).
5. Repeat steps 3 and 4, 1000 times, each time recording the F-statistic value.

6. Let $Range\_F_{shuffled}$ be the range of the 1000 F-statistic values recorded in step 5. Compare the range $Range\_F_{shuffled}$ with $F_{unshuffled}$.

In order to reject the null hypothesis, we need evidence against the hypothesis. This evidence would be seen to have been exhibited if a particular allocation of trees to classifiers produces a pattern or a value, which is largely different from an arbitrary allocation. So if the $F_{unshuffled}$ is a value largely different from $Range\_F_{shuffled}$, then the null hypothesis would be rejected. And correspondingly, this would support our experimental finding that distinct trees represent distinct classifiers.

We use the trees we obtained from $g = 11$ ACM CCS classifiers. Specifically, we selected $n_i = 20$ trees from each classifier to form a total of $n = 220$ trees. Randomization analysis was performed on 5 independent runs of our GP classification experiments, where each run is based on a different random seed used in the GP experiment setup. The analysis results are shown in Table 6.5.

From Table 6.5, we can see that the F-statistic for unshuffled data, $F_{unshuffled}$, has a large value and does not fall into the range given by $Range\_F_{shuffled}$. This shows that GP was able to discover distinct trees for different classifiers. Thus, we reject the null hypothesis that classifiers are formed by some arbitrary allocation of trees.

Table 6.5: $F_{unshuffled}$ and $Range\_F_{shuffled}$ for the data from 5 runs.

Random Seed	$F_{unshuffled}$	$Range\_F_{shuffled}$
1	1.1080242	0.0017144 – 0.01794048
2	0.9308702	0.0074398 – 0.0172862
3	0.8947737	0.0081242 – 0.0172675
4	0.7861073	0.0078098 – 0.0175876
5	1.0447248	0.0071262 – 0.0175256

It is worth noticing that results shown in Table 6.5 could be the results of some trees belonging to a few classifiers being the same, while the rest of the trees are distinct. So we need to check further whether the trees of each classifier are distinct. In order to do this, we conduct the randomization analysis for every pair of classifiers. The top 20 trees from every pair of classifiers were chosen and analyzed. The results show that GP discovered distinct trees for every pair of classifiers.

6.1.3 Central Tree for a Tree Population

A central tree is defined as a tree that captures the overall behavior of the population. This definition is analogous to that of the statistical mean of a population. We study two types of central tree in our analysis: 1) median tree, which is a member tree of the population; and 2) mean tree, which need not be a member tree of the population.

The main motivation of identifying or constructing a central tree is to find the "center point" of the data space defined by the sample trees. This notion of "center point" can be used in various ways to analyze the trees to provide valuable insight into some properties of the GP trees. The central tree can be used as a representative of a population of trees from one GP generation. Thus analyzing the central tree of different generations can be used to study evolution of trees from one generation to another. A central tree also can be used to study the total variance of a population of trees. Here total variance is defined as the sum of distances between the central tree and all trees in the population. This measure of variance can be used to analyze the convergence of GP trees.

Based on edit distance between trees, we explore different techniques to identify or construct the central tree. Particularly, to identify the median tree, we have used: 1) smallest maximum distance; 2) smallest median distance; and 3) smallest average distance. To construct a mean tree, we use a method which constructs the tree using frequent nodes.

Smallest maximum distance: Given a tree T_i belonging to a population of n trees, we calculate the distance between T_i and all the other $n-1$ trees in the population and record the maximum distance δ_{max_i}. Repeat this process for all n trees, and the tree with the smallest maximum distance to all other trees is the median tree of the population.

Smallest median distance: Given a tree T_i belonging to a population of n trees, we calculate the distance between T_i and all the other $n-1$ trees in the population and calculate the median of all these $n-1$ distance values. The median is defined as the middle of the distribution and is less sensitive to extreme values in the distribution. When there is an odd number of distance values, the median is simply the middle number. And when there is an even number of distance values, the median is the mean of the two middle numbers. Repeat this process for all n trees, and the tree with the smallest median distance to all the others trees is the median tree of the population.

Smallest average distance: Given a tree T_i belonging to a population of n trees, we calculate the distance between T_i and all the other $n-1$ trees in the population and calculate the average of all these $n-1$ distance values. Repeat this process for all n trees, and the tree with the smallest

average distance to all the other trees is the median tree of the population.

We applied the above methods to the ACM CCS's 11 classes' good performing trees separately in order to identify the median tree for each class's tree population. To check how accurately the median tree represents the corresponding tree population, we identified the 3 most frequent operators and 3 most frequent types of evidence for each tree population. And then we compared this data with that of the median trees generated by the above discussed 3 methods. The results show that the top 3 operators and evidence of the median tree match that of the population for all 11 classes, which verifies that the median tree is able to capture the characteristics of the corresponding tree population. We also noted that function '+' is the most frequent function and evidence Full_Okapi is the most frequent evidence for the median trees. This is expected because the '+' operation is summing and reinforcing relations while the Full_Okapi evidence is the best type of evidence as shown in Table 5.3.

To construct a mean tree, we follow a method proposed by Wang [115], which uses frequent nodes of the trees in the population. A frequent node, in a population of n trees, is defined as the node which occurs at a particular position in at least $n/2$ trees. Each tree in the population is traversed to identify the nodes at each position in the tree, where position is the index given by the breadth-first traversal. The breadth-first traversal of a tree visits the nodes in the order of their depth in the tree. Breadth-first traversal first visits all the nodes at depth zero (i.e., the root), then all the nodes at depth one, and so on. At each depth the nodes are visited from left to right. If at a particular position, a node occurs in at least n/2 trees, then that node is a frequent node. A tree constructed using all frequent nodes is the mean tree for that population of trees. The mean tree can be used in analyzing the total variance of the tree population. Wang [115] proposed a top-down approach to build a mean tree using frequent nodes in a set of unlabeled trees. We adapted Wang's method to build a mean tree for labeled trees. The method is as follows:

1. A sample of n trees is withdrawn from the tree population.
2. Starting from the root, assign an index number to each node of the tree. The root has index number 1. For a node with index number i, its left child has the index number $2 \times i$, and its right child has the index number $2 \times i + 1$.
3. At each index, calculate the frequency of each occurring label at that index.
4. All the nodes that occur at least $n/2$ times at an index are selected. The labels for these nodes are the ones that occur the maximum times at that index in the population of trees.

5. Form the mean tree from these frequent nodes.

Figure 6.5 shows an example of the mean tree for an artificial data set of 3 trees, A, B, and C. At each index, the nodes that occur in at least 2 trees are selected as the frequent nodes. For example, at index 1, nodes with label '+' occur in 2 trees, while at index 2, nodes with label '*' occur in 3 trees. At index 5, nodes with label 'S' occur in 2 trees but the node with label '+' only occurs in 1 tree. So at index 5, the node with label 'S' is selected as the frequent node. The node with label 'c' only occurs in 1 tree at the position with index 11.

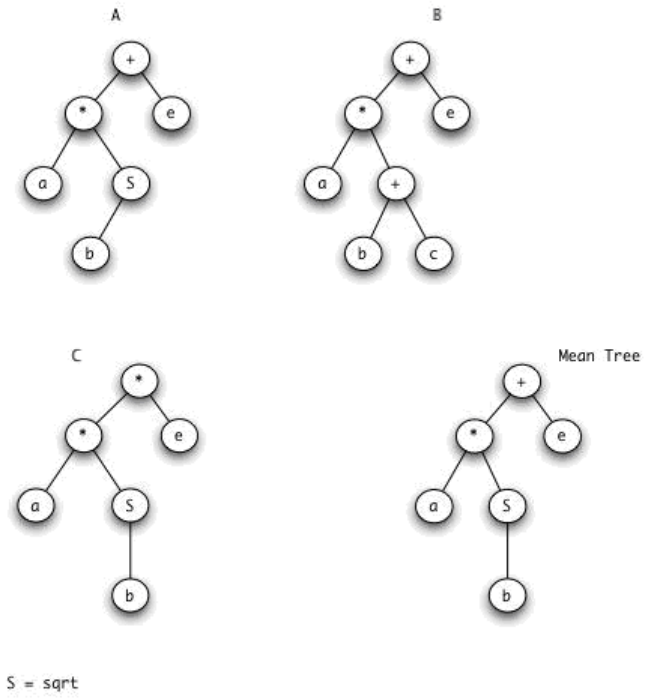


Figure 6.5: Sample mean tree for an artificial data set of 3 trees.

It is worth noticing that it is possible that at some index, there is no node which occurs in more than $n/2$ trees. If this occurs, we simply pick the most frequent node label that occurs at that index. In case of ties, we pick the node label which occurs more frequently in the whole population.

After obtaining the mean tree, we calculate the distance of the mean tree with all the trees in the population. Summing these distances together gives us the total variance of the population. We expect that the mean tree is the closest tree to all the trees in the population. This is formally stated as:

$$\sum_i d(meantree, t_i) < min(\sum_{i,j} d(t_j, t_i)) \tag{6.5}$$

In other words, the sum of distances between the mean tree and all the trees in the population is the smallest, comparing with even the minimum sum of the distances between any tree t_i with all other trees in the population.

We show an example of this mean tree being the closest to all the trees in the population by analyzing top 20 performing trees of class D of ACM CCS. In Table 6.6, the second column is the sum of the distance between the tree in the first column and all the other trees in this class's tree population. We can see that the mean tree is the closest to all the trees and is the "center point" for the tree data set of class D. The trees with IDs 1–11 are the smallest 11 sums of distance to the rest of the trees in the population. The "center point" of a data set should ideally be an average of features of the data set and should capture the main characteristics of the data set in consideration. So the mean tree is a good representative for the whole data set.

Table 6.6: Comparison between sum of distance between mean tree and other trees.

Tree	Sum of distance to other trees
Mean Tree	**113**
1	432
2	434
3	444
4	447
5	468
6	470
7	472
8	472
9	476
10	488
11	494

Similar to what we did with median tree, we also analyzed the frequencies of evidence in the mean tree and we noticed that it was similar to that of the population.

We also analyzed the fitness of the mean trees. The mean tree for each class of ACM CCS is constructed. For most classes, the mean tree is a member of the population of good performing trees. We were able to obtain a mean tree which is not a member of the population for classes B and H. Table 6.7 shows the mean tree's fitness compared with the good performing tree for these

two classes. As can be seen from this table, the mean tree's fitness is slightly worse but very close to the good performing tree of the population.

Table 6.7: Mean Tree's Fitness.

Class	Good Performing Tree Fitness	Mean Tree Fitness
B	82.03	80.58
H	92.60	92.47

The total variance is one of the measures to study the variation in a population. It would tell us how the tree population varies from generation to generation. We calculated the total variance for each of the generations, from 0 to 30, of the GP run. The results are shown in Figure 6.6.

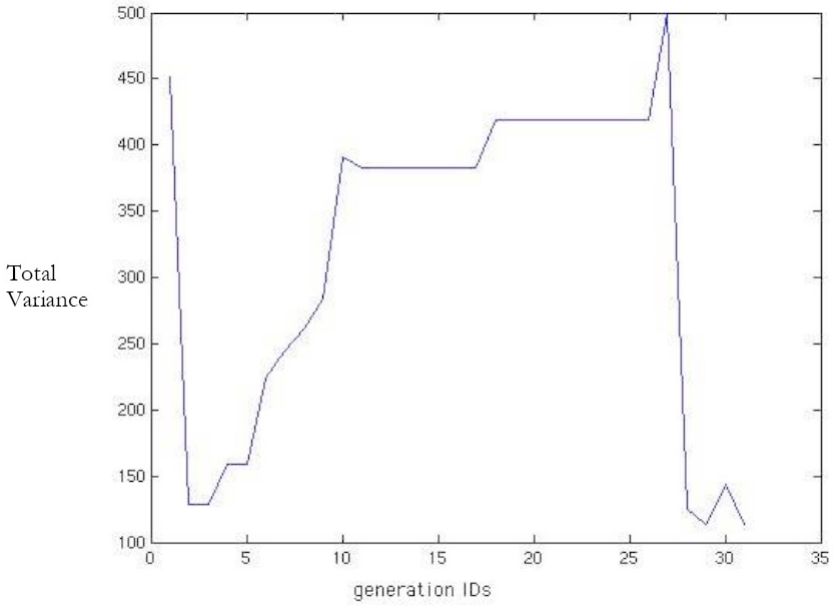


Figure 6.6: Change in total variance along with GP's evolution generation by generation.

From Figure 6.6, we can see that initially the total variance is very large. This is because random trees are generated to fill the population when GP starts to run. The mean tree for the initial population must be very far from the trees in the population. The initial random trees are also of very small size. Along with the evolution, the total variance drops down. And then it starts to increase. The increase is because, along with GP's evolution, trees are getting larger. Recall that when calculating edit distance between trees, a unit cost matrix is used, where insert, delete, and change edit operations each have a cost of 1. So the edit distance is influenced by the size of the trees. The initial trees are of small size, so the distance between trees is small even if those

trees are way apart. But for larger trees, the distance might be larger even though 90% of the trees are the same. This fact is shown in the populations of generations from 6 to 11 as the size of the trees increases dramatically. A crossover genetic operation might increase the size of one tree drastically while decreasing the size of the other tree so the distance between the two trees shoots up. This is observed from generation 26 to 27 where there is a sudden increase of the total variance. From our earlier analysis of edit distance between trees as shown in Table 6.2, we noticed that the top trees of the first few generations are the same, and then the distance increases as a step function after the first few generations, and finally the trees stabilize after generation 17. Even though the top tree from different generation's population might be the same, the populations can still vary due to various genetic operators used in GP. This is reflected in the last few generations of Figure 6.6 where total variance is changing even though the best tree remains the same (as shown in Table 6.1). From generation 26 onwards, the mean tree is able to represent the tree population more closely. The minimum total variance is at generation 29 while the last generation's total variance is larger than the minimum. This might be due to genetic operations that change the composition of individuals in the population.

6.2 Common Subtree Mining

Given a population of good performing trees, we identified common subtrees aiming to gain more insight into subtrees' influence on the tree performance. We explored if subtrees exist which impact the ability of GP to evolve high performing trees. More specifically, we identified subtrees and relate them to the success of a tree. The approach we adopted is based on mining common subtrees in a population of trees. This is based on the building block hypothesis of GP. The hypothesis states that smaller highly fit tree substructures (subtrees) or building blocks combine to form trees, which form the solution(s) to the problem under consideration. Figure 6.7 shows the building blocks for 3 artificial trees A, B, and C. In order to identify the building blocks, we developed a tree miner and mined frequent subtrees from the trees in a population of high performing GP trees. The frequent subtrees might be the reason why a high performing GP tree performs better than other trees (which do not have high performance). After identification of these subtrees, we analyzed them to see if they relate to the performance of the corresponding tree, or in other words, if these subtrees are the building blocks.

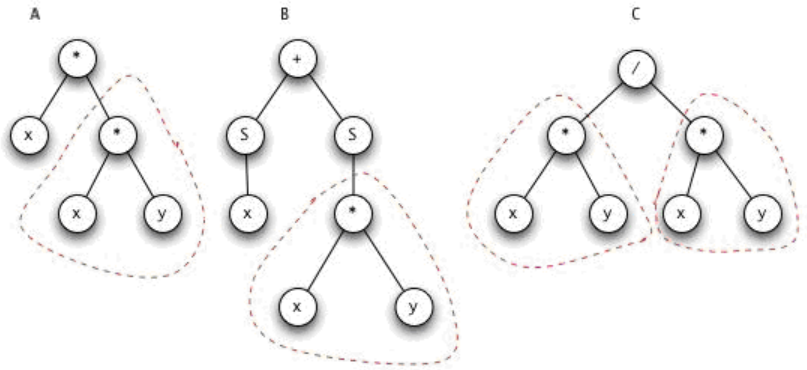


Figure 6.7: The subtrees marked by thin dashed red lines form the building blocks of these trees. If A, B, and C are high performing trees, subtree $(x \times y)$ might be the reason for their performance.

6.2.1 The Building Block Hypothesis

GA and GP probe a large search space to find a good solution for the problem under consideration. Thus it has attracted researchers' interest to know how these search methods arrive at acceptable solutions. Holland [53] has modelled the method by which GA works through the schema theorem. And works such as [46, 107, 106] have further extended this work.

In the GA schema theory, a schema is a template for describing chromosomes (bit strings). For example, *1 matches 01 and 11, i.e., 01 and 11 are instances of *1. The schema encodes both the position and values of a number of specified bits, leaving the bits in other positions unspecified. The number of specified bits in a schema is referred to as the *order* of the schema. The distance between outermost specified bits is defined as the *length* of the schema. A schema has *average fitness* equal to the mean fitness of all its instances. The schema theorem states that short, low-order schema with high average fitness will increase in number along the evolution. Fitter schemas are likelier to survive. In other words, if individuals matching a particular schema have higher fitness values than is the average for other individuals within the population then these individuals are more likely to pass their genetic material into future generations. Shorter and lower-order schemas tend to be preserved by crossover and mutation events. Thus, as a result of fitness pressure and the chance of preserving of low order and short schemas, it has been shown that the number of individuals matching low order, short, and high fitness schema within a population will increase generation by generation. At the same time, the number of individuals matching low order, short schemas with low fitness values will decrease.

Based on the schema theorem, the building block hypothesis states that instances of good schemas are combined to make better, higher order, and longer schema. The lower order, short, and high fitness schemas form a set of basic building blocks which are combined to form larger building blocks, which can again combine into effective solutions.

The schema theorem of GA provides strong theoretical evidence for modelling how GA works. The theorem also provides support for the building block hypothesis. In the GP field, different works have been performed in the aim of deriving a schema theorem for GP. But the variable size and structure of GP individuals makes it difficult to define properties like position and length for schemas, which are a composition of functions and terminals forming GP trees. On the other hand, the crossover operation can change the position of subtree components. Thus it is even more difficult to determine how schemas will be represented within a population along with the evolution. Several works have tried to predict how different schemas will be represented within a population from generation to generation. For example, in [64] Koza presented an informal argument as to how Holland's schema theorem can be applied to GP tree structures. In recent years, works shown in [68, 76, 86, 85, 87, 88] have formulated an exact schema theorem for GP for predicting the expected numbers of schemas from one generation to the next. More details of these works can be found in [68].

In addition to the works on schema theorem for GP, different researches such as [83, 86, 95, 99] also have been performed to analyse the building block hypothesis for GP with the aim of discovering if building blocks exist within GP and what form they might take. These works have been based on the theoretical work of the schema theorem. However, there is experimental work performed outside the schema theorem to identify components which readily occur within populations [67]. In this work, the frequency of terminals and functions were tracked throughout the evolution of different generations.

