A Genetic Algorithm for Query Optimization

Alexandra Russell

Department of Computer and Information Sciences

University of Strathclyde, Glasgow

August 26, 2019

Declaration

This dissertation is submitted in part fulfilment of the requirements for the degree of MSc of the University of Strathclyde.

I declare that this dissertation embodies the results of my own work and that it has been composed by myself. Following normal academic conventions, I have made due acknowledgement to the work of others.

I declare that I have sought, and received, ethics approval via the Departmental Ethics Committee as appropriate to my research.

I give permission to the University of Strathclyde, Department of Computer and Information Sciences, to provide copies of the dissertation, at cost, to those who may in the future request a copy of the dissertation for private study or research.

I give permission to the University of Strathclyde, Department of Computer and Information Sciences, to place a copy of the dissertation in a publicly available archive. (please tick) Yes [x] No [ ]

I declare that the word count for this dissertation (excluding title page, declaration, abstract, acknowledgements, table of contents, list of illustrations, references and appendices is 12702 words.

I confirm that I wish this to be assessed as a Type 5 Dissertation

Signature:

Date: 26/08/2019

Abstract

The growing quantities of unstructured textual information online can represent an untapped goldmine of data, but one of the barriers to exploiting it is finding where these resources are. Information retrieval systems and search engine technologies have developed to tackle this need. However, there are elements of them that can be further developped to suit our needs. In particular, the problem of user queries not being optimal for the search they are trying to perform persists. If we could optimize the query being provided to a retrieval system this could go a long way to improving the quality of information users are receiving from search engines.

In this dissertation the applicability of a genetic algorithm to query optimization within the context of information retrieval is explored. First and foremost the goal is to investigate whether this is an effective way of altering a search query to improve retrieval. However, the research is done within the context of trying to develop technologies that would allow easier research and data collection via internet search engines. This specific context requires a unique experimental design framework where the information retrieval system, and the document collection it has indexed, provide limited information to the query expansion techniques being applied.

This report shows that a genetic algorithm approach to query optimization proves more effective than other query expansion techniques for retrieval, and introduces a framework for performing such query optimization applicable to any search engine.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Tim Berners-Lee gave a Ted Talk in 2009 entitled "The next web of open linked data". In this talk he speaks about the motivation behind creating the World Wide Web. He was frustrated with the unlocked potential lying in isolated documents across the world; if they could be linked together they could create new and exciting information and opportunities. "If these documents could be linked and made available in some big virtual documentation system in the sky", he says, "life would be so much easier". Creating the World Wide Web allowed for just that. The availability of information from wherever you are in the world has had an incalculable impact on our how our lives, society and science have developed. In this Ted Talk, twenty years after creating the World Wide Web, Tim Berners-Lee is speaking again about connecting data. This time, he is introducing a project aiming to create linked data across the internet to unlock the data potential that exists, untapped, online. His project proposal tackles the problem lying in unstructured information online, or as he says: "we haven't got data on the web as data" (Ted 2009).

10 years on from this talk, the same problems still exist with information online. More information is available online than ever before, and the rate of increase in information is accelerating. Some estimates claim that by 2020 every person will be creating 1.7

megabytes of new information every second (NodeGraph 2017). However, a lot of that information is unstructured textual information scattered across the web. This is inconvenient as there is no easy way for a computer to understand that kind of information and collecting information sources on a topic can be a challenge. However, with necessity being the mother of all invention, this inconvenience has generated countless new technologies that seek to help us exploit and understand the abundance of information online.

Examples of these technologies include internet crawlers; applications that can follow links online, crawling through web domains, collecting information as they go. Another example is natural language processing, which takes unstructured text and extracts information from it that computers can understand. However, arguably the most wide spread technology to have come from the explosion of availability of online data has to be the internet search engine.

Information retrieval systems are a type of application that takes unstructured information and allows you to search through it. Internet search engines are a subset of these applications where the unstructured information in question is web pages. Information retrieval system were around before the internet, but the rise of the World Wide Web boosted the development of these types of technologies astronomically. However, the development of information retrieval systems faces challenges and amongst them sits the problem of converting a user's abstract information need into a set of relevant documents via the medium of a query string that is input into the information retrieval system. It is this challenge that this paper explores.

The process from information need to relevant documents has two distinct groups of difficulties. The first lies in the fact that often the query that the user formulates is not the best way to characterize their information need, and many users with the same need will not phrase it in the same way. The second difficulty concerns the application

itself, going from a string of words to a collection of relevant documents is a complex one. This paper focuses on the formulation of the best possible query to satisfy an information need.

A technique used to address the problem of an initial query not being informative enough for the system, or being unfit for the information need it is trying to characterize, is query expansion. This involves adding terms to the search query in such a way that the system returns better results. There are lots of ways to perform query expansion, but some interesting ones involve using different types of algorithms to add to the query based on the results returned by the initial search with the initial query. When considering the different types of algorithms that can be used to expand a query, one might consider the family of genetic algorithms.

Based on the mechanism behind the theory of evolution, genetic algorithms are a promising way to explore large search spaces and converge to optimal solutions. This paper explores whether a genetic algorithm that would 'evolve' towards an optimal query string could be a good way to tackle query expansion.