Some recent works have cast doubt upon the usefulness of the schema theory for modelling the workings of GA [19, 41, 114]. The doubt is mainly because of the difficulty of using the schema theorems to predict how evolutionary systems like GA will behave over a number of generations, rather than in the progression from only one generation to the next. Due to particular effects of the GP crossover operation differing in GA crossover operations, doubt also has been cast over the applicability of the building block hypothesis for GP [82].

Regardless of the outcomes of the GP schema theorem and building block hypothesis debate, the building block hypothesis remains to date the best explanation as to how GP is capable of producing

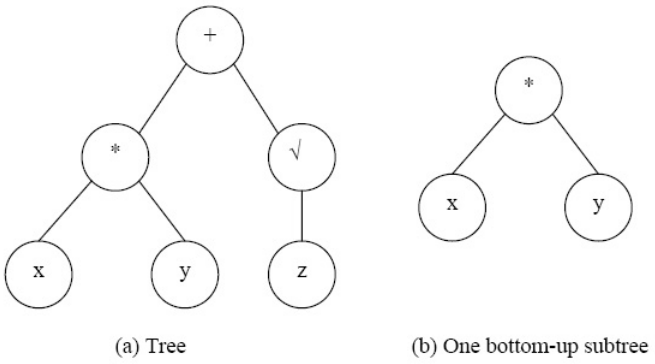


Figure 6.8: Bottom-up Subtrees.

solutions to complex problems, as has been demonstrated in many previous studies [6, 65].

6.2.2 Tree Miner

Based on the building block hypothesis, we tried to mine common subtrees within a population of high performing trees. We wanted to find the subtrees that occur most frequently in the pool of good result trees. What this means is we want to find out which fusions of similarity measures work best, and thus discover the factors that contribute to good classification.

Bottom-up Subtree

The subtrees we were trying to mine are of the type of bottom-up subtrees [18, 112]. Recall from Section 6.1.1 that the GP trees are rooted trees. For a rooted tree T with vertex set V and edge set E, a rooted tree T' with vertex set V' and edge set E' is a *bottom-up subtree* of T if and only if 1) $V' \subseteq V$, 2) $E' \subseteq E$, 3) the labeling of V' and E' is preserved in T', 4) for a vertex $v \in V$, if $v \in V'$ then all descendants of v (if any) must be in V', and 5) if T is ordered, then the left-to-right ordering among the siblings in T should be preserved in T'. Intuitively, we can obtain a bottom-up subtree T' (with the root v) of T by taking a vertex v of T together with all v's descendants and the corresponding edges. Figure 6.8 gives an example of the bottom-up subtree. For the tree (a) on the left, tree (b) is a bottom-up subtree.

Canonical Representations for Trees

To implement the tree miner, we first represent the trees using the canonical representation. A canonical representation is a unique way to represent a labeled tree and facilitates functions such

as comparing two trees for equality and enumeration of subtrees. The specific canonical representation we adopted is called Luccio's *pre-order string* format. Luccio et al. [73, 72] gave the following recursive definition for a *pre-order string*: 1) for a rooted ordered tree T with a single vertex r, the pre-order string of T is $S_T = l_r 0$, where l_r is the label for the single vertex r, and 2) for a rooted ordered tree T with more than one vertex, assuming the root of T is r (with label l_r) and the children of r are $r_1, \cdots, r_K$ from left to right, then the pre-order string for T is $S_T = l_r S_{T_{r_1}} \cdots S_{T_{r_K}} 0$, where $S_{T_{r_1}} \cdots S_{T_{r_K}}$ are the pre-order strings for the bottom-up subtrees $T_{r_1}, \cdots, T_{r_k}$ rooted at $r_1, \cdots, r_K$, respectively. For example, the string for the rooted ordered tree in Figure 6.8(a) is $S_T = + \times x 0 y 00 \sqrt{} z 000$. The reason to choose Luccio's pre-order string representation is that one can easily compute the pre-order string of any bottom-up subtree from the total pre-order string. For example, when we scan S_T from label $\times$ until we get an equal number of vertex labels and the symbol 0, the resulting substring $(\times x 0 y 00)$ is exactly the pre-order string for the bottom-up subtree rooted at $\times$. With such a useful property, the bottom-up subtree matching problem can be reduced to the substring matching problem, which can be solved efficiently by algorithms for string matching, e.g., by using the *suffix array* data structure [52]. Also this string encoding is more space-efficient than traditional data structures to store rooted ordered trees [123].

Mining Frequent Subtrees

From the bottom-up subtree's definition, we can see that the number of bottom-up subtrees for a rooted tree T is equal to the size of the tree T, i.e., the number of nodes T has. Based on this, the following algorithm details our frequent bottom-up subtree miner.

1. Encode the database of trees in the Luccio pre-order string format.
2. For every node in each tree, store the pre-order index in the tree along with its corresponding tree ID. Note that every pre-order index points to the root of a bottom-up subtree.
3. Sort the bottom-up subtrees given by the node indices by comparing the string encodings of the subtrees which start at those indices.
4. Store the occurrence of the bottom-up subtrees in a record.
5. Count the frequency of all the bottom-up subtrees which occur in more than threshold percentage of the tree population. These are the frequent bottom-up subtrees.
6. Sort the frequent bottom-up subtrees on the basis of string length.

7. The top hit is the largest frequent subtree, the second hit is the second largest frequent subtree, and so on.

The input to the tree miner is the population of trees we would like to mine frequent subtrees from. The output of the tree miner yields all the frequent subtrees of different sizes. The key to this tree miner is that the number of the bottom-up subtrees for a rooted tree is bounded by the number of nodes in the tree.

6.2.3 Tree Miner Results

Frequency of Evidence

Before mining the largest subtrees, we first mined the frequency of each type of evidence in the set of data which contains the best tree of each of the eleven classes of ACM CCS. Figure 6.9 shows the result. As we can see from this figure, full_Okapi has the maximum frequency which conforms to the result for baselines we saw earlier in Table 5.3.

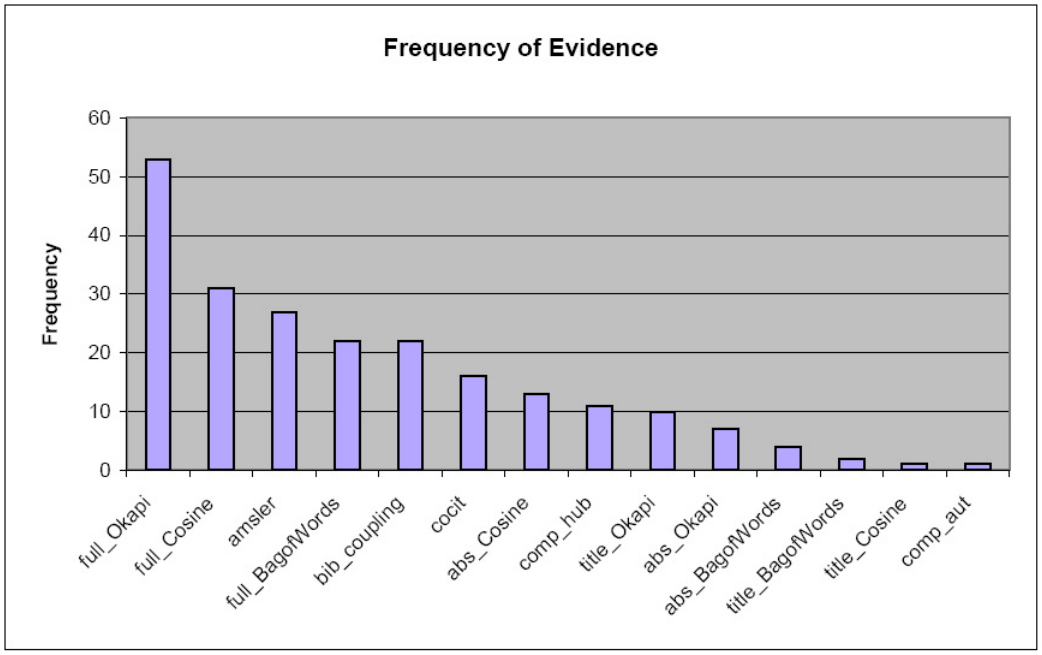


Figure 6.9: Frequency of Evidence for the best 11 trees of ACM CCS.

Frequency of Functions

In the set of data which contains the best tree of each of the eleven classes of ACM CCS, we also mined the frequency of the functions which are used to combine those types of evidence. The result is shown in Figure 6.10. From this figure, we notice that the function '+' has the maximum frequency. The order of the frequency for the functions seem to conform with each function's meaningful operations on relations. For example, the + and * operations are summing and reinforcing the relations, respectively. On the other hand, the / operation accomodates inverse relationships, while the sqrt operation serves to scale the values.

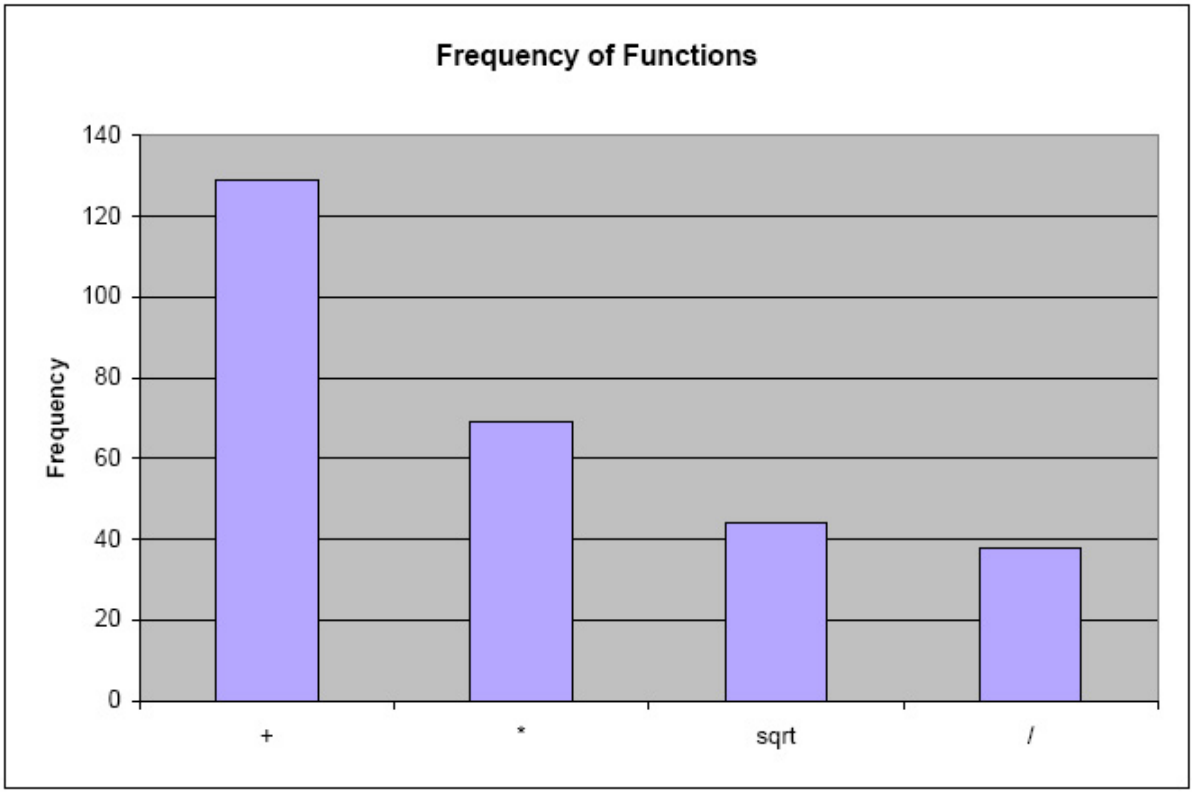


Figure 6.10: Frequency of Functions for the best 11 trees of ACM CCS.

Largest Common Subtree Results

For each class of the ACM CCS, we extracted the top 20 performing trees and mined for the largest common subtree within this set of top 20 trees. After obtaining this largest common subtree, we analyzed its performance and compared that with the performance of the good performing tree. Then for this largest common subtree, we extracted all subtrees of size 3 and up, and analyzed the performance of these subtrees.

Figure 6.11 shows the largest common subtree we obtained for class A. The part indicated by the polygon shows the best performing subtree of this largest common subtree. For this class, the largest common subtree performs the best when compared with all its possible subtrees of size 3 and up. We can see that this largest common subtree is still relatively large. Figure 6.12 shows properties of the largest common subtree for class A. From Figure 6.12(a), we notice that the performance of the largest subtree is close to the good performing tree. Among all the possible subtrees of size 3 and up for this largest common subtree, the largest common subtree itself performs the best. Figure 6.12(b) shows how this largest common subtree (at the same time the best performing subtree) classifies class A's documents compared with the good performing tree. The part marked by horizonal lines shows the proportion that the good performing subtree classifies documents consistently compared with the good performing tree. The diamond marked part indicates document classification mismatch between the good performing tree and the best performing subtree. We can see that for most documents, the best performing subtree's classification decision matches that of the good performing tree.

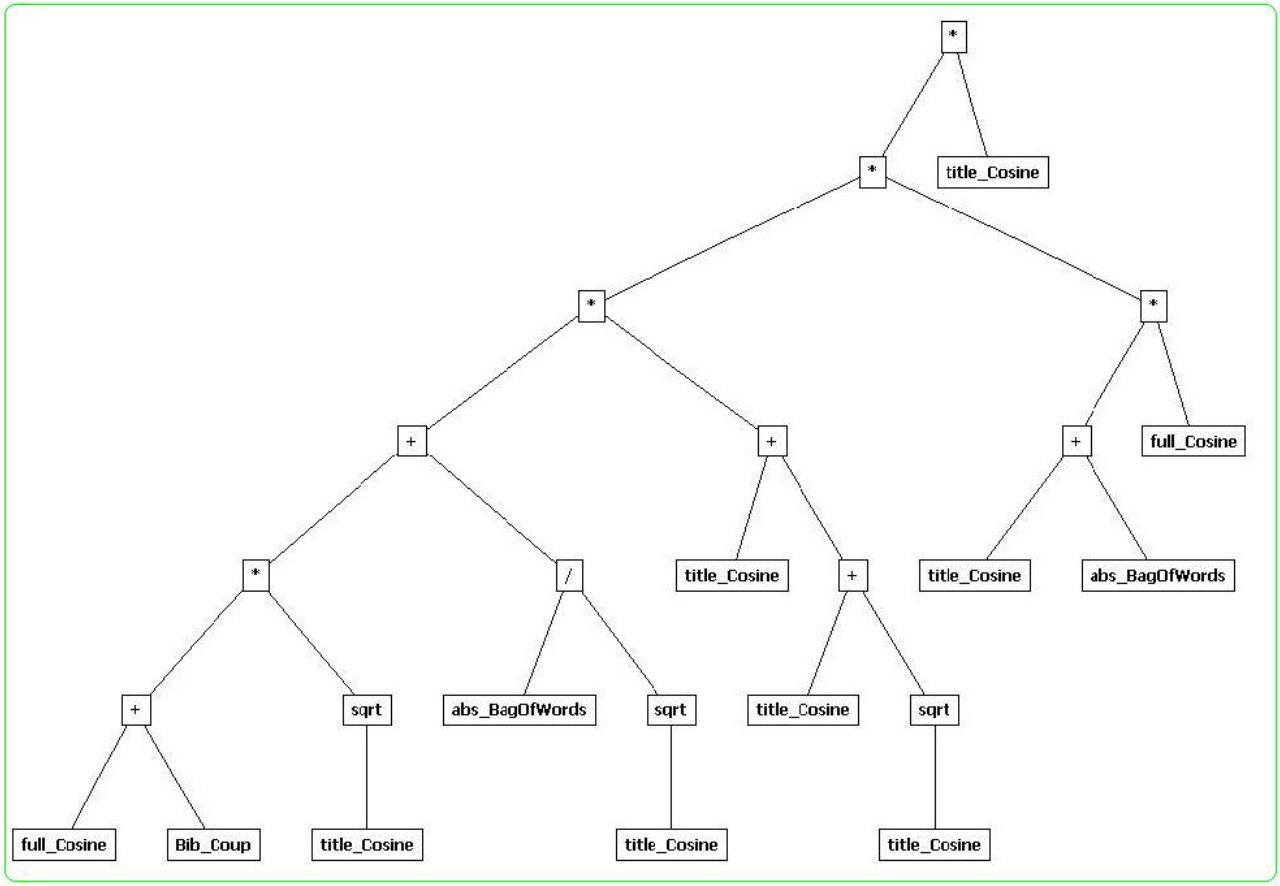
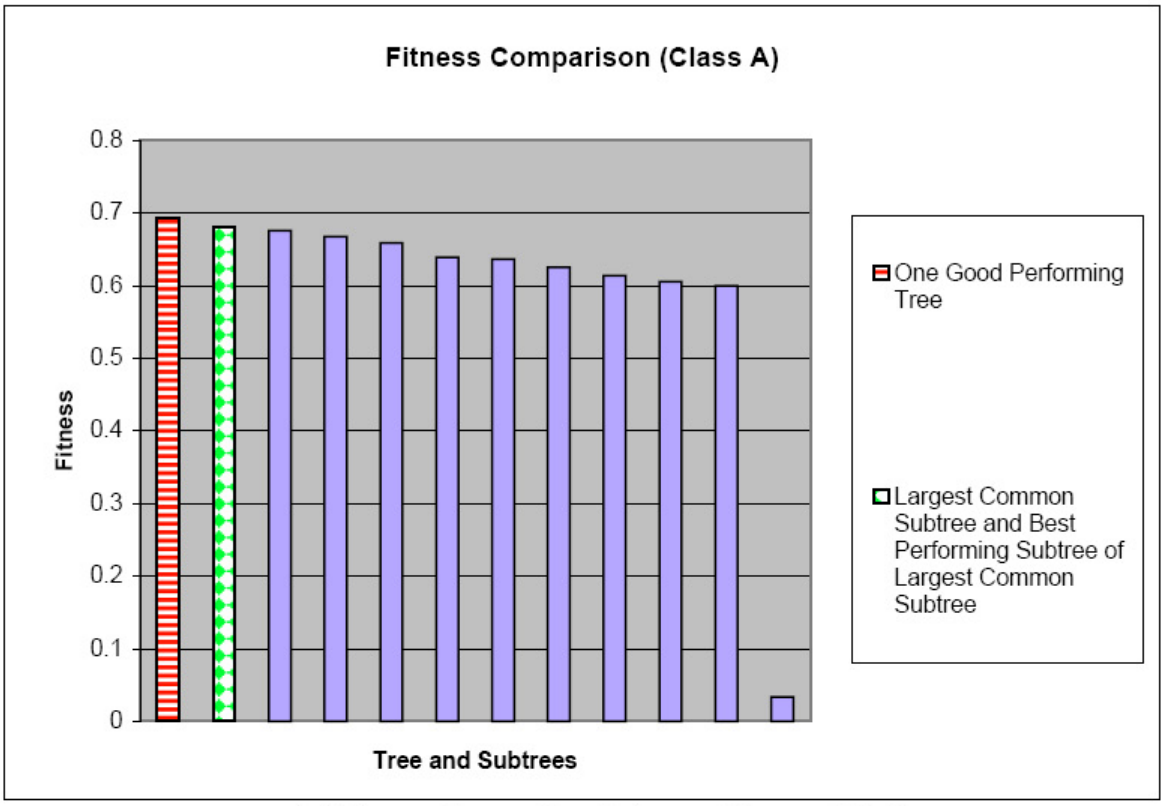
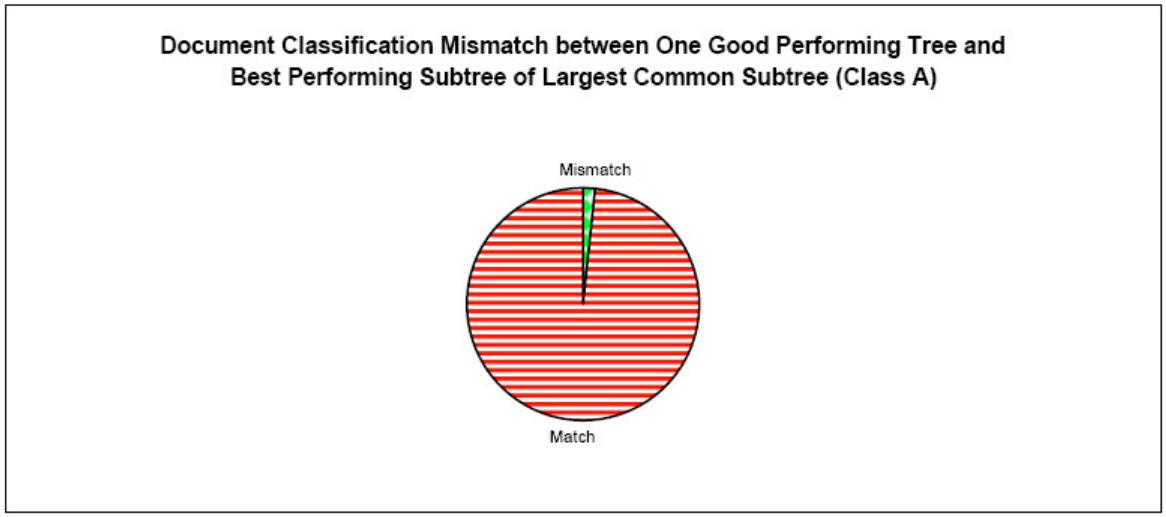


Figure 6.11: Largest Common Subtree for Class A's Good Performing Trees Population.



(a) Fitness Comparison between Tree and Subtrees



(b) Tree and Best Performing Subtree Document Classification Mismatch

Figure 6.12: Properties of the Largest Common Subtree for Class A's Good Performing Trees Population.

Figure 6.13 shows the largest common subtree we obtained for class B. The best performing subtree of this largest common subtree is (full_Okapi + comp_hub). Figure 6.14 shows properties of the best performing subtree (the second bar, indicated by diamond pattern) of the largest common subtree for class B. From Figure 6.14(a), we notice that the performance of the largest subtree is worse than that of the good performing tree. But the best performing subtree's performance is relatively close to the good performing tree. From Figure 6.14(b), we can see that for most documents, the best performing subtree's classification decision matches that of the good performing tree.

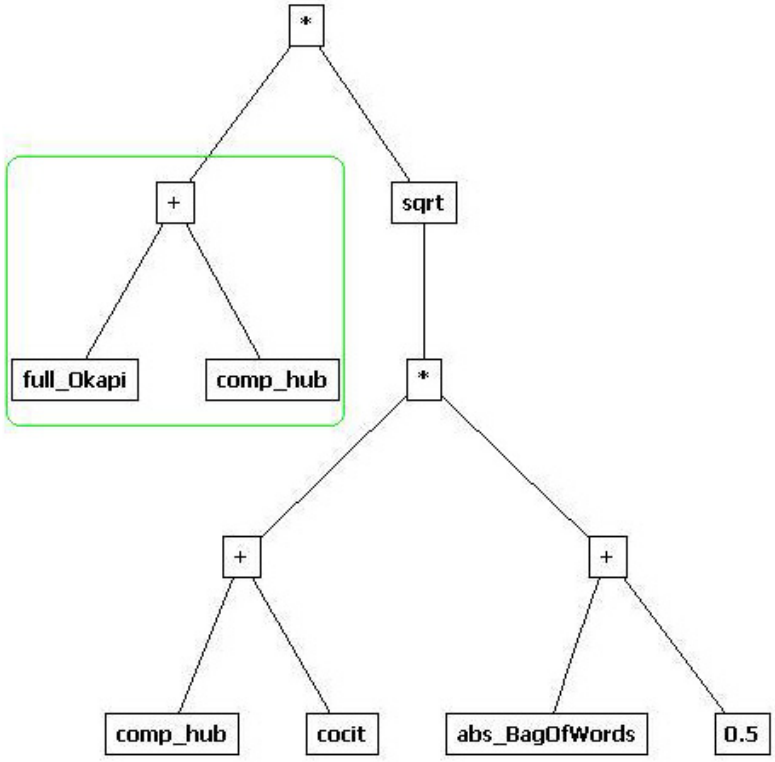
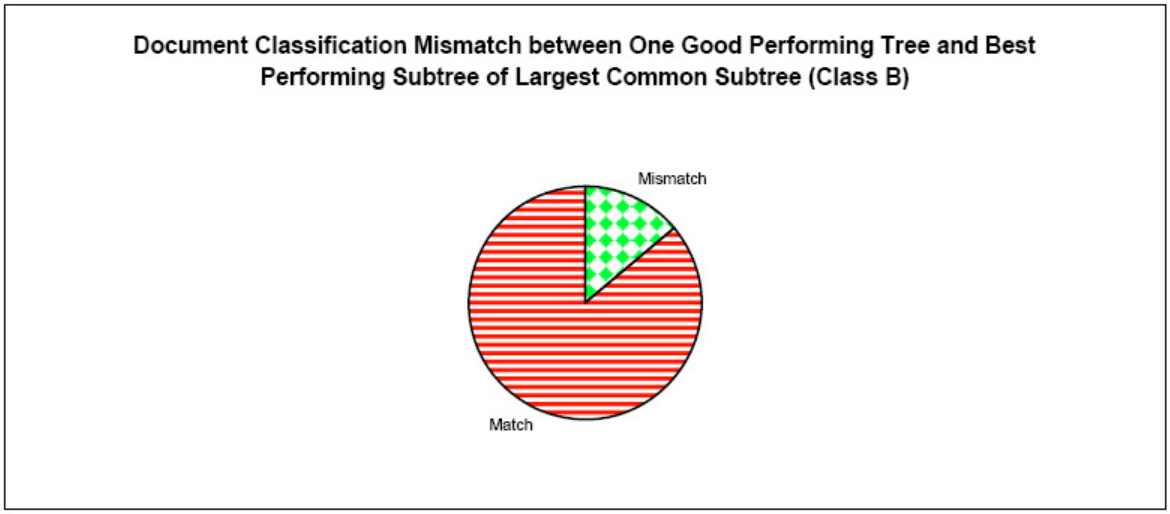


Figure 6.13: Largest Common Subtree for Class B's Good Performing Trees Population.

(a) Fitness Comparison between Tree and Subtrees



(b) Tree and Best Performing Subtree Document Classification Mismatch

Figure 6.14: Properties of the Largest Common Subtree for Class B's Good Performing Trees Population.

Figure 6.15 shows the largest common subtree we obtained for class C. The best performing subtree of this largest common subtree is itself. In this common subtree, we can notice that the instances of full-based type of evidence appear frequently. We analyzed the baseline and noticed that full-based type of evidence, full_BagOfWords, full_Cosine, and full_Okapi, are the best 3 types of evidence for this class. Figure 6.16 shows properties of the best performing subtree (the second bar, indicated by diamond pattern) of the largest common subtree for class C. From Figure 6.16(a), we also notice that the performance of the largest subtree is close to that of the good performing tree. From Figure 6.16(b), we can see that for most documents, the best performing subtree's classification decision matches that of the good performing tree.

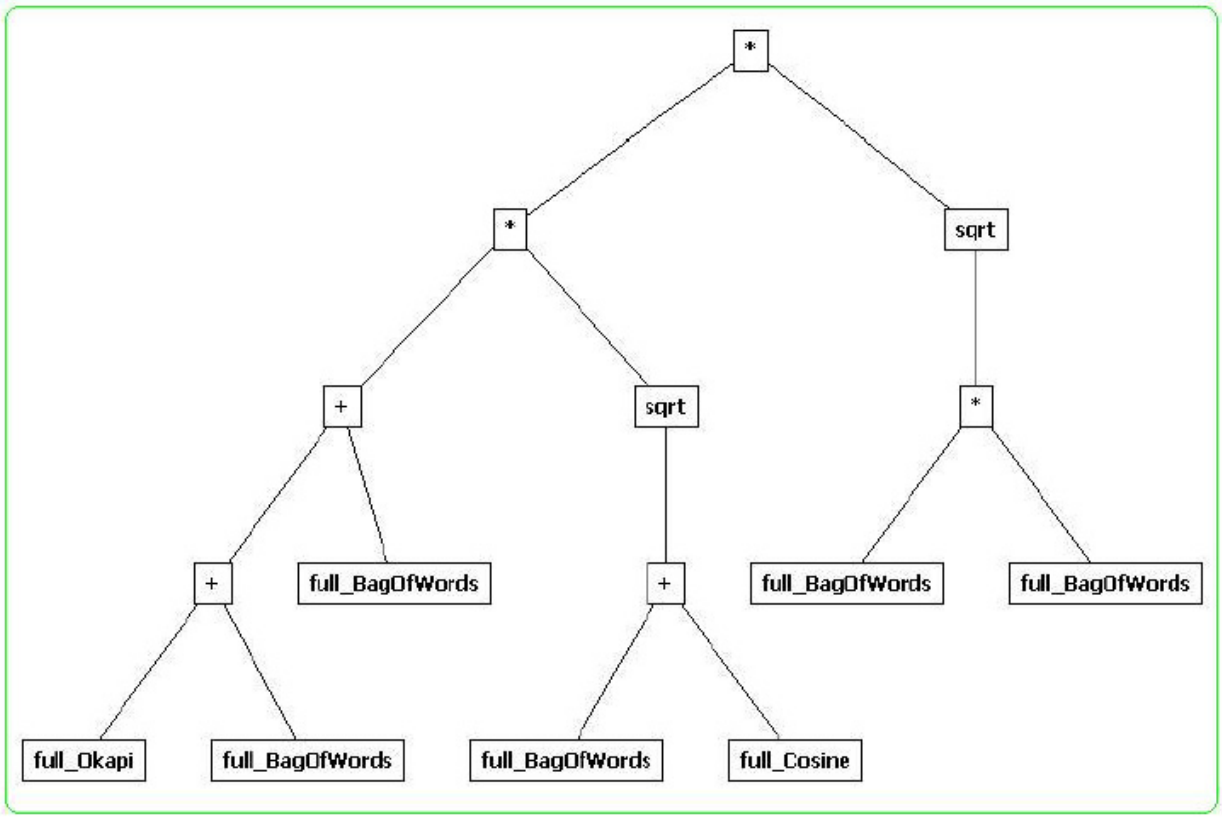
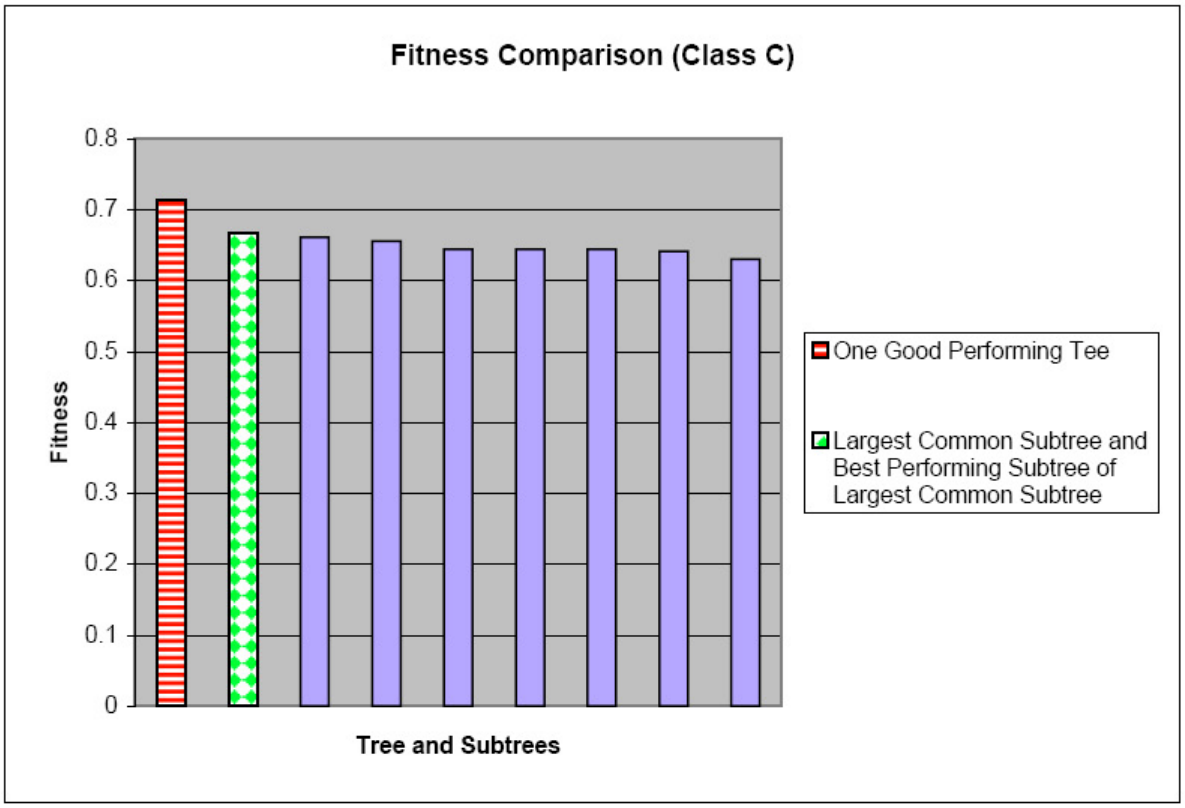
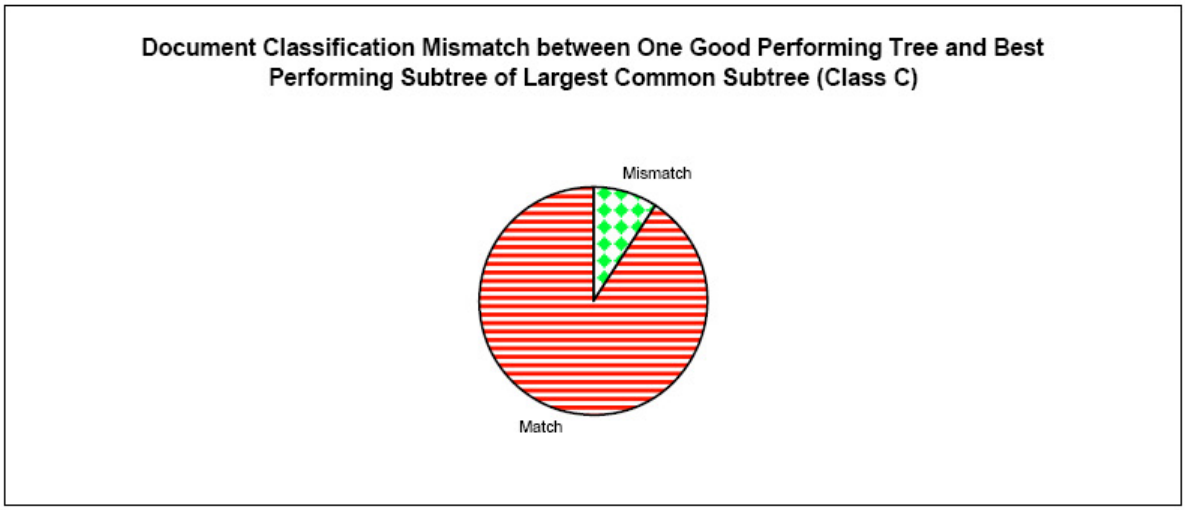


Figure 6.15: Largest Common Subtree for Class C's Good Performing Trees Population.



(a) Fitness Comparison between Tree and Subtrees



(b) Tree and Best Performing Subtree Document Classification Mismatch

Figure 6.16: Properties of the Largest Common Subtree for Class C's Good Performing Trees Population.

For class D, the largest common subtree, which is shown in Figure 6.17, is much simpler. The only size 3 and up subtree of this largest common subtree is itself. From Figure 6.18(a), we can see that this subtree's performance is very good. And as shown in Figure 6.18(b), it classifies class D's documents consistently compared with the good performing tree.

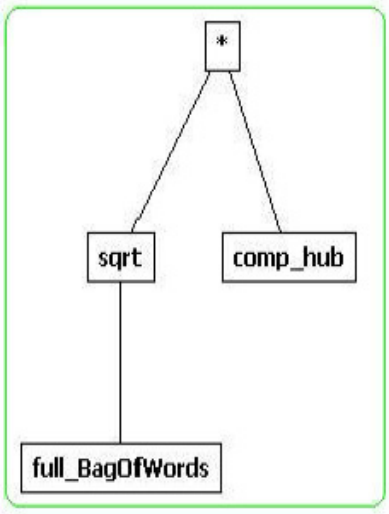
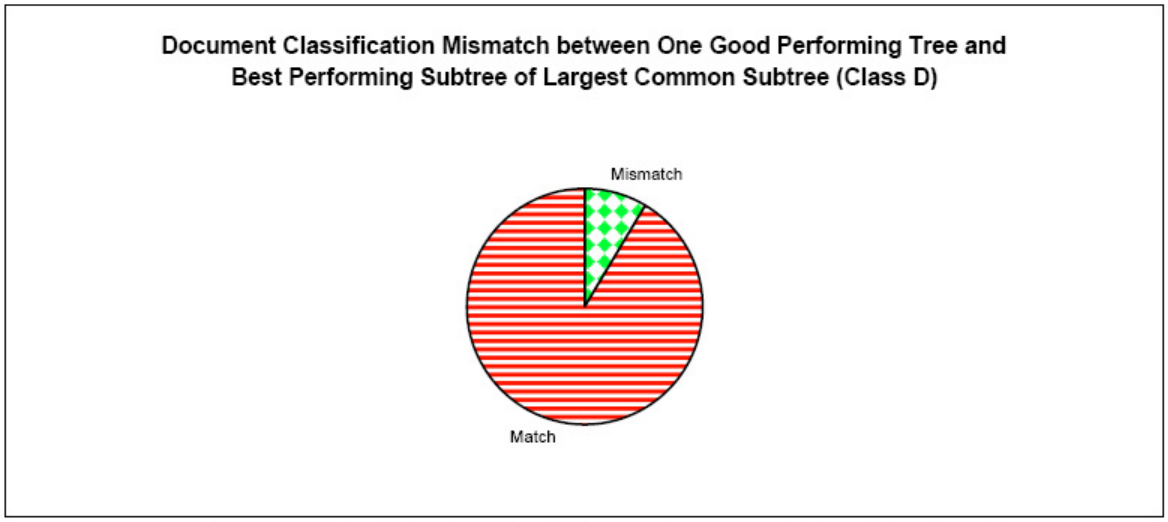


Figure 6.17: Largest Common Subtree for Class D's Good Performing Trees Population.