Before beginning to explore this idea however, it is prudent to think about exactly what kind of information need is most interesting for this type of query expansion. Thinking back to the enormous amount of unstructured data that exists online, there are some interesting opportunities it presents. In particular, there is huge potential for researching topics that may not have centralized repositories of information. There are many examples of such topics, however Milosevic et al. (2018) focuses on the particular example of social innovation projects and illustrates the potential of online information in that field. There is an appetite to collect data about social innovation projects, however there are few comprehensive collections of data about them that would be sufficient for in-depth research about the domain as a whole. However, because social innovation projects are often small, run primarily with public good objectives and de-

veloped by start-ups or small companies, there is a huge amount of information about them made available publicly online (Milosevic et al. 2018). If the online, unstructured information about these types of projects was able to be harnessed, the research opportunities in that field would be extensive.

With that kind of broad information need the first step in collecting information about projects like this is being able to find the information sources you want to extract the information from. This scenario is to be borne in mind throughout this paper; a user seeking information on a topic and wants to create the best way of searching for information or data about that topic. This scenario is particularly apt for applying query expansion techniques to because a user will not have access to the inner workings of the search engine they are using. So the only way to optimize the system is to optimize the query.

In this paper the application of genetic algorithms is explored. First through an overview of background information in chapter 2 and a literature review of the ongoing research in the field in chapter 3. A methodology is then proposed to evaluate how feasible and useful such an application would be. Next, an implementation is detailed, giving the ability to test the hypotheses outlined in chapter 4. The results of those experiments are then presented in chapter 5, allowing a conclusion in chapter 6 on the effectiveness of genetic algorithms in this context and a presentation of the ways this research could be furthered and improved.

Chapter 2

Background

As is hopefully obvious, the two key topics of this dissertation are information retrieval and genetic algorithms. This chapter covers background on both topics to establish some common ground knowledge. It introduces how these two techniques can be used in conjunction and what the desired outcome of such a pairing would be.

2.1 Information Retrieval

In the broadest sense of the term, information retrieval is a process that allows a user to find some information amongst sources that are not explicitly searchable. Today, the term is mostly applied to techniques and technologies for searching through unstructured digital documents (often web pages) to respond to a users information need.

We can understand information retrieval broadly as the following process (Manning et al. 2010):

> *Information retrieval (IR) is finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers).*

In other words, an information retrieval system takes an information need and a collec-

tion of documents and returns a subset of those documents, ranked according to their relevance to that information need.

When it comes to the modern understanding of information retrieval we mostly think about search engines for the world wide web. The first web search engine was developed about 30 years ago when, at McGill University, computer science students Alan Emtage, Bill Heelan and J. Peter Deutsch developed Archie. This tool downloaded directory listings of files on public anonymous sites and created a searchable database (Seymour et al. 2011). At the time this tool was sufficient given the small searchable space, but since then the number of online resources has increased exponentially, and thus information retrieval technologies have developed fast.

The use of information retrieval systems quickly overtook querying databases as the most popular form of finding information. The information retrieval technologies that search the web have developed and today the search engine industry is huge. Forbes recently estimated that in the United States the search engine optimization (SEO) industry would be worth 80 billion US dollars by 2020 (McCue 2018).

The architecture of most information retrieval systems is shown in Figure 2.1. This paper focuses on the development of the best query possible to characterize an information need, so the section of the architecture we will focus on are the steps from information need to query. An example of this particular stage of the process would be when using an online search engine, the user has an abstract information need that they wish to satisfy. The user formulates that need into a query, which is then typed into the search bar and submitted to the search engine. However, that is a very basic example of the process which assumes that the search engine used takes the query as is and uses it for matching. In reality, there will probably be much more going on behind the scenes with that initial query before the matching algorithm handles it. More information will be added surrounding that initial query. That information

could be data about the user themselves, information about their previous searches, or additional terms might be added to the user's original search terms to make the query be more informative.

Often the user input query text is not sufficient for the information retrieval system, so they are built to expand the query to return more relevant results. The reason the query text is often not sufficient is is two-fold. The first reason relates to the nature of language and the second relates to the nature of users interacting with such a system. Firstly, language creates difficulties in effective querying, which Deerwester et al. (1990) boils down to two factors; 'synonymy' and 'polysemy'. Synonymy refers to the fact that you can use multiple words or phrases to refer to a singular object or concept. Examples include 'sidewalk' and 'pavement' or 'buy' and 'purchase'. Synonymy creates situations where a user offers a query using a particular word, but a useful document only contain it's synonyms. Polysemy is the opposite side of the linguistic coin, where one word refers to two different things. Examples of this include 'Apple', which could designate the fruit or the tech company . However, additional difficulties lie in the inherent nature of search engines, mainly that users are trying to satisfy an information need so by definition this need represents a blind spot in their knowledge. Thus the chances are the user isn't using the best terms to characterize this need, or they wouldn't be searching for it in the first place. This second issue is perhaps the most difficult to tackle as it is the most intangible.

Many different techniques have been developed to combat these difficulties, but a popular technique has proved to be altering the initial query. This is known as query expansion (if you are adding terms to the initial query) or query reformulation (if you are changing the initial query). Both are query optimization techniques, the technique at the core of this paper's research.

Query optimization is a particularly interesting aspect of search engine optimization

because it has two different possible points of application. It can be an internal component of your information retrieval system or be a technique applied externally. It is the latter case focused on here due to the applications it has to the generalized collection of information via the medium of a pre-existing search engine.

Consider a scenario where a user is trying to collect information on a certain topic using a search engine. This is an increasingly popular task as there is an abundance of unstructured information available on the web and if this can be appropriately harnessed there are ample research opportunities. The user cannot alter the search engine's algorithm, all they can change is the query being input; it would be prudent to optimize it to return the best results. This is the case this paper addresses, looking at different techniques to optimize an input query to maximize the relevant results returned to the user.

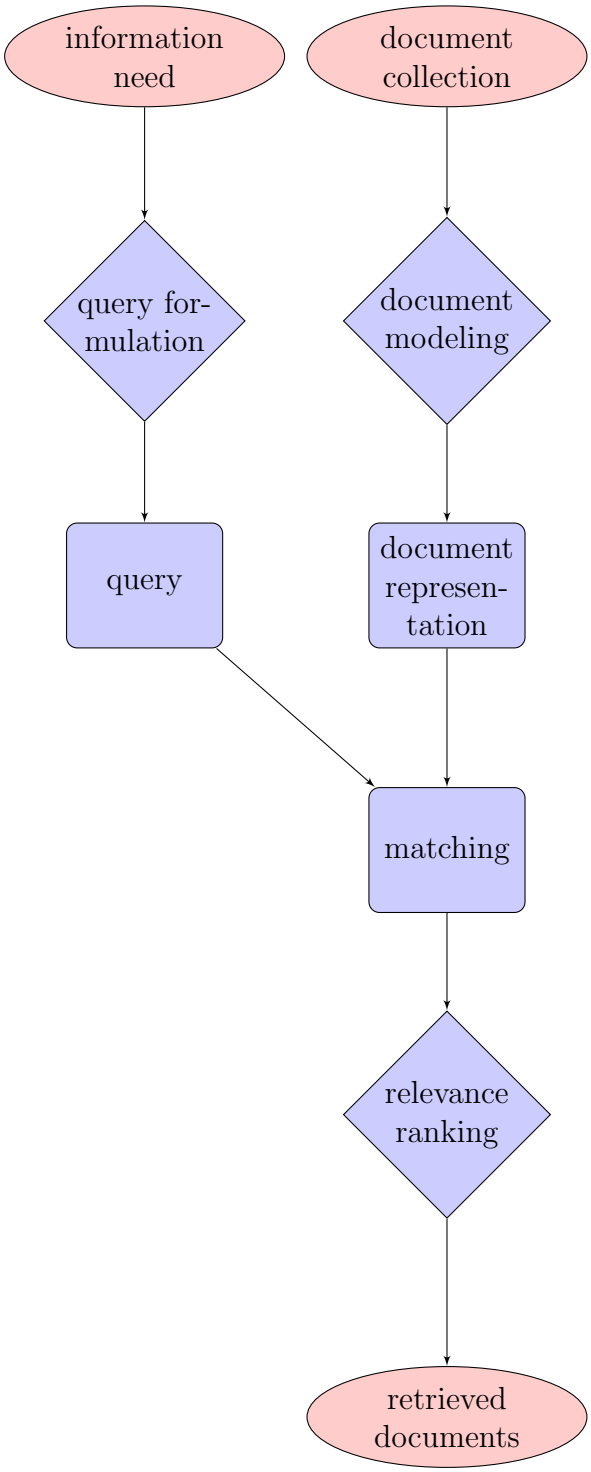

Figure 2.1: Information retrieval system architecture

2.2 Genetic Algorithms

Genetic algorithms are an optimization technique inspired by the biological theory of evolution (Kramer 2017). In the natural world, populations evolve slowly over time to become more optimal (or more adapted to their environment). This change occurs through successive populations being generated through breeding. The populations improve because the better an individual, the more likely they are to be a successful breeder and generate viable offspring.

Algorithms for finding optimal solutions to problems have been inspired by this process in nature and now genetic algorithms are a popular mechanism for exploring search spaces and returning solutions. They are particularly useful in large search spaces were hill climber or gradient descent models are likely to converge towards locally optimal solutions (Ingber & Rosen 1992). Genetic algorithms tend to follow the general process in Figure 2.2.

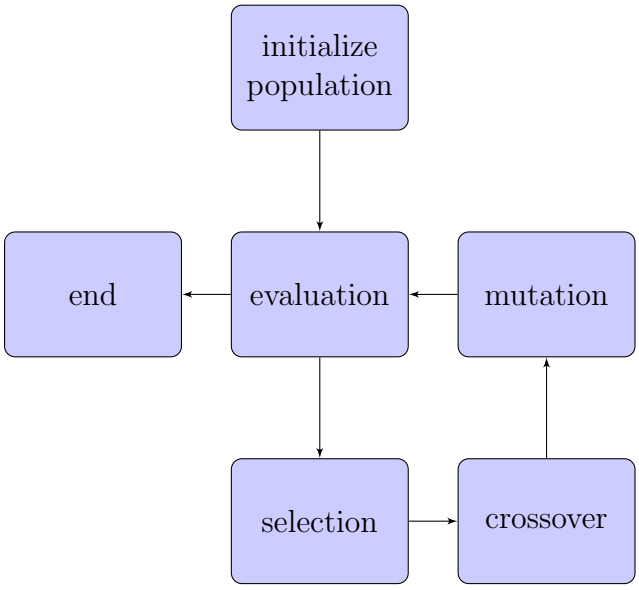

Figure 2.2: Genetic algorithm process

We can take a brief look at each of these stages to understand how the process functions.