(a) Fitness Comparison between Tree and Subtrees



(b) Tree and Best Performing Subtree Document Classification Mismatch

Figure 6.18: Properties of the Largest Common Subtree for Class D's Good Performing Trees Population.

Class E's largest common subtree is shown in Figure 6.19. This largest common subtree itself is the best performing subtree compared with its subtrees. But for this class, as shown in Figure 6.20(a), the largest common subtree's performance is not that close to the good performing tree. Also as shown in Figure 6.20(b), the classification mismatch between the best performing subtree and the good performing tree for this class is higher when compared with other classes. We think this result is due to the very small size of this class, thus GP discovered good performing trees that have much lower fitness value compared with other classes.

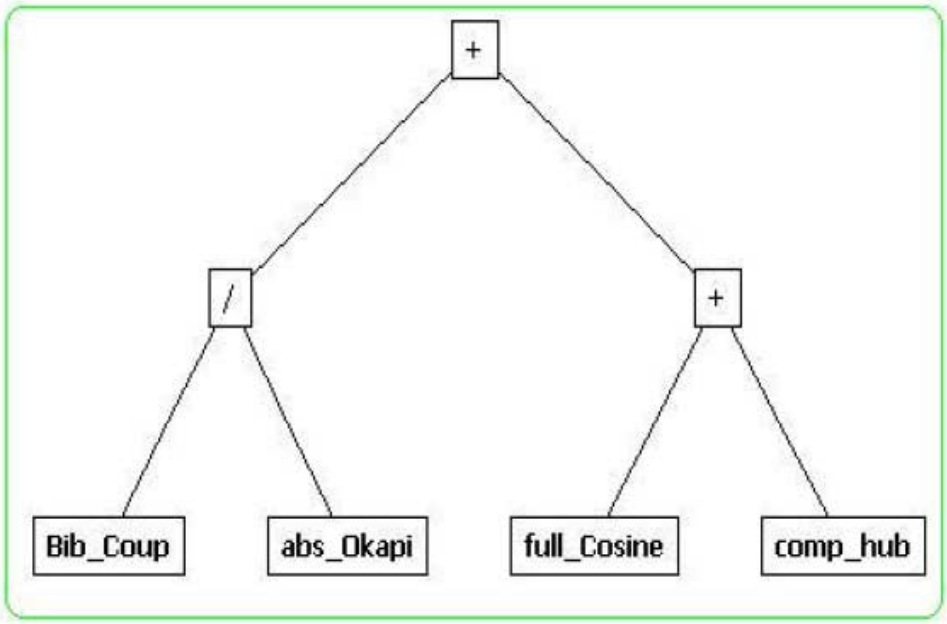
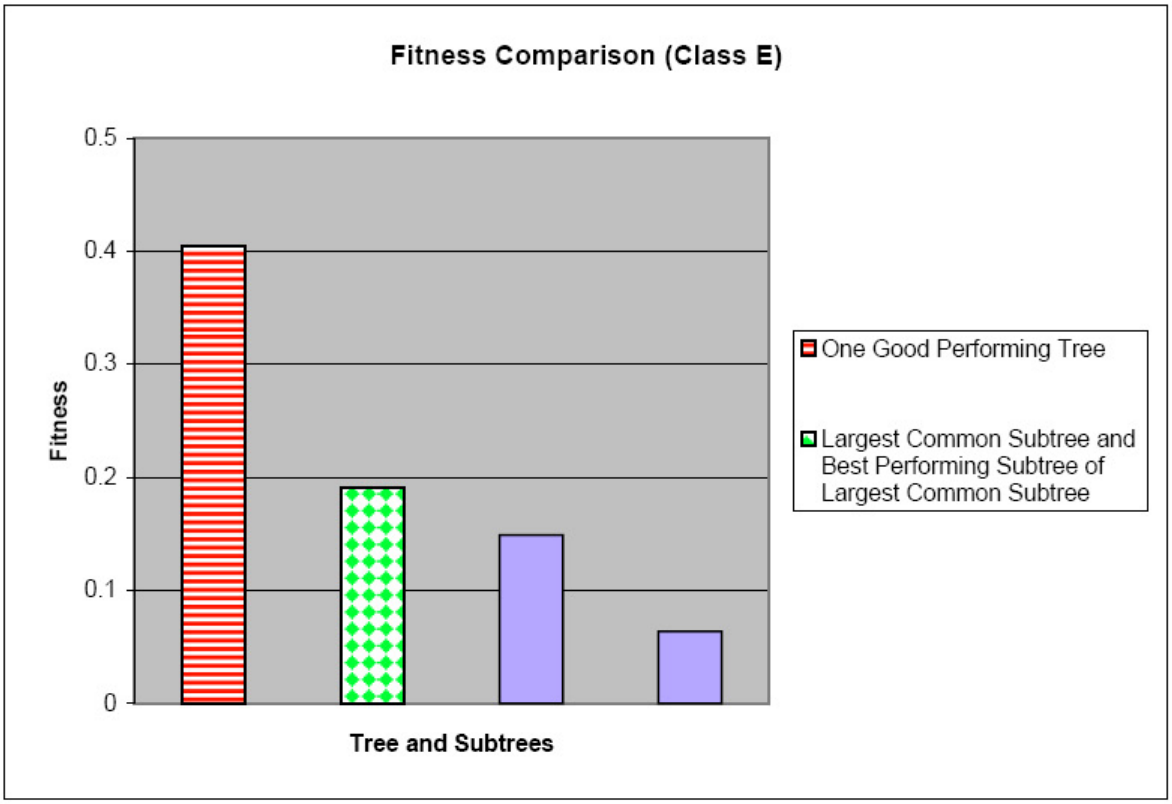
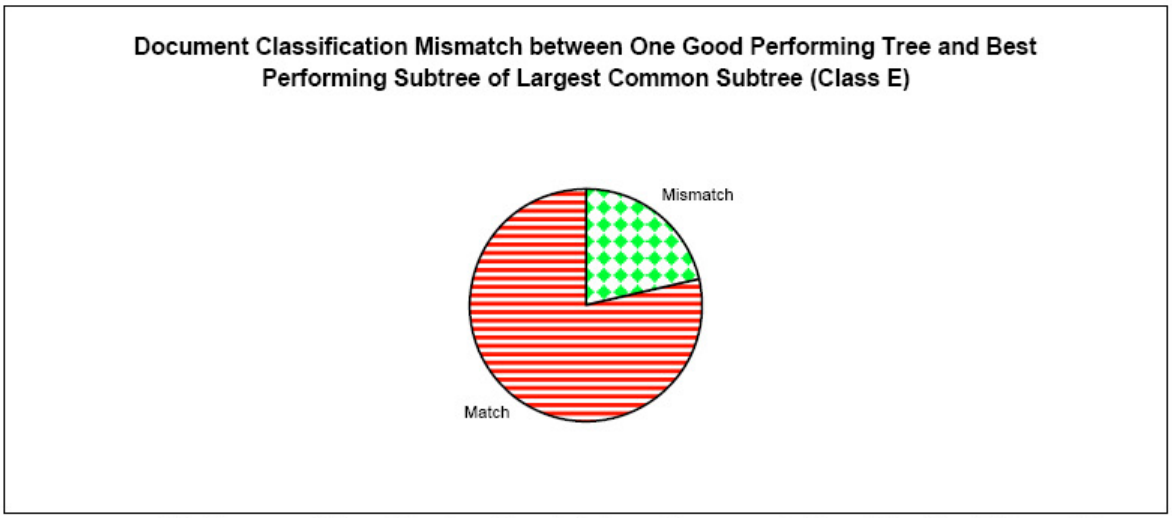


Figure 6.19: Largest Common Subtree for Class E's Good Performing Trees Population.



(a) Fitness Comparison between Tree and Subtrees



(b) Tree and Best Performing Subtree Document Classification Mismatch

Figure 6.20: Properties of the Largest Common Subtree for Class E's Good Performing Trees Population.

Figure 6.21 shows the largest common subtree for class F. As indicated by the polygon, the best performing subtree within this largest common subtree is (amsler + full_Okapi). For this class, the best two types of evidence are amsler and full_Okapi. So this subtree is adding the best two types of evidence together, thus reinforcing the documents relationship. As expected, this subtree's fitness is very close to the good performing tree as shown in Figure 6.22(a). However, the largest common subtree for this class does not have a good fitness. The best performing subtree seems to be a building block. And we also can see that this best performing subtree classifies class F's documents very similarly to the good performing tree, as shown in Figure 6.22(b).

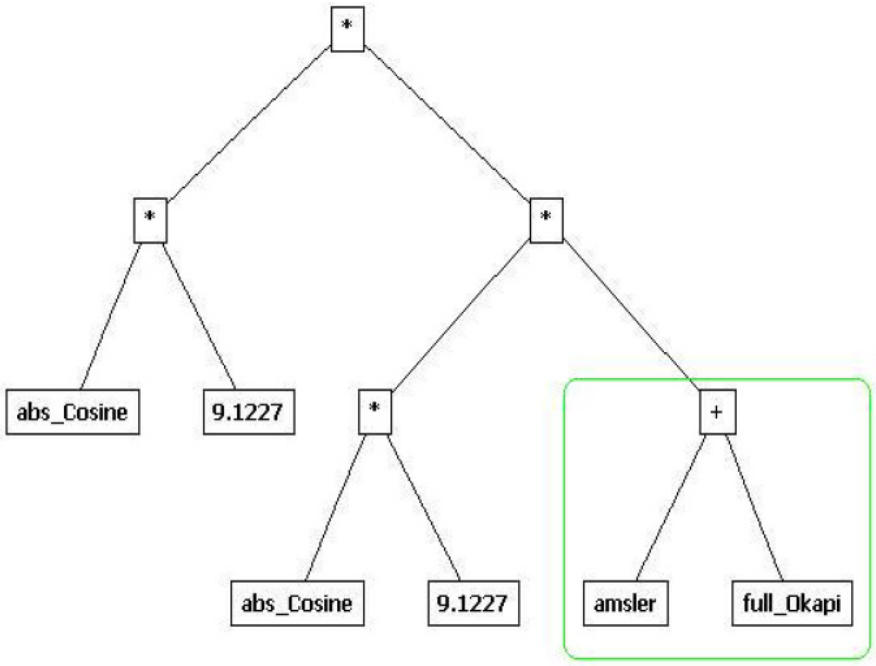
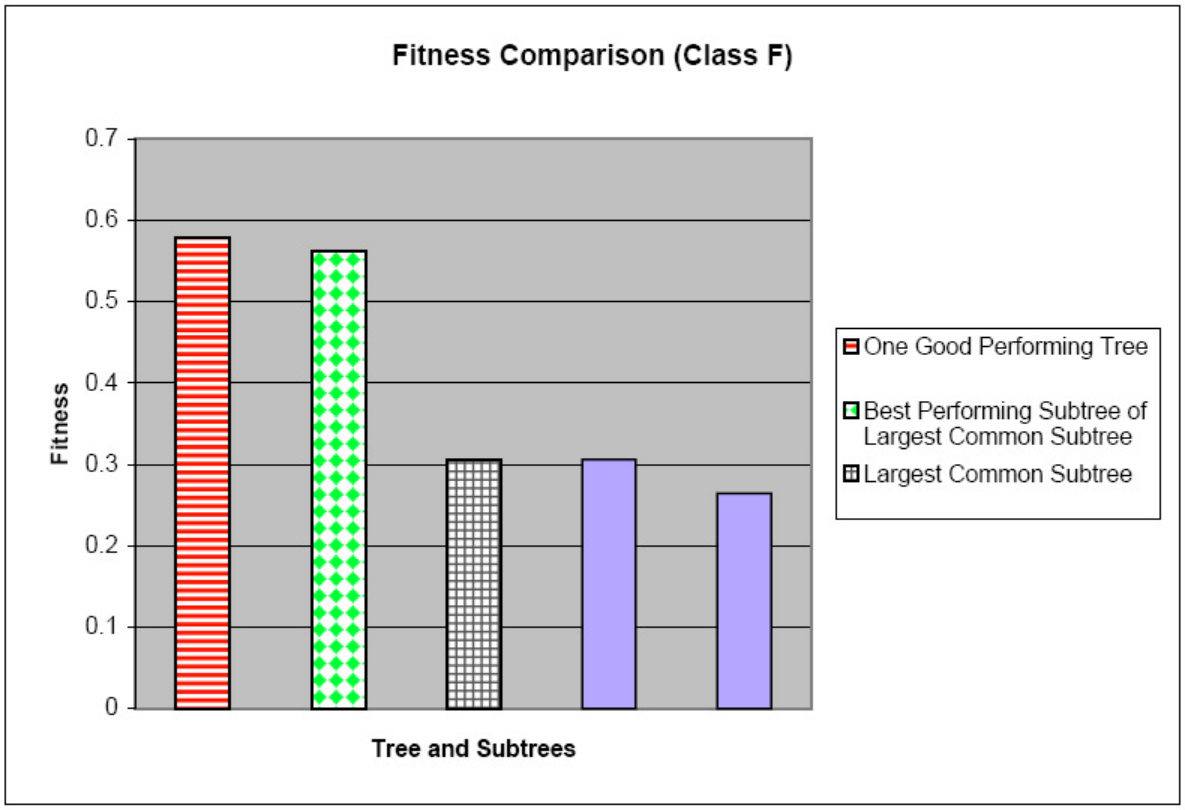
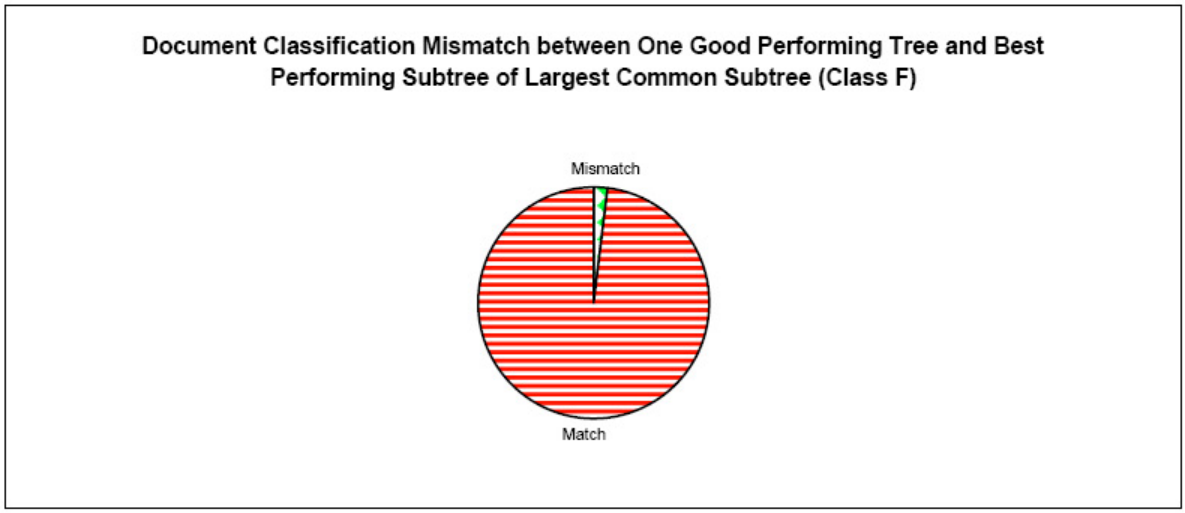


Figure 6.21: Largest Common Subtree for Class F's Good Performing Trees Population.



(a) Fitness Comparison between Tree and Subtrees



(b) Tree and Best Performing Subtree Document Classification Mismatch

Figure 6.22: Properties of the Largest Common Subtree for Class F's Good Performing Trees Population.

For class G, the largest common subtree is shown in Figure 6.23. This largest common subtree itself is the best performing subtree compared with its subtrees. As shown in Figure 6.24, this common subtree's performance is close to the good performing tree and it classifies class G documents consistently with the good performing tree for most documents.

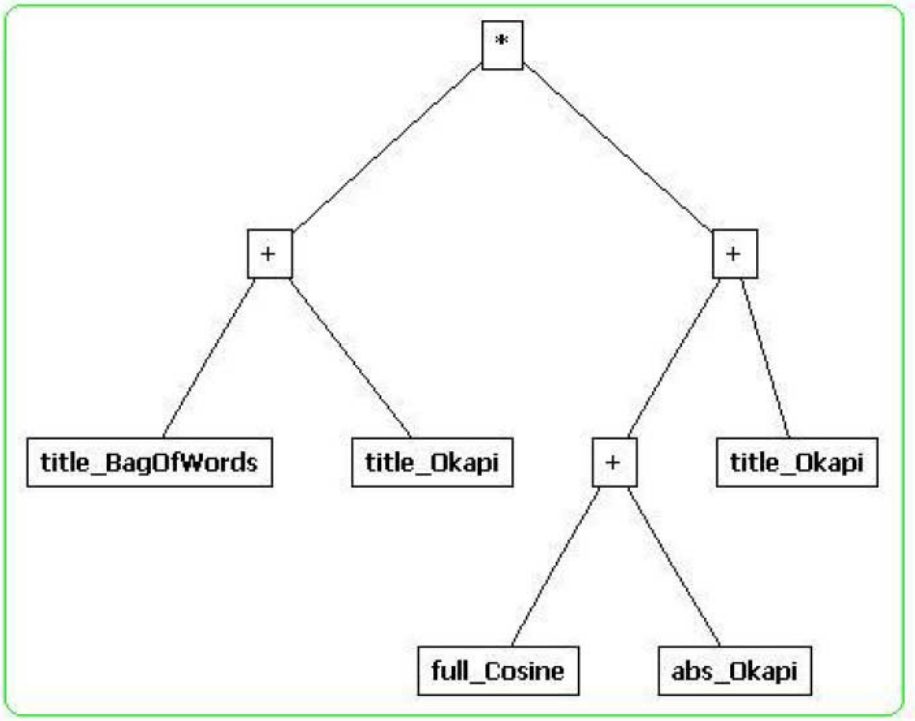
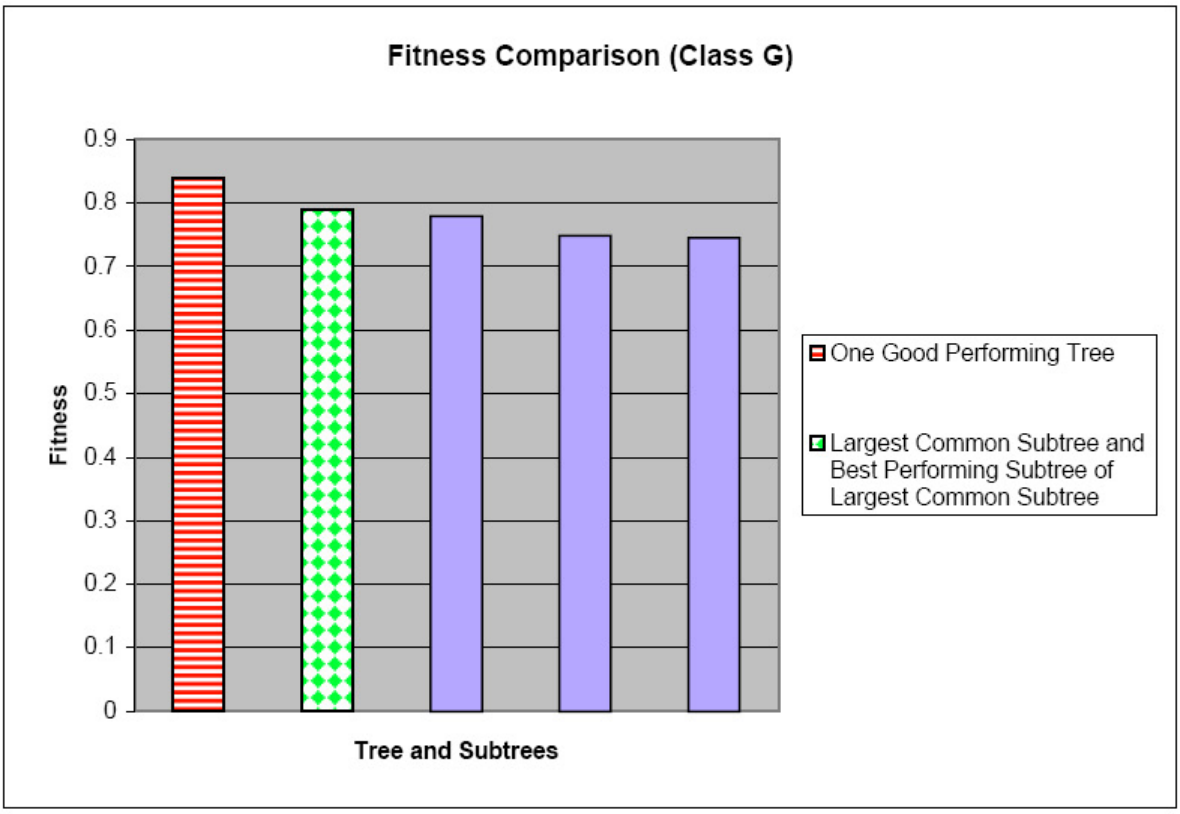
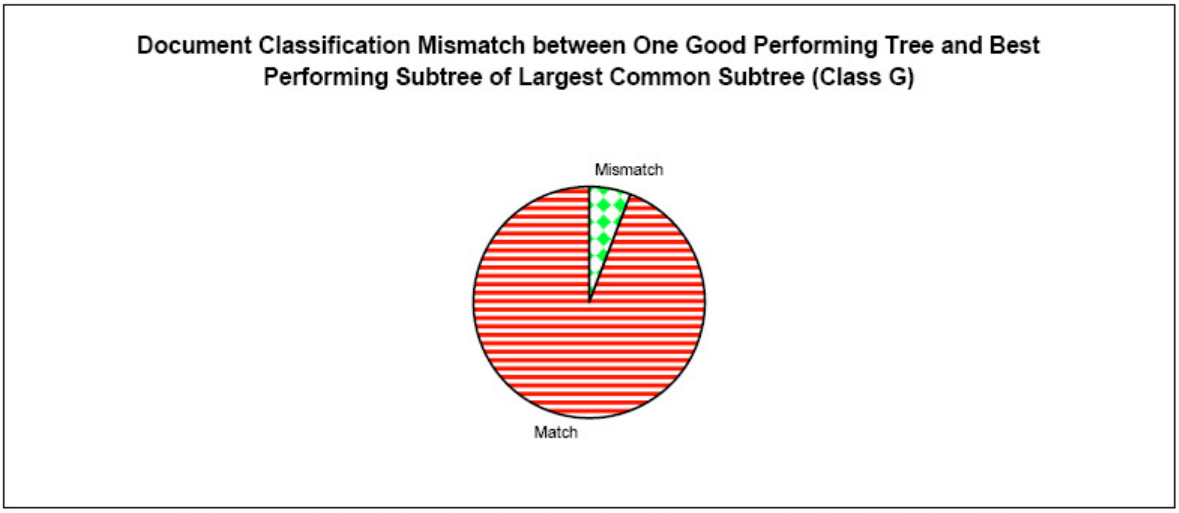


Figure 6.23: Largest Common Subtree for Class G's Good Performing Trees Population.



(a) Fitness Comparison between Tree and Subtrees



(b) Tree and Best Performing Subtree Document Classification Mismatch

Figure 6.24: Properties of the Largest Common Subtree for Class G's Good Performing Trees Population.

As shown in Figure 6.25, the best performing subtree for class H is within the largest common subtree. From Figure 6.26, we can see that the largest common subtree performs very badly. But the subtree within the largest common subtree performs much better and its performance is somewhat close to the good performing tree.

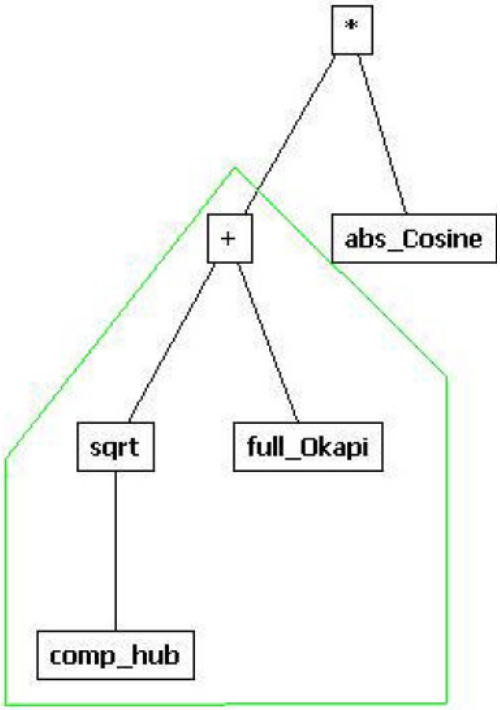
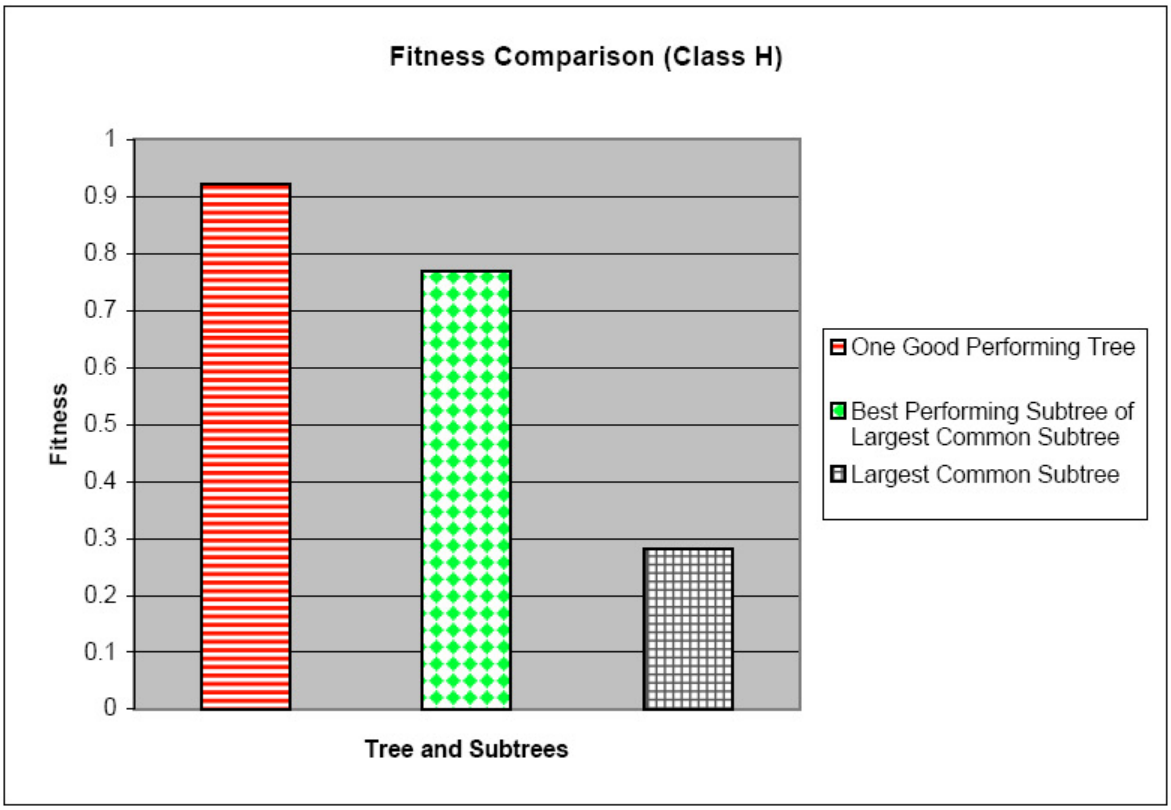
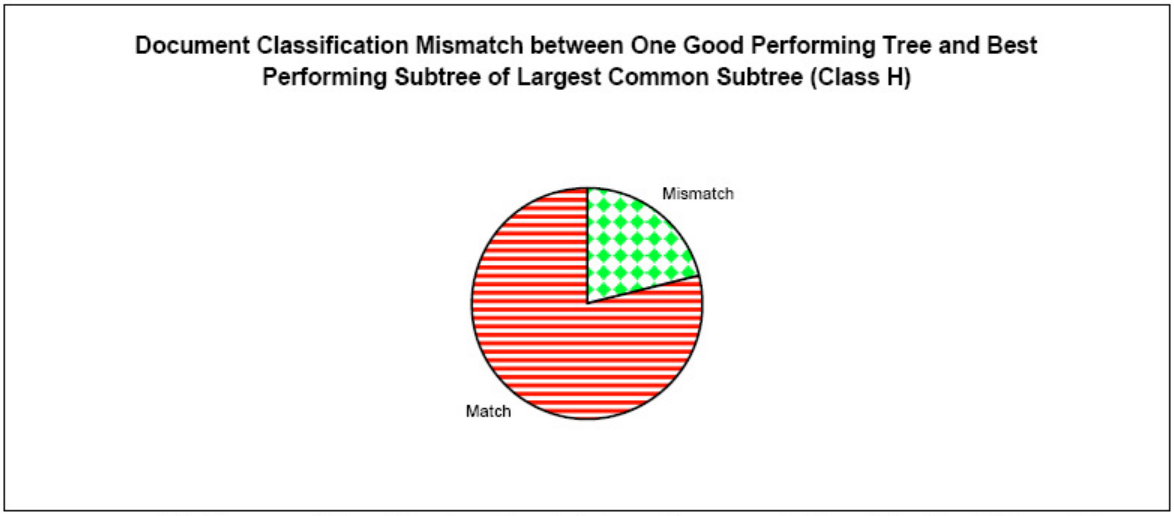


Figure 6.25: Largest Common Subtree for Class H's Good Performing Trees Population.



(a) Fitness Comparison between Tree and Subtrees



(b) Tree and Best Performing Subtree Document Classification Mismatch

Figure 6.26: Properties of the Largest Common Subtree for Class H's Good Performing Trees Population.

Class I's largest common subtree is shown in Figure 6.27. The best performing subtree for this largest common subtree is one of its subtrees. As we can see from Figure 6.28, this best performing subtree's performance is very close to that of the good performing tree. It also classifies class I's documents consistently when compared with the good performing tree.

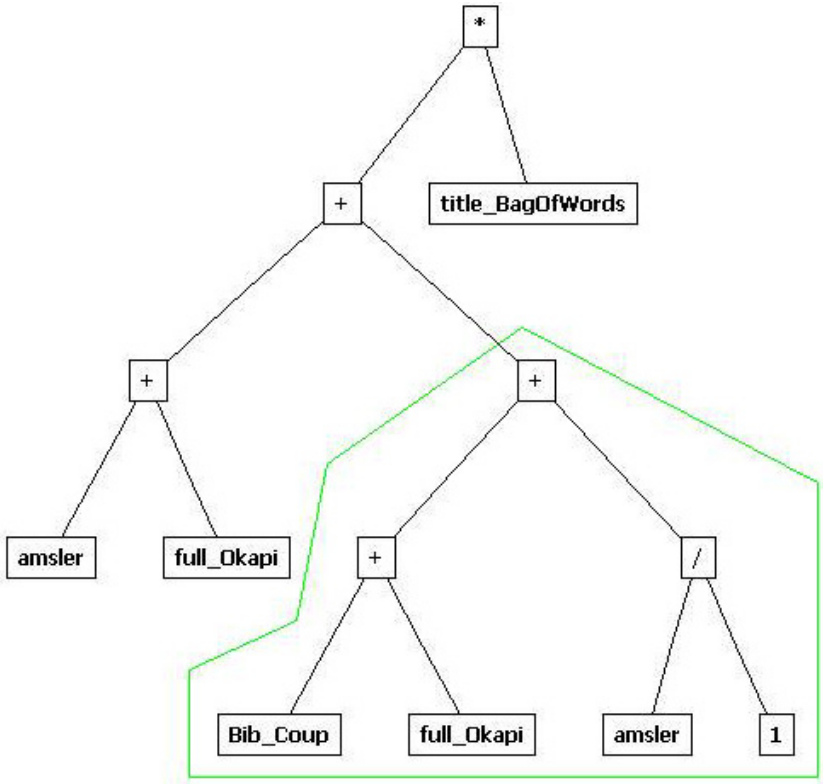
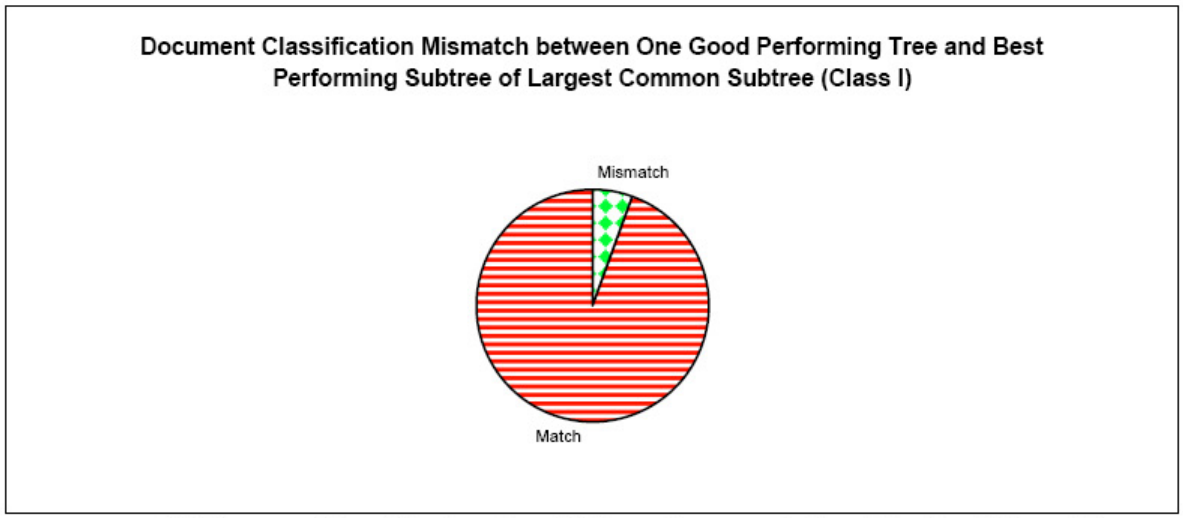


Figure 6.27: Largest Common Subtree for Class I's Good Performing Trees Population.

(a) Fitness Comparison between Tree and Subtrees



(b) Tree and Best Performing Subtree Document Classification Mismatch

Figure 6.28: Properties of the Largest Common Subtree for Class I's Good Performing Trees Population.

Figure 6.29 shows the largest common subtree for class J. For this class, the largest common subtree itself is the best performing subtree. We notice that this subtree also contains the block (amsler + full_Okapi). Compared with the good performing tree, the subtree's performance is relatively close as shown in Figure 6.30(a). And most of the time the subtree classification decision regarding class J's documents matches that of the good performing tree as can be seen from Figure 6.30(b).