- **Initialize Population:** During this first step the initial population is created to start the algorithm. At this stage the representation of an individual must be determined (which will represent a potential solution to your problem). Usually an individual is represented by an array and is initialized with random contents (or as random as they can be according to certain rules of the problem domain).

- **Evaluation:** Individuals in the population are evaluated to find the best solution (or set of solutions) so far.

- **Selection:** Pairs of individuals are chosen to breed and create the next generation. There are a variety of ways to make this selection, typically a balance is being struck between over-fitting and being completely random.

- **Crossover:** With the 'parents' chosen, they are combined using whatever technique is most appropriate to create new 'children'. The simplest technique, assuming the individuals are represented by an array, is to take the first half of one parent and combine it with the second half of the other to generate a child.

- **Mutation:** To further mimic the natural evolutionary process, random mutations are introduced into the child population. This creates more diversity in the population and helps to avoid converging towards locally optimal solutions.

- **Iterate:** This brings us back to the evaluation phase. Here, the process its iterated a set number of times or until the individuals reach a certain predefined criteria.

- **End:** At the end of the algorithm's iterations the best solution will have been recorded and is returned as the proposed solution to the problem.

With respect to query expansion, a genetic algorithm could be an interesting way to explore the search space of potential query terms, and as we shall see is particularly

adept when considering a system where we have no interaction with the rest of the retrieval system.

Chapter 3

Literature Review

3.1 Query Optimization

Query optimiztion is a method whereby words are added, removed or changed in the original query supplied by the user of an information retrieval system. This method is applied to overcome the inaccuracy within information retrieval systems that comes from the initial query users supply, which is arguably the source of most inaccuracies within IR systems (Carpineto & Romano 2012). There is considerable research into this method, with even some work done into machine learning techniques to recommend the best query expansion method to a user (Haiduc et al. 2013). There are a variety of techniques that can be used for query optimization and these can be categorized as either automated or semi-automated. Semi-automated techniques will typically produce potential terms to expand the query, but the user makes the final decision about whether they should be added to the query or not (Qiu & Frei 1993). However these techniques have not been very successful, as discussed in the Cuna Ekmekcioglu et al. paper (1992) where they explore semi automated expansion compared to initial query search results. This lack of success is likely caused by the fact that the user query represents an information blind spot for the user, so choosing better query terms can be a stab in the dark. In addition, when we are trying to those better query terms,

'better' will be entirely dependant on the way the information retrieval system works internally, which the user will have no visibility of. For this reason, this paper focuses on automated query optimization, as it bypasses the issues created by the user, which is an advantage to the system but also the user.

Within the field of automated query optimization techniques, some are known as relevance feedback systems. This is where the optimization uses information or terms from relevant documents retrieved in a previous search (Robertson 1990). Alternatives techniques can use information about the initial query (adding synonyms of query terms for example) or information about the user (how they have interacted with a search system in the past for example) as a way to expand (Cui et al. 2003). This paper focuses on relevance feedback systems primarily due to the availability of data to perform that kind of task, but also because when considering applying these techniques to research purposes, you wouldn't want to build a system to dependant on a particular individual's profile.

When considering automatic relevance feedback based query optimization there are lots of different techniques to chose from. A tried and tested method is the Rocchio algorithm (Rocchio 1971), which produces a new weight vector from an existing weight vector using a set of training examples (Lewis et al. 1996). Weight vectors in this context refer to vectors that represent potential queries within a vector space of features (features here meaning words that are potential query terms).

As an aside, it is worth explaining at this point what the vector space model is as it features heavily in the query optimization methods and also features later in the implementation section of this report. The vector space model is a way of representing text. Each text element you need to model is represented by a vector of terms. These terms are typically words, with every word in the vocabulary being an independent dimension in a very high dimensional vector space (Singhal et al. 2001). So each doc-

ument is represented by a vector, and the elements of that vector are the frequencies of different terms. Using this representation, you can determine similarity between documents and queries using some measure of distance between the vector of the query and the vector of the document. In this paper the vector space model is not used to determine similarity between documents and queries, however it is used to represent queries and documents as vectors in order to apply different algorithms to them for the purposes of query expansion and feature selection.

Other algorithmic techniques for query expansion include the Widrow-Hoff Algorithm, the Kivnen Warmuth Algorithm (Lewis et al. 1996) and perceptron learning algorithms (Ng et al. 1997) all of which use gradient descent procedures and the vector space model representation. The weights of the query vector are updated as you iterate through a series of training data instances. At this point we can note the appeal of a genetic algorithm for this kind of problem as they have been praised in the past for outperforming gradient descent procedures as they are less likely to converge to a local optima (Ingber & Rosen 1992).

At this point it would be amiss not mention how different machine learning techniques have been applied to information retrieval systems. Mostly these techniques are not applied to query expansion but rather to optimizing the system itself. In fact they are particularly popular when it comes to the classification of retrieved documents rather than query based retrieval. For example, in the work by Lei (2012) a Naive Bayes classifier is used to classify text documents, and in the work by Uğuz (2011) a K-Nearest-Neighbours method and a decision tree are also applied to a similar problem. Interestingly both of these retrieval systems have genetic algorithm components, as will be discussed in the next section. There has also been work into using Support Vector Machines, but again mostly for the purposes of classification (Glover et al. 2001).

3.2 Genetic Algorithms

Genetic algorithms have been widely used within the field of information retrieval systems, but are less common within that of query optimization. They have been used when applied to the indexing and matching components of the retrieval system (Chen 1995, Vrajitoru 1998, Pathak et al. 2000). They are very often applied to 'feature selection' techniques, such as in the aforementioned works of Lei (2012) and Uğuz (2011) as well as the work of Chen et al. (1998) and many more. Or they are used to modify the representation of documents and queries (Pathak et al. 2000).

Feature selection is the process by which important features (here, words) are chosen to help improve efficency and accuracy of a retriveal system. This technique is discussed in the next section as it will prove to be a crucial component of the final implementation, however at this stage we can note that genetic algorithms are a popular tool for performing such a selection. In the three papers mentioned above genetic algorithms are used to reduce the dimensions of documents to improve search speeds and reduce complexity and all note the effectiveness of these techniques.

When looking at how query expansion can be performed using genetic algorithms the papers by Horng and Teh (2000), Al Mashagba et al. (2011) and Abdelmgeid (2007) all propose implementations. These three papers use genetic algorithms to expand queries and are applied to document collections in Chinese, Arabic and English respectively. Each reports improved retrieval results after the application of the genetic algorithm to the query compared to the original seed query. Among those papers the Al Mashagba et al. paper is of particular interest for it's experimental design. The genetic algorithm component does not use any information about the retrieval system or the overall document collection. In short, any retrieval system could be used and the mechanism of the genetic algorithm would be the same. The query expansion uses only information from documents that are returned from an initial seed query put to the system. This

method means that this technique could be applied to any information retrieval system out there so long as it accepts a query and that there is some mechanism for identifying relevant documents once a set of potentially relevant documents has been returned by the system. This mechanism for identifying relevant documents could be a purpose built classifier or a manual, on the fly, identification by a user for example.

This brings us back to the appeal of query expansion mentioned in the introduction; that it can be done regardless of the type of information retrieval system you use. In fact the paper by Abdelmgeid highlights the applicability of a genetic algorithm to that kind of situation. However, this paper dealt exclusively with text in Arabic and reminds us to bare in mind that the Arabic language is very different to English, particularly with respect to the frequency of individual words. With this in mind it can be argued it is worth recreating a system similar to the one devised in that paper to measure the effectiveness of a genetic algorithm being applied to query expansion in this way, using document collections and queries in English. There is also scope to test different parameters and design of genetic algorithm as well. Finally, it could be interesting to evaluate the system in different ways, comparing the genetic algorithm approach to other query expansion techniques, as the paper only compares the results of the genetic algorithm to the initial query's rate of success.

3.3 Feature Selection Techniques

As previously mentioned, the feature selection mechanism is a practice where a subset of features are chosen amongst the full original feature set according to some criteria of feature importance (Uğuz 2011). It can be used as a way to reduce the number of dimensions in your document representation, improving the efficiency of your retrieval (as a document is represented by a smaller object), or it can be used to select potential features for query expansion. This is typically applied when the model of the information retrieval system is already a vector space model, but can also be used in other

cases. For our purposes we will be using feature selection to create a subset of terms which are candidates for the query expansion, as it would be unrealistic and inefficient to test all terms that exist within a collection.

There are different techniques that can be used for this selection and as with information retrieval systems as a whole, there are automatic and semi-automatic techniques for feature selection. In fact the research done by Ng, Goh and Low found that at least some manual input into feature selection improved the results of the querying (Ng et al. 1997).

However, for our purposes we will focus on automatic techniques and the most popular automatic techniques are information gain, statistical document frequency, mutual information, x^2-test and term strength (Yang & Pedersen 1997). The comparative study by Yang and Pedersen finds that the most effective techniques for feature selection (using a KNN classifier) were information gain and mutual information, with document frequency close behind and being the simplest method to use. However their case was feature selection for the purposes of document classification rather that retrieval. As mentioned before, genetic algorithms have also been used successfully for feature selection.

Armed with the promising results of genetic algorithms within query expansion, this paper proposes an information retrieval system that uses a genetic algorithm to expand queries to return more relevant documents. The implementation is such that it models a scenario where the query expansion has no visibility into the underlying structure of the information retrieval system, or the document collection it is retrieving from. This allows these techniques to be completely applicable to any information retrieval system.

Chapter 4

Methodology

This chapter covers the hypotheses we are seeking to test, the experimental design used to test them, and finally the implementation of the experimental design.