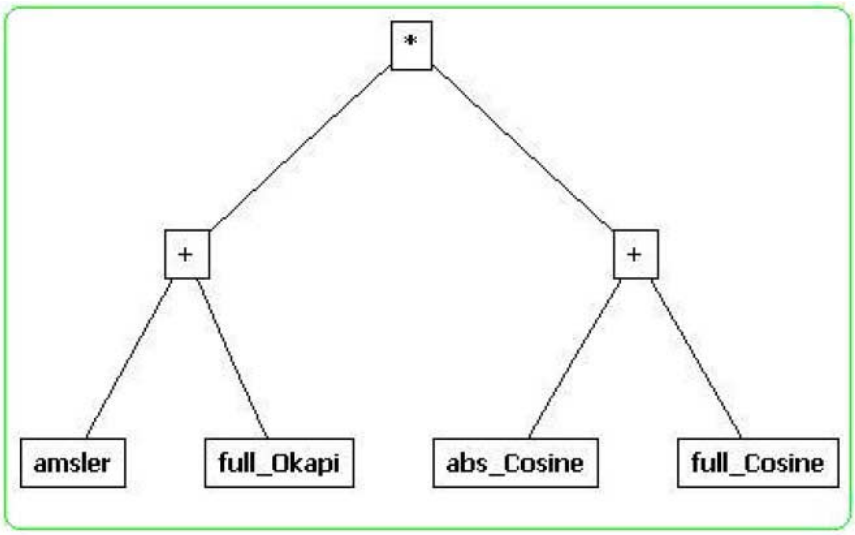
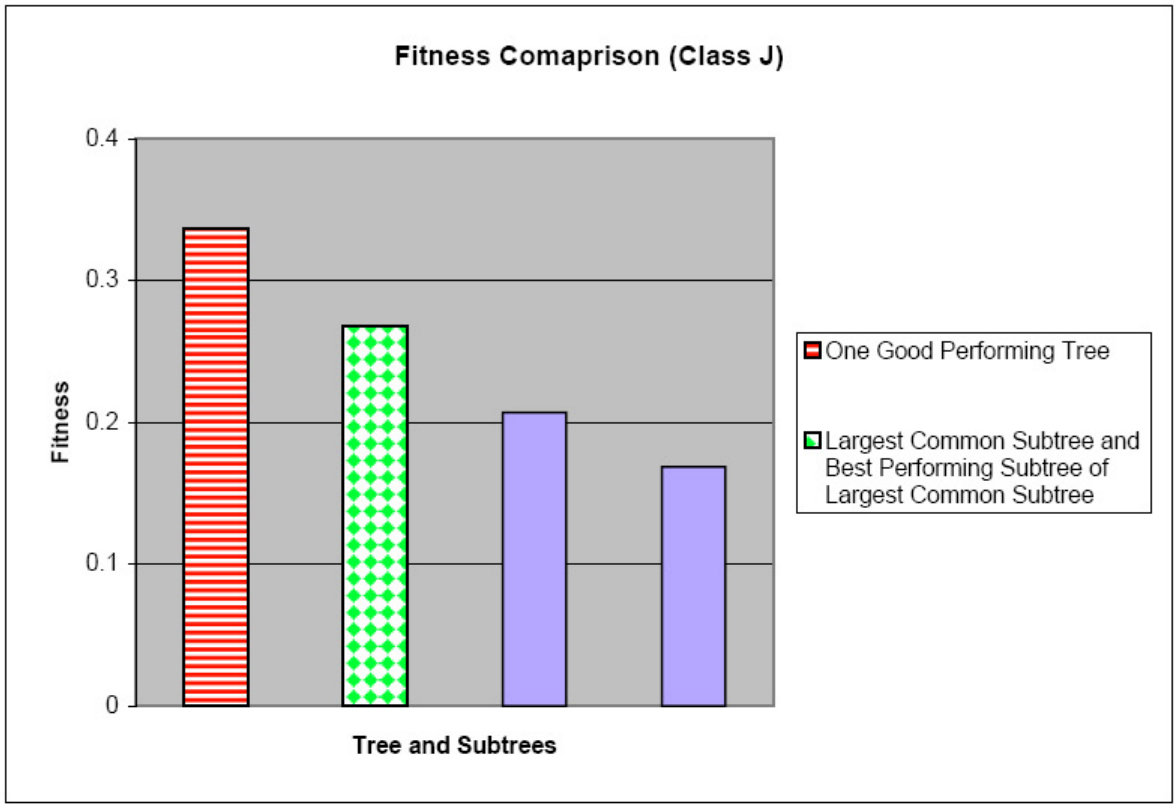
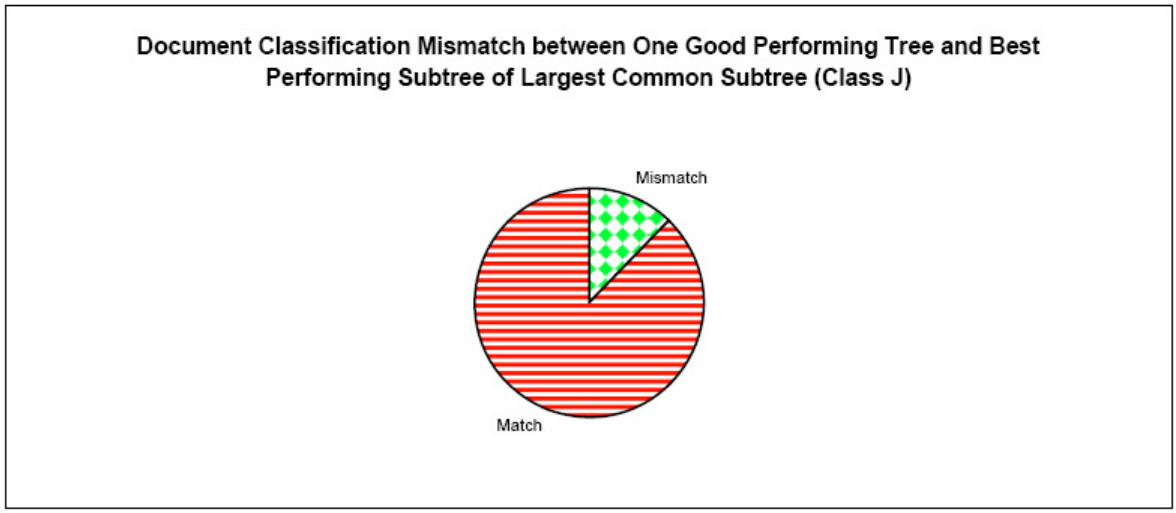


Figure 6.29: Largest Common Subtree for Class J's Good Performing Trees Population.



(a) Fitness Comparison between Tree and Subtrees



(b) Tree and Best Performing Subtree Document Classification Mismatch

Figure 6.30: Properties of the Largest Common Subtree for Class J's Good Performing Trees Population.

Similarly as class J, class K's largest common subtree is the best performing subtree as shown in Figure 6.31. For this subtree, we noticed that the right child of the root is "$\times 1$". GP discovered trees sometimes contain nodes like this. So it is possible to simplify or to prune the tree as discussed in the future work section of Chapter 7. As shown in Figure 6.32, the subtree's performance is relatively close to the good performing tree and its classification decision also matches that of the good performing tree most of the time.

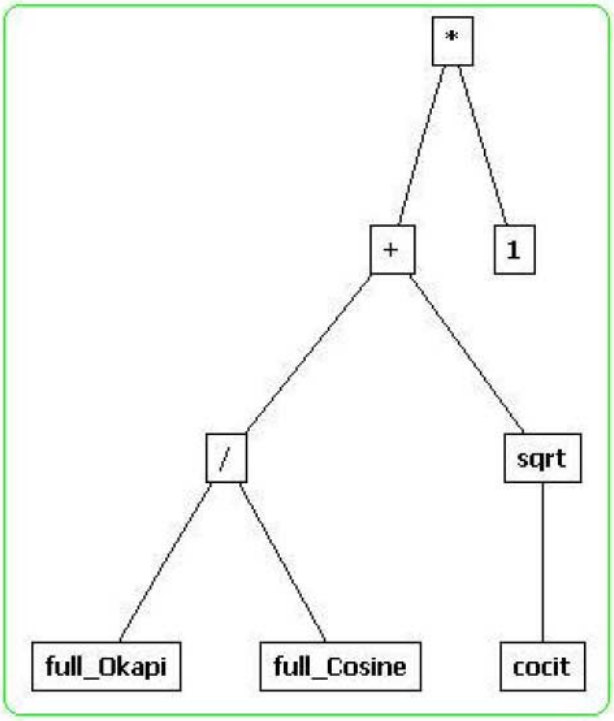
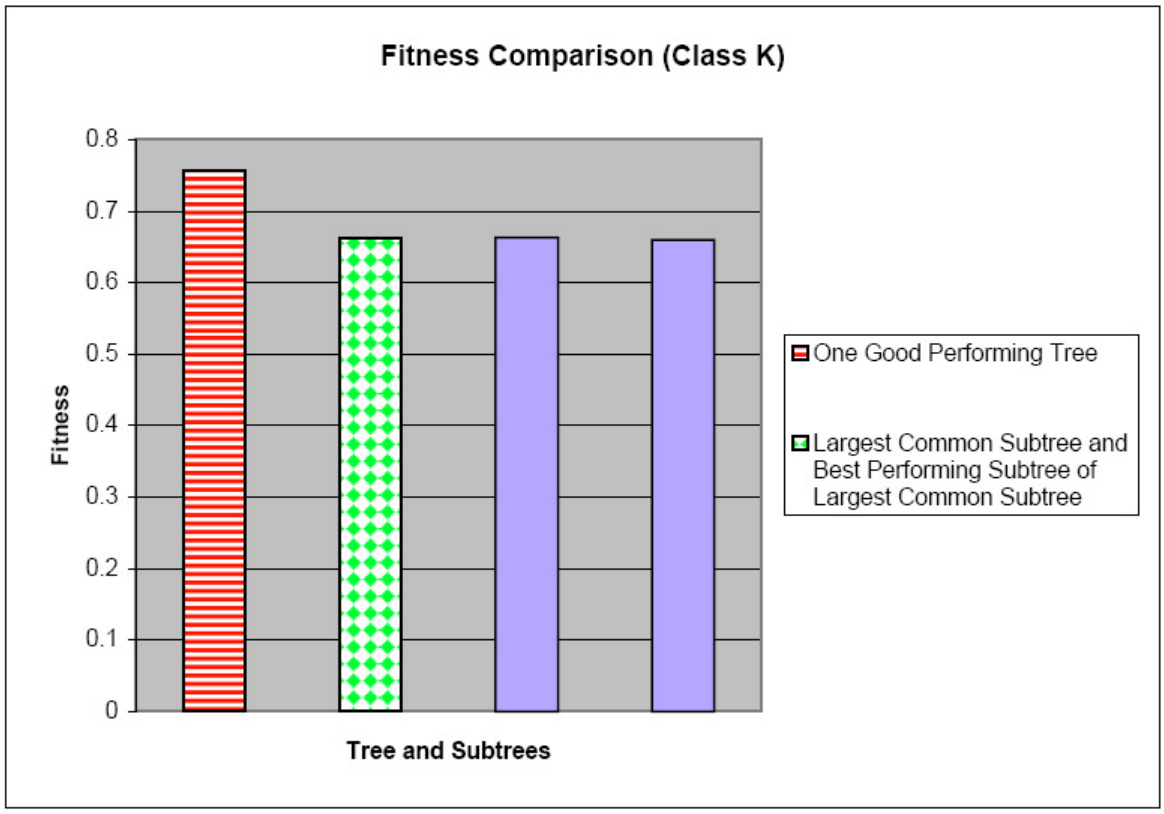
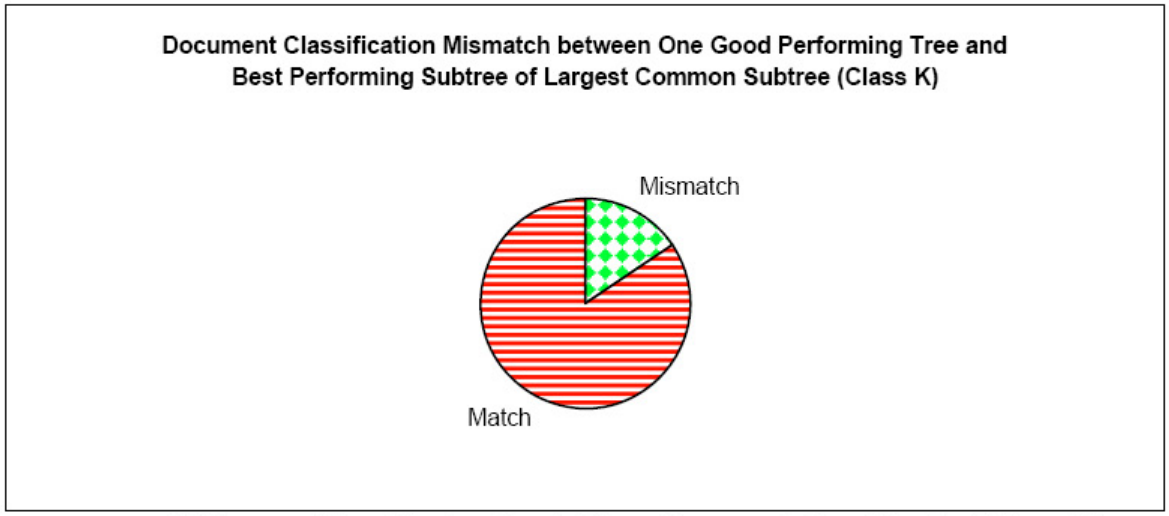


Figure 6.31: Largest Common Subtree for Class K's Good Performing Trees Population.



(a) Fitness Comparison between Tree and Subtrees



(b) Tree and Best Performing Subtree Document Classification Mismatch

Figure 6.32: Properties of the Largest Common Subtree for Class K's Good Performing Trees Population.

Largest Common Subtree for Population of Small Size Trees

We also run GP experiments restricting GP to evolve smaller size trees. We compared these smaller trees' performance with the results we obtained through our regular experimental settings and noticed that the performance of these smaller trees did not drop much. But these trees are smaller so it is easier for us to analyze them. For a population of smaller size good performing trees, we also ran the miner and extracted the largest common subtree and then the best performing subtree within the largest common subtree. Table 6.8 shows the mining results.

When the best performing subtree is compared with the good performing tree, between Table 6.8 columns 2 and 3, we notice that like in the comparison for larger trees (columns 5 and 6), the best performing subtree's fitness is very close or relatively close for most of the classes. If we look at the subtree comparison between smaller trees and larger trees, there also are some similarities at least for some classes. For example, for class B, the subtree for larger trees is contained within the subtree for smaller trees. For class C, both subtrees contain the block (+ full_Okapi full_BagOfWords). And for class E, the subtree for smaller trees is also a subtree of the larger trees' subtree. For class F, the two subtrees are exactly the same. And as pointed out earlier, this subtree seems to be a good building block. For class I, the best subtree for larger trees is contained within the subtree for smaller trees. And for class K, both subtrees contain the block (/ full_Okapi full_Cosine).

Largest Common Subtree for Five Runs' of Trees

We have 5 runs of data for each class. So we lump together good performing trees from these 5 runs for each class and analyze the largest common subtree. The motivation behind this analysis is to find building blocks for larger pool of trees. These building blocks might be important factors for classification. Table 6.9 shows the results.

First, when we compared columns 2 and 3, we can see that it's also the fact that the best performing subtree's fitness is very close or relatively close for most of the classes. When the subtrees shown in column 4 are compared with the subtrees we obtained using smaller trees and using one run's data, we also can notice some patterns. For class B, the block is the same as the one obtained from one run's data. This is also true for class F. For classes G, H, and I, the subtree is contained within their corresponding largest common subtree from one run's data. For class J, subtree (+ amsler full_Okapi) is also a frequent subtree among the population of good performing trees and this subtree is contained within the subtree obtained from one run's data. Finally, for class K, the block (/ full_Okapi full_Cosine) is a frequent subtree which also appears in both subtrees we

obtained using smaller trees and using one run's data.

Our various analyses comparing the performance of the largest common subtree to full trees and subtrees within the largest common subtree indicate that the largest common subtree represents the "essence" of what contributes to the good performance of the full tree. We also can see that for some classes, the good performance of the full tree was not explained by the largest common subtree itself, but rather by specific types of branches attached to it. One possible explanation is that the largest common subtree was used in a negative way, i.e., as the denominator of an expression where the numerator is another branch of the full tree. This is true for class B, but not for the others. So more extensive work needs to be done to explain this.

Table 6.8: Tree Mining Results for Smaller Size Trees.

Class	Smaller Trees			Larger Trees		
	Tree Fitness	Best Sub-tree Fitness	Best Subtree	Tree Fitness	Best Sub-tree Fitness	Best Sub-tree
A	69.58	67.04	(* (* (+ full_Cosine abs_BagOfWords) (* title_Cosine full_Cosine)) (sqrt title_Cosine))	69.30	68.17	Figure 6.11
B	78.84	78.84	(* (+ full_Okapi comp_hub) (+ Bib_Coup (sqrt full_Cosine)))	81.74	74.78	(+ full_Okapi comp_hub)
C	71.43	65.83	(* (+ abs_BagOfWords full_BagOfWords) (+ (+ full_Okapi full_BagOfWords) cocit))	71.43	66.67	Figure 6.15
D	96.44	92.62	(+ Cocit Bib_Coup)	96.53	94.79	Figure 6.17
E	38.30	14.89	(+ full_Cosine comp_hub)	40.43	19.15	Figure 6.19
F	56.62	56.16	(+ amsler full_Okapi)	57.99	56.16	(+ amsler full_Okapi)
G	82.89	79.85	(* full_Okapi full_Cosine)	84.03	79.09	Figure 6.23
H	91.30	77.01	(+ amsler full_Okapi)	92.21	76.88	(+ (sqrt comp_hub) full_Okapi)
I	76.17	70.05	(* (+ amsler (+ Bib_Coup full_Okapi)) (+ Bib_Coup full_Okapi))	71.66	69.08	(+ (+ Bib_Coup full_Okapi) (/ amsler 1))
J	31.80	26.82	(* (+ full_Okapi cocit) (+ full_Okapi comp_hub))	33.72	26.82	Figure 6.29
K	72.95	72.60	(* (+ amsler full_Okapi) (* full_BagOfWords (/ full_Okapi full_Cosine)))	75.57	66.32	Figure 6.31

Table 6.9: Tree Mining Results for Trees from Five Runs.

Class	Tree Fitness	Best Sub-tree Fitness	Best Subtree
A	69.30	61.41	(* (+ title_BagOfWords abs_BagOfWords) (sqrt full_Cosine))
B	81.74	74.78	(+ full_Okapi comp_hub)
C	71.43	66.67	(+ (/ full_Okapi full_Cosine) cocit)
D	96.53	94.79	(* Bib_Coup comp_aut)
E	40.43	19.15	(+ full_Cos amsler)
F	57.99	56.16	(+ amsler full_Okapi)
G	84.03	79.09	(+ title_BagOfWords title_Okapi)
H	92.21	76.88	(+ abs_Cosine full_Okapi)
I	71.66	69.08	(+ amsler full_Okapi)
J	33.72	26.82	(* full_Okapi full_Okapi)
K	75.57	66.32	(+ full_Okapi abs_Cosine)

Chapter 7

Conclusions and Future Work

In this chapter, we summarize the achievements of this work and point out some possible future research directions.

7.1 Conclusions

In this work, we considered the problem of classification in the context of document collections where textual content is scarce and imprecise citation information exists. A framework for tackling this problem based on Genetic Programming has been proposed and tested. Through the use of Genetic Programming, different sources of evidence like content-based and citation-based (or link-based) information were combined, and the effects of such combination were explored.

We studied the effectiveness of the proposed framework with two testbeds, the ACM Digital Library and Web data, in particular Cadê. Our experiments with these collections show that good classification is possible with documents which are noisy or which have small amounts of text (e.g., short metadata records) if multiple sources of evidence are fused in an intelligent way. Our experimental results have shown that improvement can be achieved relative to other machine learning approaches if genetic programming (GP) methods are combined with classifiers such as kNN.

Our experimental results on two different sets of documents from each level of the ACM Computing Classification System have demonstrated that the GP framework can be used to discover better similarity functions that, when applied to a kNN algorithm, can produce better classifiers than ones using individual evidence in isolation. Our experiments also showed that the framework

achieved results better than both traditional content-based and combination-based SVM classifiers. Comparison between the GP framework and linear fusion as well as GA also showed that GP has the ability to discover better similarity functions.

Experiments performed using the Cadê Web collection verified that the proposed framework is applicable to other collections. In this collection, links provide highly valuable information on the similarity of Web pages. Link-based types of evidence offer much better baselines than the content-based types of evidence. Due to this, GP was only able to show slight improvement over the best evidence in isolation. But the experiments also showed that GP yielded more improvement over classes with relatively worse baselines. Our experiments showed that the framework achieved results better than linear fusion. GA showed good results in this collection due to the very good link-based types of evidence. Our experiments also showed that the framework achieved much better results than traditional content-based SVM classifiers. This is as per expectation since the content-based types of evidence offer very weak baselines. Comparison between the GP framework and a Bayesian network model indicates that GP was able to show slightly better results.

For both collections, experiments were performed under each level of the corresponding classification category system. Experiments showed that the GP based framework works on both levels.

We also explored different sampling strategies to address the scalability issue, one common problem with GP. Experiments showed that active sampling was able to achieve comparable results compared with a full training set generated by random sampling when the size of the data pool is large. The results showed that active sampling could be more time efficient and at the same time offers good results when there are enough samples in the data pool to actively construct the training samples. Effects of different fitness functions also were explored during the experiments. The results showed that fitness functions play an important role for GP to discover good solutions.

In this research, we also proposed, designed, and implemented different approaches to analyze the trees with the aim of tackling the problem of the difficulty to interpret previously discovered GP trees for text classification. Through tree distance calculation, we showed that high performing trees for different classes are largely different with each other, while high performing trees for the same class are much closer to each other. We have provided statistical support for this experimental result by conducting multi-sample analysis. The results of the analysis proved that distinct trees are produced for distinct classes. We have derived a central tree, which is a generalized high performance tree characterizing the dominant features of other good performing trees. We proposed and implemented a tree miner to analyze the common subtrees of the population of good performing

trees. The mining results showed that some important factors are related to, and may predict the good performance of trees, at least for some classes.

7.2 Future Work

In this section, we give a list of suggestions for the continuation of the work. The list addresses open questions left by the research, and new ideas found during the course of the work.

Parallelization

The effectiveness of evolutionary computation techniques comes at a high computational cost, especially when evaluation of fitness requires a long processing time, as for classification purposes. But genetic programming is highly amenable to parallelization [4]. There are currently two major approaches to parallelization of genetic programming, namely the asynchronous island approach and the master-slave approach [44]. In the island approach, semi-isolated subpopulations (or demes) are evolved in separate processors and, upon completion of a generation, selected individuals migrate from each processor to neighbors, asynchronously. In the master-slave approach, parallelization is done in the level of individuals. One processor (the master) stores the whole population and applies evolutionary operators (e.g., selection, crossover, mutation) but the fitness function for each individual is calculated in a distributed fashion in the slave-nodes. Finally, hierarchical hybrids combine both techniques to exploit the advantages of each and avoid their respective limitations. To solve issues of scalability, massive parallelization can be applied to our genetic programming approach to classification. The above techniques can be applied and evaluated, and perhaps will lead to new ones, with the Virginia Tech supercomputer, System X [117]. In terms of speed, our supercomputer can theoretically handle 17 teraflops, or 17 trillion operations per second; it is the fastest academic supercomputer in the world, and is available in flexible configurations and at a deeply discounted rate.

Multiclass Classification

In this research, we experimented with documents that only belong to one category. But it is very common that documents cover topics from more than one category. This is known as multiclass classification. Multiclass classification is a central problem in machine learning, as applications that require discrimination among several classes are ubiquitous. So our framework can be extended to remove the constraint that documents only can come from one category. The returning

scores, comparable across categories as a result of the kNN classifier, can be used to assign multiple categories to a document.

Hierarchical Classification

For each test bed collection, we ran experiments on two levels of the corresponding classification system in order to validate our proposed approach. We treated each category or class separately, thus in effect "flattening" the class structure. In other words, the classification model distinguishes a second-level category from all other second level categories. But the hierarchical structure can be utilized to decompose the classification problem into a set of smaller problems corresponding to hierarchical splits in the category system tree. The idea is to learn to distinguish among classes at the top level, then lower level distinctions are learned only within the appropriate top level of the tree. Each of these sub-problems can be solved much more efficiently, and hopefully more accurately as well [31].

Classification Update Problem

For text classification, it is possible that documents for a new category appear after some amount of time, like documents for a new research field. So it's a common problem to update the classification model periodically. Since training takes an enormous amount of time, when a new category appears, we can analyze this new category and in the current category system, discover the category which is closest to this new one. Then we use that close category's trees as the input to the training process to discover new trees for this new category. This would be more time efficient.

Improve Evidence

Analysis will help us identify which types of evidence are inaccurate and incomplete, the two major problems mentioned in the first chapter. Then we can correct these problems to get even better classification results. For example, we could define a way to clean up the citation text by removing non-useful/noisy information and then explore partial matching algorithms to correct the OCR errors. Experimentation can allow us to validate these algorithms, and will allow us to select the best among all we devise. This can help us to get more accurate citation relationships between documents. Techniques could be applied to fill in missing information, e.g., by looking at other available information to derive and reinforce possible relationships between documents. Documents also can be weighted to boost highly cited documents, so that GP will learn the classification more effectively and efficiently.

Classification Framework Refinement

We adopted a simple majority voting in the framework. Other ways of doing majority voting could be explored, e.g., taking into account each classifier's confidence. Another reasonable alternative to the majority voting in the framework would be to generate only one "global" similarity function instead of n "local" ones. However, discovery of such a globally unique similarity function, besides the potential of suffering overfitting, was too demanding in terms of training time and necessary computational resources while the applied "per class" training allowed easy distribution of the training task. The parallel strategies mentioned above would allow "global vs. local" experiments.

Fitness functions for GP can be defined very flexibly. We explored three different fitness functions in this research. Other possible fitness functions can be explored to guide GP's evolution.

More Extensive Analysis

The trees can be selectively pruned [9, 22, 77] such that their performance does not degrade significantly. Doing this, we can selectively remove certain parts of the trees that do not have much influence on the performance of the tree. The trees after pruning are smaller and thus easier to analyze.

The tree miner can be applied to not only good performing trees, but also bad performing ones. This way we can look for subtrees that explain the bad performance and discriminatively analyze the good tree's population and bad tree's population to see whether different blocks can be identified, and then possible identify how they relate to the performance. This could be done for the trees throughout the course of an evolving run. The result of this can be used to determine factors that bias a run towards success or failure.

Regarding multi-sample testing, we used edit distance as the distance metric. However, edit distance does not satisfy all of Feigin and Alvo's requirements of a distance metric. Specifically, edit distance is not concave on the space of probability measures and this suggests that other distance metrics might be more suitable for this type of analysis.

We discussed two methods to develop central trees in Chapter 6. Another method to derive a central tree could be through constructing intermediate trees. An *intermediate tree* is a tree which forms an intermediate point on the transformation path generated when one tree is transformed to another. Apparently, we will get a set of intermediate trees when transforming one tree to another. A mean point for these intermediate trees along the transformation path could be identified and this mean point should be the one equidistant from source tree and destination tree. Then these mean points can be used to form a mean tree for the population. One of the possible approaches to form a mean tree is to sample the population and get 3 trees as vertices of a triangle joined by edges.

The edges represent transformation paths and their length is equal to the edit distance between trees at their ends. The mean tree for these 3 trees will be the centroid of the triangle. One possible approach to identify the centroid is as follows: first find the three mean points on the three edges of the triangle and join them to form a new triangle; then consider this new triangle as the new input and iteratively create mean points till a triangle, whose edges are of unit length, is obtained. Any one of the vertices of this triangle, which corresponds to a tree, can now form the centroid of the initial triangle defined.

Bibliography

[1] ACM. Association for Computing Machinery, http://www.acm.org, 2006.

[2] ACM_CCS_Update_Committee. The ACM Computing Classification System [1998 version], http://www.acm.org/class/1998/, 2006.

[3] Robert Amsler. Application of citation-based automatic classification. Technical Report 72-14, The University of Texas at Austin, Linguistics Research Center, Austin, TX, December 1972.

[4] David Andre and John R. Koza. A parallel implementation of genetic programming that achieves super-linear performance. In Hamid R. Arabnia, editor, *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications*, volume III, pages 1163–1174, Sunnyvale, 9-11 August 1996. CSREA.

[5] Tatsuya Asai, Kenji Abe, Shinji Kawasoe, Hiroki Arimura, Hiroshi Satamoto, and Setsuo Arikawa. Efficient substructure discovery from large semi-structured data. In *Second SIAM International Conference on Data Mining (SDM)*, pages 158–174, Arlington, VA, USA, April 2002.

[6] Wolfgang Banzhaf, Peter Nordin, Robert E. Keller, and Frank D. Francone. *Genetic Programming – An Introduction; On the Automatic Evolution of Computer Programs and its Applications*. Morgan Kaufmann, dpunkt.verlag, January 1998.

[7] David C. Blair and M. E. Maron. An evaluation of retrieval effectiveness for a full-text document-retrieval system. *Communications of the ACM (CACM)*, 28(3):289–299, 1985.

[8] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual Web search engine. In *WWW7: Proceedings of the 7th International World Wide Web Conference*, pages

107–117, Brisbane, Australia, April 1998. Elsevier Science Publishers B. V., Amsterdam, The Netherlands.

[9] Wray Buntine and Tim Niblett. A further comparison of splitting rules for decision-tree induction. *Mach. Learn.*, 8(1):75–85, 1992.

[10] Cadê. A directory for websites of Brazil, http://www.cade.com.br, 2006.

[11] Pável Calado, Marco Cristo, Edleno Silva de Moura, Nivio Ziviani, Berthier A. Ribeiro-Neto, and Marcos André Gonçalves. Combining link-based and content-based methods for Web document classification. In *Proceedings of CIKM-03, 12th ACM International Conference on Information and Knowledge Management*, pages 394–401, New Orleans, USA, November 2003. ACM Press, New York, USA.

[12] Pável Calado, Berthier Ribeiro-Neto, Nivio Ziviani, Edleno Moura, and Ilmério Silva. Local versus global link information in the Web. *ACM Trans. Inf. Syst.*, 21(1):42–63, 2003.

[13] Pável P. Calado, Marcos A. Gonçalves, Edward A. Fox, Berthier Ribeiro-Neto, Alberto H. F. Laender, Altigran S. da Silva, Davi C. Reis, Pablo A. Roberto, Monique V. Vieira, and Juliano P. Lage. The Web-DL environment for building digital libraries from the Web. In *JCDL '03: Proceedings of the 3rd ACM/IEEE-CS Joint Conference on Digital Libraries*, pages 346–357, Houston, Texas, 2003. IEEE Computer Society, Washington, DC, USA.

[14] Soumen Chakrabarti, Byron Dom, and Piotr Indyk. Enhanced hypertext categorization using hyperlinks. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 307–318, Seattle, Washington, June 1998.

[15] Sin Man Cheang, Kin Hong Lee, and Kwong Sak Leung. Data classification using genetic parallel programming. In E. Cantú-Paz, J. A. Foster, K. Deb, D. Davis, R. Roy, U.-M. O'Reilly, H.-G. Beyer, R. Standish, G. Kendall, S. Wilson, M. Harman, J. Wegener, D. Dasgupta, M. A. Potter, A. C. Schultz, K. Dowsland, N. Jonoska, and J. Miller, editors, *Genetic and Evolutionary Computation – GECCO-2003*, volume 2724 of *LNCS*, pages 1918–1919, Chicago, 2003. Springer-Verlag.

[16] Hsinchun Chen, Yi-Ming Chung, Marshall Ramsey, and Christopher C. Yang. A smart itsy bitsy spider for the web. *J. Am. Soc. Inf. Sci.*, 49(7):604–618, 1998.

[17] Hsinchun Chen, Ganesan Shankaranarayanan, Linlin She, and Anand Iyer. A machine learning approach to inductive query by examples: an experiment using relevance feedback,

ID3, genetic algorithms, and simulated annealing. *J. Am. Soc. Inf. Sci.*, 49(8):639–705, 1998.

[18] Yun Chi, Siegfried Nijssen, R. R. Muntz, and Joost N. Kok. Frequent subtree mining–an overview. In *Fundamenta Informaticae*, volume 66, pages 161–198. IOS Press, 2005.

[19] Weon Sam Chung and Rafael A. Perez. The schema theorem considered insufficient. In *ICTAI 94: Proceedings of the Sixth IEEE International Conference on Tools with Artificial Intelligence*, pages 748–751, New Orleans, USA, 1994.

[20] CITIDEL. Computing and Information Technology Interactive Digital Educational Library, www.citidel.org, 2006.

[21] Chris Clack, Johnny Farringdon, Peter Lidwell, and Tina Yu. Autonomous document classification for business. In *AGENTS '97: Proceedings of the first international conference on autonomous agents*, pages 201–208, Marina del Rey, California, United States, 1997. ACM Press, New York, USA.

[22] Peter Clark and Tim Niblett. The CN2 induction algorithm. *Mach. Learn.*, 3(4):261–283, 1989.

[23] David Cohn and Thomas Hofmann. The missing link - a probabilistic model of document content and hypertext connectivity. In Todd K. Leen, Thomas G. Dietterich, and Volker Tresp, editors, *Advances in Neural Information Processing Systems 13*, pages 430–436. MIT Press, 2001.

[24] James H. Coombs. Hypertext, full text, and automatic linking. In *Proceedings of the 13th International Conference on Research and Development in Information Retrieval*, pages 83–98, 1990.

[25] W. Bruce Croft, T. J. Lucia, Janey Cringean, and Peter Willet. Retrieving documents by plausible inference: An experimental study. *Information Processing & Management*, 25(6):111–134, 1989.

[26] Stephen D'Alessio, Keitha Murray, Robert Schiaffino, and Aaron Kershenbaum. Category levels in hierarchical text categorization. In *Proceedings of EMNLP-98, 3rd Conference on Empirical Methods in Natural Language Processing*, Granada, ES, 1998. Association for Computational Linguistics, Morristown, US.

[27] Jeffrey Dean and Monika Rauch Henzinger. Finding related pages in the World Wide Web. *Computer Networks*, 31(11–16):1467–1479, May 1999. Also in Proceedings of the 8th International World Wide Web Conference.

[28] M. Dolores del Castillo and José Ignacio Serrano. A multistrategy approach for digital text categorization from imbalanced documents. *SIGKDD Explor. Newsl.*, 6(1):70–79, 2004.

[29] DMOZ. Open Directory Project, http://dmoz.org, 2006.

[30] Richard Duda, Peter Hart, and David Stork. *Pattern Classification*. John Wiley and Sons, 2001.

[31] Susan Dumais and Hao Chen. Hierarchical classification of Web content. In *SIGIR '00: Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 256–263, Athens, Greece, 2000. ACM Press, New York, USA.

[32] Jeroen Eggermont, Joost N. Kok, and Walter A. Kosters. Genetic programming for data classification: Refining the search space. In T. Heskes, P. Lucas, L. Vuurpijl, and W. Wiegerinck, editors, *Proceedings of the Fifteenth Belgium/Netherlands Conference on Artificial Intelligence (BNAIC'03)*, pages 123–130, Nijmegen, The Netherlands, 23-24 October 2003.

[33] Weiguo Fan, Edward A. Fox, Praveen Pathak, and Harris Wu. The effects of fitness functions on genetic programming-based ranking discovery for web search. *Journal of the American Society for Information Science and Technology*, 55(7):628–636, 2004.

[34] Weiguo Fan, Michael D. Gordon, and Praveen Pathak. Personalization of search engine services for effective retrieval and knowledge management. In *The Proceedings of the International Conference on Information Systems 2000*, pages 20–34, Brisbane, Australia, Dec. 10–13, 2000.

[35] Weiguo Fan, Michael D. Gordon, and Praveen Pathak. Discovery of context-specific ranking functions for effective information retrieval using genetic programming. *IEEE Transactions on Knowledge and Data Engineering*, 16(4):523–527, 2004.

[36] Weiguo Fan, Michael D. Gordon, and Praveen Pathak. A generic ranking function discovery framework by genetic programming for information retrieval. *Information Processing and Management*, 40(4):587–602, 2004.

[37] Weiguo Fan, Michael D. Gordon, Praveen Pathak, Wensi Xi, and Edward A. Fox. Ranking function optimization for effective web search by genetic programming: An empirical study. In *Proceedings of 37th Hawaii International Conference on System Sciences*, pages 105–112, Hawaii, 2004. IEEE.

[38] Weiguo Fan, Ming Luo, Li Wang, Wensi Xi, and Edward A. Fox. Tuning before feedback: combining ranking function discovery and blind feedback for robust retrieval. In *Proceedings of the 27th Annual International ACM SIGIR Conference*, pages 138–145, Sheffield, U.K., 2004. ACM Press, New York, USA.

[39] Paul D. Feigin and Mayer Alvo. Intergroup diversity and concordance for ranking data: An approach via metrics for permutations. *The Annals of Statistics*, 14:691–707, 1986.

[40] Michelle Fisher and Richard Everson. When are links useful? experiments in text classification. In F. Sebastianini, editor, *Proceedings of the 25th Annual European Conference on Information Retrieval Research, ECIR 2003*, pages 41–56, Pisa, Italy, 2003. Springer-Verlag, Berlin, Heidelberg, Germany.

[41] David B. Fogel and Adam Ghozeil. The schema theorem and the misallocation of trials in the presence of stochastic effects. In *EP '98: Proceedings of the 7th International Conference on Evolutionary Programming VII*, pages 313–321, London, UK, 1998. Springer-Verlag.

[42] Mark E. Frisse and Steve B. Cousins. Information retrieval from hypertext: update on the dynamic medical handbook project. In *HYPERTEXT '89: Proceedings of the second annual ACM conference on hypertext*, pages 199–212, Pittsburgh, Pennsylvania, United States, 1989. ACM Press, New York, USA.

[43] Johannes Furnkranz. Exploiting structural information for text classification on the WWW. In *IDA '99: Proceedings of the Third International Symposium on Advances in Intelligent Data Analysis*, pages 487–498, London, UK, 1999. Springer-Verlag.