4.1 Hypotheses

This paper seeks to explore how effectively a genetic algorithm can be used to optimize a query submitted to an information retrieval system. To explore this area three research hypotheses have been formulated. These will be evaluated to prove or disprove their statements and, in doing so, assess this research area.

The first hypothesis is simple and obvious;

1. A genetic algorithm can be used as a tool for query optimization.

This hypothesis has already been addressed in previous research as outlined in the literature review. However, we can consider the fact that many of those studies used languages other than English and that their experimental designs were slightly different. With this in mind it is interesting to address this hypothesis again, as a stand alone problem. This hypothesis will be addressed simply by implementing a functioning information retrieval system with a genetic algorithm for query optimization.

Next, we would like to address the quality of the genetic algorithm's performance with the following hypothesis;

2. A genetic algorithm method of query expansion will perform better than other query expansion methods.

Addressing this will involve implementing alternative query expansion techniques to compare the genetic algorithm to.

The final hypothesis is perhaps the most interesting one, as addressing it is the first step in exploring this type of query optimization's practical applications. We shall be exploring whether or not a genetic algorithm approach to query expansion would be useful for information gathering across the internet using pre-existing search engines.

3. Query optimization can be effectively performed using a genetic algorithm without input from the information retrieval system itself.

The design of our information retrieval system will be crucial to evaluate this hypothesis, and we shall have to be mindful of all information that crosses from the information retrieval system to the query development side.

Using these hypotheses as a guide, the next section explores the experimental design used in this project, making reference to how components of the design allow us to evaluate specific hypotheses.

4.2 Experimental Design

Having defined the hypotheses for this project the following experimental design will serve as a framework in which to evaluate them. The actual implementation of this design is detailed in section 4.3.

Overall, this experiment takes a simple information retrieval system into which a query is input and a list of documents the system deems relevant is output. This system is built in java using Lucene, a java text search engine library. The document collection is a TREC collection which is accompanied with a list of predefined queries and an associated list of relevant documents. The system uses these to evaluate the system's effectiveness at returning relevant results.

An initial search will be run using a seed query from the predefined list of queries, this will return a set of documents that may or may not be relevant to the query. This first set of returned documents will serve as the baseline comparison when evaluating the effectiveness of the genetic algorithm's query expansion. However in addition to this, it serves to provide a set of documents that will form the basis of the query expansion techniques. It is only using the content of the documents returned by the seed query that allow us to satisfy the experimental design conditions to evaluate hypothesis 3. By only using documents that have been returned we ensure that this experiment could be repeatable using any information retrieval system so long as it returns documents.

A feature selection technique will be applied to the returned set of documents to create a list of potential query terms. This is an example of a case when hypothesis 3 can be evaluated because the feature selection only uses documents returned by the information retrieval system. As we are only using the information that the information system has made externally available, this accurately models an external search engine

system. The feature selection will be performed using a technique called information gain, mentioned in chapter 3 and highlighted for it's efficiency in scenarios like this one.

Once features have been selected, two different query optimization techniques will be used to optimize the seed query. The first method is the Rocchio algorithm and the second is a genetic algorithm. Both were implemented from scratch in java and the design choices for both are detailed below. The Rocchio algorithm will serve as our comparison tool for the genetic algorithm implementation, allowing us to evaluate hypothesis 2. The successful implementation of the genetic algorithm will serve to evaluate hypothesis 1.

4.2.1 Feature Selection - Information Gain

'Information gain' is a value calculated for individual terms to determine how useful they are for classifying a particular set of documents in a collection. The function to calculate the information gain value for a term is as follows:

$$\begin{aligned} IG(t) = &- \sum_{i=1}^{|C|} P(C_i) log(P(C_i)) \\ &+ P(t) \sum_{i=1}^{|C|} P(C_i|t) log(P(C_i|t)) \\ &+ P(\bar{t}) \sum_{i=1}^{|C|} P(C_i|\bar{t}) log(P(C_i|\bar{t})) \end{aligned} \tag{4.1}$$

In this equation:

- **t** is a term in the collection
- **C** is all the categories in the document collection (in our case there are only two, relevant and irrelevant)

- $\mathbf{P(}C_i\mathbf{)}$ is the probability of the category C_i within the document collection
- $\mathbf{P(t)}$ is the probability of term t occurring in the collection
- $\mathbf{P(}C_i \mid \mathbf{t)}$ is the probability of C_i given t
- $\mathbf{P(}C_i \mid \bar{t}\mathbf{)}$is the probability of C_i given not t

The information gain score is calculated for all terms in relevant documents returned after the first run of the seed query. The higher the information gain value, the more useful the term is to differentiate between relevant and non relevant documents. Different quantities of features are tested as will be recorded in chapter 5 of this report.

Information gain was chosen as a technique for feature selection because of the favorable results yielded in the paper by Yang and Pedersen (1997). It has the advantage over simpler techniques such as term frequency (where the top occurring terms are chosen) because it accounts for the scenario where a term might be very popular in relevant documents, but also in irrelevant documents, thus not making it a very useful term.

To reiterate, it is the fact that the feature selection is only performed using documents that the retrieval system has returned that allows us to evaluate hypothesis 3, since it keeps the implementation of the information retrieval system and the implementation of the query expansion separate form each other.