[44] Christian Gagné, Marc Parizeau, and Marc Dubreuil. The master-slave architecture for evolutionary computations revisited. In E. Cantú-Paz, J. A. Foster, K. Deb, D. Davis, R. Roy, U.-M. O'Reilly, H.-G. Beyer, R. Standish, G. Kendall, S. Wilson, M. Harman, J. Wegener, D. Dasgupta, M. A. Potter, A. C. Schultz, K. Dowsland, N. Jonoska, and J. Miller, editors, *Genetic and Evolutionary Computation – GECCO-2003*, volume 2724 of *LNCS*, pages 1578–1579, Chicago, 12-16 July 2003. Springer-Verlag.

[45] Eric J. Glover, Kostas Tsioutsiouliklis, Steve Lawrence, David M. Pennock, and Gary W. Flake. Using Web structure for classifying and describing Web pages. In *Proceedings of WWW2002, Eleventh International World Wide Web Conference*, Honolulu, Hawaii, 7-11 May 2002.

[46] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1989.

[47] Marcos André Gonçalves, Robert K. France, and Edward A. Fox. Marian: Flexible interoperability for federated digital libraries. In *ECDL '01: Proceedings of the 5th European Conference on Research and Advanced Technology for Digital Libraries*, pages 173–186, Darmstadt, Germany, 2001. Springer-Verlag.

[48] Michael Gordon. Probabilistic and genetic algorithms for document retrieval. *Communications of the ACM*, 31(10):1208–1218, October 1988.

[49] Michael D. Gordon. User-based document clustering by redescribing subject descriptions with a genetic algorithm. *Journal of the American Society for Information Science*, 42(5):311–322, June 1991.

[50] Norbert Gövert, Mounia Lalmas, and Norbert Fuhr. A probabilistic description-oriented approach for categorizing web documents. In *Proceedings of the 8th International Conference on Information and Knowledge Management CIKM 99*, pages 475–482, Kansas City, Missouri, USA, November 1999.

[51] Marko Grobelnik. Feature selection for classification based on text hierarchy. In *Proceedings of the Workshop on Learning from Text and the Web, Conference on Automated Learning and Discovery (CONALD)*, Carnegie Mellon University, Pittsburgh, PA, USA, June 1998.

[52] Dan Gusfield. *Algorithms on strings, trees, and sequences: computer science and computational biology*. Cambridge University Press, New York, NY, USA, 1997.

[53] John H. Holland. *Adaption in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, MI, USA, 1975.

[54] David Hull. Using statistical testing in the evaluation of retrieval experiments. In *SIGIR '93: Proceedings of the 16th annual international ACM SIGIR conference on research and de-*

velopment in information retrieval, pages 329–338, Pittsburgh, Pennsylvania, United States, 1993. ACM Press, New York, USA.

[55] E. Ide. New experiments in relevance feedback. In G. Salton, editor, *The SMART retrieval system - Experiments in automatic document processing*, pages 337–354, Englewood Cliffs, NJ, 1971. Prentice Hall Inc.

[56] Thorsten Joachims. Text categorization with support vector machines: learning with many relevant features. In *Proceedings of ECML-98, 10th European Conference on Machine Learning*, pages 137–142, Chemnitz, Germany, April 1998.

[57] Thorsten Joachims, Nello Cristianini, and John Shawe-Taylor. Composite kernels for hypertext categorisation. In Carla Brodley and Andrea Danyluk, editors, *Proceedings of ICML-01, 18th International Conference on Machine Learning*, pages 250–257, Williams College, US, 2001. Morgan Kaufmann Publishers, San Francisco, US.

[58] Nithiwat Kampanya, Rao Shen, Seonho Kim, Chris North, and Edward A. Fox. CitiViz: A visual user interface to the CITIDEL system. In *Proc. of ECDL-04*, pages 122–133, Bath, UK, 2004.

[59] M. M. Kessler. Bibliographic coupling between scientific papers. *American Documentation*, 14(1):10–25, January 1963.

[60] J. K. Kishore, L. M. Patnaik, V. Mani, and V. K. Agrawal. Genetic programming based pattern classification with feature space partitioning. *Information Sciences*, 131(1-4):65–86, January 2001.

[61] J. K. Kishore, Lalit M. Patnaik, V. Mani, and V. K. Agrawal. Application of genetic programming for multicategory pattern classification. *IEEE Trans. Evolutionary Computation*, 4(3):242–258, 2000.

[62] Jon M. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM (JACM)*, 46(5):604–632, 1999.

[63] Daphne Koller and Mehran Sahami. Hierarchically classifying documents using very few words. In *ICML '97: Proceedings of the Fourteenth International Conference on Machine Learning*, pages 170–178, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc.

[64] John R. Koza. *Genetic programming: On the programming of computers by natural selection*. MIT Press, Cambridge, Mass., 1992.

[65] John R. Koza, Forrest H Bennett III, David Andre, and Martin A. Keane. Four problems for which a computer program evolved by genetic programming is competitive with human performance. In *Proceedings of the 1996 IEEE International Conference on Evolutionary Computation*, volume 1, pages 1–10. IEEE Press, 1996.

[66] Aaron Krowne and Edward A. Fox. An Architecture for Multischeming in Digital Libraries. In *Proc. of ICADL-03*, pages 563–577, Kuala Lumpur, Malaysia, 2003.

[67] William B. Langdon. *Data Structures and Genetic Programming: Genetic Programming + Data Structures = Automatic Programming!* Kluwer, Boston, 1998.

[68] William B. Langdon and Riccardo Poli. *Foundations of Genetic Programming*. Springer-Verlag, New York, 2002.

[69] Steve Lawrence, C. Lee Giles, and Kurt D. Bollacker. Digital libraries and autonomous citation indexing. *IEEE Computer*, 32(6):67–71, 1999.

[70] David D. Lewis. Naive (Bayes) at forty: The independence assumption in information retrieval. In *ECML '98: Proceedings of the 10th European Conference on Machine Learning*, pages 4–15, London, UK, 1998. Springer-Verlag.

[71] Thomas Loveard. *Genetic Programming for Classification Learning Problems*. PhD thesis, RMIT University, Computer Science Department, Aug 2003.

[72] Fabrizio Luccio, Antonio Enriquez, Pablo Rieumont, and Linda Pagli. Bottom-up subtree isomorphism for unordered labeled trees. Technical Report TR-04-13, Università Di Pisa, Italy, 2004.

[73] Fabrizio Luccio, Antonio Mesa Enriquez, Pablo Olivares Rieumont, and Linda Pagli. Exact rooted subtree matching in sublinear time. Technical Report TR-01-14, Università Di Pisa, Italy, 2001.

[74] Maria J. Martin-Bautista, María Vila, and Henrik L. Larsen. A fuzzy genetic algorithm approach to an adaptive information retrieval agent. *American Society for Information Science*, 50:760–771, 1999.

[75] Andrew Kachites McCallum and Kamal Nigam. Employing EM and pool-based active learning for text classification. In *Proc. 15th International Conf. on Machine Learning*, pages 350–358. Morgan Kaufmann, San Francisco, CA, 1998.

[76] Nicholas Freitag McPhee and Riccardo Poli. Using schema theory to explore interactions of multiple operators. In W. B. Langdon, E. Cantú-Paz, K. Mathias, R. Roy, D. Davis, R. Poli, K. Balakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull, M. A. Potter, A. C. Schultz, J. F. Miller, E. Burke, and N. Jonoska, editors, *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 853–860, New York, 9-13 July 2002. Morgan Kaufmann Publishers.

[77] John Mingers. An empirical comparison of pruning methods for decision tree induction. *Mach. Learn.*, 4(2):227–243, 1989.

[78] Tom M. Mitchell. *Machine learning*. McGraw Hill, New York, US, 1996.

[79] NDLTD. Networked Digital Library of Theses and Dissertations, www.ndltd.org, 2006.

[80] OAI. Open Archives Initiative, http://www.openarchives.org/, 2006.

[81] Hyo-Jung Oh, Sung Hyon Myaeng, and Mann-Ho Lee. A practical hypertext catergorization method using links and incrementally available class information. In *Proceedings of the 23rd annual international ACM SIGIR conference on research and development in information retrieval*, pages 264–271, Athens, Greece, 2000. ACM Press, New York, USA.

[82] Una-May O'Reilly and Franz Oppacher. The troubling aspects of a building block hypothesis for genetic programming. In L. Darrell Whitley and Michael D. Vose, editors, *Foundations of Genetic Algorithms 3*, pages 73–88, Estes Park, Colorado, USA, 31 July–2 August 1994. Morgan Kaufmann.

[83] Una-May O'Reilly and Franz Oppacher. Using building block functions to investigate a building block hypothesis for genetic programming. Working Paper 94-02-029, Santa Fe Institute, 1399 Hyde Park Road Santa Fe, New Mexico 87501-8943 USA, 1994.

[84] Frederick E. Petry, Bill Buckles, Devaraya Prabhu, and Donald Kraft. Fuzzy information retrieval using genetic algorithms and relevance feedback. In *Proceedings of the ASIS Annual Meeting*, pages 122–125, 1993.

[85] Riccardo Poli. Exact schema theorem and effective fitness for GP with one-point crossover. In Darrell Whitley, David Goldberg, Erick Cantu-Paz, Lee Spector, Ian Parmee, and Hans-Georg Beyer, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000)*, pages 469–476, Las Vegas, Nevada, USA, 8-12 July 2000. Morgan Kaufmann.

[86] Riccardo Poli. Hyperschema theory for GP with one-point crossover, building blocks,and some new results in GA theory. In R. Poli, W. Banzhaf, W. B. Langdon, J. Miller, P. Nordin, and T. C. Fogarty, editors, *Proceedings of the Third European Conference on Genetic Programming (EuroGP-2000)*, volume 1802 of *LNCS*, pages 163–180, Edinburgh, Scotland, 2000. Springer Verlag.

[87] Riccardo Poli. General schema theory for genetic programming with subtree-swapping crossover. In Julian F. Miller, Marco Tomassini, Pier Luca Lanzi, Conor Ryan, Andrea G. B. Tettamanzi, and William B. Langdon, editors, *Genetic Programming, Proceedings of EuroGP'2001*, volume 2038, pages 143–159, Lake Como, Italy, 18-20 April 2001. Springer-Verlag.

[88] Riccardo Poli and Nicholas Freitag McPhee. Exact schema theory for GP and variable-length GAs with homologous crossover. In Lee Spector, Erik D. Goodman, Annie Wu, W. B. Langdon, Hans-Michael Voigt, Mitsuo Gen, Sandip Sen, Marco Dorigo, Shahram Pezeshk, Max H. Garzon, and Edmund Burke, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 104–111, San Francisco, California, USA, 7-11 July 2001. Morgan Kaufmann.

[89] Vijay V. Raghavan and Brijesh Agarwal. Optimal determination of user-oriented clusters: an application for the reproductive plan. In John J. Grefenstette, editor, *Proceedings of the 2nd International Conference on Genetic Algorithms and their Applications*, pages 241–246, Cambridge, MA, July 1987. Lawrence Erlbaum Associates.

[90] C. Radhakrishna Rao. Diversity and dissimilarity coefficients: A unified approach. *Journal Theoretical Population Biology*, 21:24–43, 1982.

[91] Brainingstorming Report. Digital Libraries: Future Directions for a European Research Programme. San Cassiano, Alta Badia, Italy, 13-15 June 2001.

[92] Berthier A. N. Ribeiro and Richard Muntz. A belief network model for IR. In *SIGIR '96: Proceedings of the 19th annual international ACM SIGIR conference on research and de-*

velopment in information retrieval, pages 253–260, Zurich, Switzerland, 1996. ACM Press, New York, USA.

[93] Berthier Ribeiro-Neto, Ilmério Silva, and Richard Muntz. *Soft Computing in Information Retrieval: Techniques and Applications*, chapter 11—Bayesian Network Models for IR, pages 259–291. Springer Verlag, 1st edition, 2000.

[94] C. J. Van Rijsbergen. *Information Retrieval*. Butterworth-Heinemann, Newton, MA, USA, 1979.

[95] Justinian Rosca. Towards automatic discovery of building blocks in genetic programming. In E. V. Siegel and J. R. Koza, editors, *Working Notes for the AAAI Symposium on Genetic Programming*, pages 78–85, MIT, Cambridge, MA, USA, 10–12 November 1995. AAAI.

[96] Maytal Saar-Tsechansky and Foster Provost. Active learning for class probability estimation and ranking. In Bernhard Nebel, editor, *Proceedings of the Seventeenth International Conference on Artificial Intelligence (IJCAI-01)*, pages 911–920, San Francisco, CA, August 4–10, 2001. Morgan Kaufmann Publishers, Inc.

[97] Gerard Salton. *Automatic Text Processing*. Addison-Wesley, Boston, Massachusetts, USA, 1989.

[98] Gerard Salton and Chris Buckley. Term-weighting approaches in automatic text retrieval. *Information Processing and Management*, 24(5):513–523, 1988.

[99] Kumara Sastry, Una-May O'Reilly, David E. Goldberg, and David Hill. Building-block supply in genetic programming. In Rick L. Riolo and Bill Worzel, editors, *Genetic Programming Theory and Practice*, chapter 9, pages 137–154. Kluwer, 2003.

[100] Fabrizio Sebastiani. Machine learning in automated text categorization. *ACM Comput. Surv.*, 34(1):1–47, 2002.

[101] Dennis Shasha, Jason Tsong-Li Wang, Kaizhong Zhang, and Frank Y. Shih. Exact and approximate algorithms for unordered tree matching. *IEEE Transactions on Systems, Man, and Cybernetics*, 24(4):668–678, 1994.

[102] Ilmério Silva, Berthier Ribeiro-Neto, Pável Calado, Edleno Moura, and Nívio Ziviani. Link-based and content-based evidential information in a belief network model. In *SIGIR '00:*

Proceedings of the 23rd annual international ACM SIGIR conference on research and development in information retrieval, pages 96–103, Athens, Greece, 2000. ACM Press, New York, USA.

[103] Amit Singhal, Mandar Mitra, and Chris Buckley. Learning routing queries in a query zone. In *SIGIR '97: Proceedings of the 20th annual international ACM SIGIR conference on research and development in information retrieval*, pages 25–32, Philadelphia, Pennsylvania, USA, 1997. ACM Press, New York, USA.

[104] Henry G. Small. Co-citation in the scientific literature: A new measure of relationship between two documents. *Journal of the American Society for Information Science*, 24(4):265–269, July 1973.

[105] Ashwin Srinivasan. A study of two sampling methods for analysing large datasets with ILP. *Data Mining and Knowledge Discovery*, 3(1):95–123, 1999.

[106] Chris Stephens and Henri Waelbroeck. Schemata evolution and building blocks. *Evolutionary Computation*, 7(2):109–124, 1999.

[107] Chris R. Stephens and Henri Waelbroeck. Effective degrees of freedom in genetic algorithms and the block hypothesis. In Thomas Bäck, editor, *Proceedings of the Seventh International Conference on Genetic Algorithms (ICGA97)*, San Francisco, CA, 1997. Morgan Kaufmann.

[108] Aixin Sun, Ee-Peng Lim, and Wee-Keong Ng. Web classification using support vector machine. In *Proceedings of the Fourth International Workshop on Web Information and Data Management*, pages 96–99, McLean, Virginia, USA, 2002. ACM Press, New York, USA.

[109] Eiichi Tanaka. A metric between unrooted and unordered trees and its bottom-up computing method. *IEEE Trans. Pattern Anal. Mach. Intell*, 16(12):1233–1238, 1994.

[110] Turtle, Howard, Croft, and W. Bruce. Evaluation of an inference network-based retrieval model. *ACM Transactions on Information Systems*, 9(3):187–222, 1991.

[111] Gabriel Valiente. An efficient bottom-up distance between trees. In *SPIRE 2001: Proceedings of the 8th International Symposium on String Processing and Information Retrieval*, pages 212–219, Santiago, Chile, 13-15 November 2001.

[112] Gabriel Valiente. *Algorithms on Trees and Graphs*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2002.

[113] Vladimir N. Vapnik. *The Nature of Statistical Learning Theory*. Springer, New York, 1995.

[114] Michael D. Vose. *The simple genetic algorithm: foundations and theory*. MIT Press, Cambridge, MA, 1999.

[115] Haonan Wang. *Functional data analysis of populations of tree-structured objects*. PhD thesis, University of North Carolina-Chapel Hill, Department of Statistics, 2003.

[116] Andreas S. Weigend, Erik D. Wiener, and Jan O. Pedersen. Exploiting hierarchy in text categorization. *Inf. Retr*, 1(3):193–216, 1999.

[117] System X. Virginia Tech Terascale Computing Facility: System X, http://www.tcf.vt.edu, 2006.

[118] Jing-Jye Yang and Robert Korfhage. Effects of query term weights modification in document retrieval: a study based on a genetic algorithm. In *Proceedings of the Second Annual Symposium on Document Analysis and Information Retrieval*, pages 271–285, Las Vegas, NV, April 1993.

[119] Jing-Jye Yang, Robert Korfhage, and Edie Rasmussen. Query improvement in information retrieval using genetic algorithms - A report on the experiments of the TREC project. In *Text REtrieval Conference (TREC) TREC-1 Proceedings*, pages 31–58. Department of Commerce, National Institute of Standards and Technology, 1992. NIST Special Publication 500-207: The First Text REtrieval Conference (TREC-1).

[120] Yiming Yang. Expert network: effective and efficient learning from human decisions in text categorisation and retrieval. In W. Bruce Croft and Cornelis J. van Rijsbergen, editors, *Proceedings of SIGIR-94, 17th ACM International Conference on Research and Development in Information Retrieval*, pages 13–22, Dublin, Ireland, 1994. Springer Verlag, Heidelberg, Germany.

[121] Yiming Yang and Xin Liu. A re-examination of text categorization methods. In *22nd Annual International SIGIR*, pages 42–49, Berkley, August 1999.

[122] Yiming Yang, Seán Slattery, and Rayid Ghani. A study of approaches to hypertext categorization. *Journal of Intelligent Information Systems*, 18(2-3):219–241, 2002.

[123] Mohammed J. Zaki. Efficiently mining frequent trees in a forest. In *KDD '02: Proceedings of the eighth ACM SIGKDD international conference on knowledge discovery and data mining*, pages 71–80, Edmonton, Alberta, Canada, 2002. ACM Press, New York, USA.

[124] Baoping Zhang, Marcos André Gonçalves, and Edward A. Fox. An OAI-based filtering service for CITIDEL from NDLTD. In *Proc. of ICADL-03*, pages 590–601, Kuala Lumpur, Malaysia, 2003.

[125] Kaizhong Zhang and Dennis Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM J. Comput.*, 18(6):1245–1262, 1989.

[126] Kaizhong Zhang, Jason Wang, and Dennis Shasha. On the editing distance between undirected acyclic graphs. *IJFCS: International Journal of Foundations of Computer Science*, 7(1):43–57, 1996.