4.2.2 Rocchio Algorithm

The query expansion technique chosen as a comparison tool for the genetic algorithm method is the Rocchio algorithm. This algorithm was chosen as a comparison due to it's historical use in the field. Additionally, as is apparent in the equation, it takes into account the initial query, assuming it will hold some relevance to the documents it is

trying to retreive. The technique is explicitly defined in terms of document and query vectors, concepts defined in the chapter 2. The algorithm functions as follows;

$$\begin{aligned} \vec{Q_m} =& (a \times \vec{Q_0}) \\ &+ (b \times \frac{1}{|D_r|} \times \sum_{\vec{D_j} \in D_r} \vec{D_j}) \\ &- (c \times \frac{1}{|D_{nr}|} \times \sum_{\vec{D_k} \in D_{nr}} \vec{D_k}) \end{aligned} \tag{4.2}$$

In this equation:

- $\vec{Q_m}$ is the modified (expanded) query vector
- $\vec{Q_0}$ is the seed query vector
- $|D_r|$ is the set of relevant documents
- $|D_{nr}|$ is the set of non-relevant documents
- $\vec{D_j}$ is a document vector
- **a, b and c** are constants that must be determined

This will generate a vector to represent the new query. When translating between the vector and the query, each element of the vector that is greater than a certain threshold is input into the new query. Initially that threshold was set to 0, but as is explored in chapter 5 changing this threshold significantly improves the results returned by the Rocchio algorithm.

This tried and tested method will allow us to situate the results of the genetic algorithm query expansion within a wider context, thus helping us to evaluate hypothesis number 2.

4.2.3 Genetic algorithm

The genetic algorithm technique is explained in detail in the Implementation section, but here we will broadly cover the theory of how it is going to find the best query.

Random queries will be generated, using the words provided by the feature selection. New generations of queries will be created by breeding queries from the previous generation. The better a query is at returning relevant results, the more likely it is to be chosen to breed. This should generate successively stronger generations of queries.

For each generation we will save the best individual and after a certain number of generations have been created, the best individual found will be returned. Using this final query the effectiveness of the system will be evaluated allowing us to evaluate hypothesis 1. Comparing the genetic algorithm's results to the results returned by the queries created by the Rocchio Algorithm and the seed queries will allow us to evaluate hypothesis 2.

4.2.4 Evaluation

To evaluate the effectiveness of an information retrieval system typically precision and recall are the two metrics used and they are defined as follows:

$$precision = \frac{|\{relevant \quad documents\} \cap \{retrieved \quad documents \quad \}|}{|\{retrieved \quad documents\}|} \tag{4.3}$$

$$recall = \frac{|\{relevant \quad documents\} \cap \{retrieved \quad documents\}|}{|\{relevant \quad documents\}|} \tag{4.4}$$

The precision value indicates the proportion of the returned results that are relevant whilst the recall value indicates the proportion of relevant results that have been returned. In our case we have chosen to use recall as our primary measure of effectiveness

as there is a maximum number of returned documents regardless so precision is less informative.

In addition to this, we will be measuring both the increase in recall between the seed queries and the expanded queriess. This will allow us to get an idea of how much the expanded queries are improving the retrieval. Finally in some cases we will look at actual number of additional retrieved documents for increased contextual information about the queries.

4.3 Implementation

In this section shall be detailed the implementation of the system. First and foremost, all of the implementation has been done in Java using Java version SE-12. This language was chosen due to the availability of the Lucene package to handle the indexing of documents.

In the code submission that accompanies this report there is a Readme file which indicates how to run the code that supports this report. It assumes the code will be run in Eclipse. This paper doesn't go into extensive detail about the programming side of the implementation, but will make reference to some of the java classes that implement the functionality discussed. However, we will discuss the parameter file LuceneConstants.java because this makes explaining certain functionality and implementation choices later on easier to understand.

4.3.1 LuceneConstants.java

The LuceneConstants class does what it says on the tin. The constants are mostly file locations and referencing them explicitly before discussion the implementation in more detail will make the implementation more understandable. Obviously these file paths need to updated for your own environment when running the code. Below is the code in the LuceneConstants.java file.

```
// LuceneConstants.java

public class LuceneConstants {
      public static final String OUTPUTFILE =
          "C:\\Users\\Alex\\Documents\\output";
      public static final String INDEXFILE =
          "C:\\Users\\Alex\\Documents\\Index";
      public static final String DATAFILE =
```

```
        "C:\\Users\\Alex\\Documents\\DOTGOV";
    public static final Boolean REINDEX = false;
    public static final String QRELS =
        "C:\\Users\\Alex\\Documents\\qrels.txt";
    public static final String QUERIES =
        "C:\\Users\\Alex\\Documents\\querytext.txt";
}
```

The 'OUTPUTFILE' is the directory of an empty folder on your machine. It is used as a storage space for some certain processes that require it, namely the indexing process.

The 'INDEXFILE' is the directory where either the index will be stored if it has not already been created, or when it is currently stored.

The 'DATAFILE' is the DOTGOV folder, this is where the TREC collection is stored in the specific TREC storage format.

The 'REINDEX' boolean is used to determine whether or not the collection need to be indexed, as the indexing process is long and should only be performed if necessary.

Finally the 'QRELS' and 'QUERIES' are references to the location of the text files containing the qrels and string seed queries respectively.

4.3.2 Lucene

Apache Lucene is an open-source search package in Java, it implements "indexing and search technology, as well as spell checking, hit highlighting and advanced analysis/-tokenization capabilities" (Lucene 2000). It is used in this application firstly to index the document collection and secondly to query the collection (using either the seed query or the expanded queries). The 'skeleton' of the code was developed following a

basic Lucene tutorial (Tutorialspoint 2019), but has been developed far beyond the framework provided by the tutorial.

This implementation uses the StandardAnalyzer class to index the collection. This uses the StandardTokenizer class which breaks down the document into individual tokens (words) according to Word Break Rules from the Unicode Text Segmentation Algorithm (Davis 2019). The tokens, once extracted, are normalised using two Lucene classes; LowerCaseFilter and StopFilter. This normalizes all the tokens by converting them to lowercase and discounts tokens that match stopwords in a list of English stop words provided by Lucene. Stop words are terms that occur so frequently that they are unlikely to be useful when indexing and querying a document collection. This will include words like 'a', 'the', 'at', 'be' etc...

Lucene uses an inverted index for searching purposes. Without going into too much detail about an inverted index, it is a very common way of indexing documents in a collection and retrieving them. It is so popular because of how efficiently it facilitates retrieval, although comes at the cost of being slower to index the documents in the first place. Simply put, the inverted index creates a mapping between terms and the documents containing them.

When indexing, the Lucene indexer creates Documents objects to represent all the documents in the collection using different fields as the components of the document. These fields can include whatever you need, the documents name, the path to access it, etc. Crucially in this case the implementaion is such that the document stores the term vector information which will simplify the access to term frequency information needed for feature selection. The term vector information is a vector space model representation of the document.

Technically, when the term vector component is used to facilitate the access to term

frequency information this is an instance of the information retrieval system interacting with the query expansion technique in a way that wouldn't be feasible in the case of an external information retrieval system such as a search engine. This would appear to contravene the design choices made to allow us to evaluate hypothesis 3. This will be further discussed in subsections 4.3.4 and 4.3.5, but this choice was made for efficiency reasons and it would be easy to same extract the information from the returned text documents directly, without accessing the index information.

Once the index has been created, a Searcher object is used to search the index using a query parser. The QueryParser can be implemented in a variety of ways, however this implementation uses simple Boolean querying to the indexed content of the body of the files.

4.3.3 TREC

We are using a TREC collection of web pages as the document collection for our retrieval system. The Text REtrieval Conference (TREC) supports "research within the information retrieval community by providing the infrastructure necessary for large-scale evaluation of text retrieval methodologies" (NIST 2000) and provides collections of documents along with queries and the associated documents these queries should retrieve.

Our retrieval system will use a subset of the queries TREC provided, retrieve documents from the TREC document collection, and use the list of relevant queries TREC provides to evaluate the systems effectiveness.

The specific TREC collection we are using is the DOTGOV collection distributed by The University of Glasgow. It contains 1.25 million documents that are crawls of .gov web sites collected in 2002. This particular collection was chosen as the documents

represent web pages and the most obvious application of this work would be to web search engines.

The documents are indexed using Lucene's functionality as described above. Unfortunately, both the index and the DOTGOV collection are too large to submit with this paper, so will be provided separately. If you wish to recreated the index the REINDEX parameter must be set to true, DATAFILE set to the location of the DOTGOV collection, OUTPUTFILE set to an empty folder in your environment and INDEX to where you would like the index stored.

This OUTPUTFILE is a crucial component of the implementation of the indexing process in this application. Due to the large number of TREC documents in the collection they have been stored in a specific way, by TREC, to save space. Many documents have been concatenated into a single text file and that has then been compressed into a .gz file. There are 4,612 compressed files, stored across 47 larger folders. However, each document needs to be stored as an individual text file to be indexed correctly by Lucene. To account for this, the application handles each of the 47 larger folders individually, and extracts the content of all the .gz files it contains. It reads the content of the .gz files and separates that into the individual documents. These individual documents are stored as text files in the aforementioned OUTPUTFILE. Once all the documents exists as text files in the OUTPUTFILE they are indexed by Lucene. After they have been indexed, those text files are deleted and the now empty OUTPUTFILE is used to store the documents from the next folder of the 47 larger folders. This process of processing the documents is handled by the CleanDOTGOV class, and the indexing by the Indexer class.

This design choice was made so as to use a minimal amount of storage space on the machine running the application. This has runtime costs if you need to index the collection again however this was deemed worthwhile given that the process must only be

run once when the index is being compiled and stored.

During the process whereby the individual document are being extracted and stored as individual text files for indexing, the CleanDOTGOV class also filters the information in the documents. The Jsoup java HTML Parser is used to strip the web page information down to the text content of the page and only this text content is saved in the text file and indexed. This avoids a scenario when the indexer is confusing HTML code as words to index.

The DOTGOV queries used and associated number of relevant documents in the collection are in Table 4.1. Only a subset of queries are being used to test the system because a significant number of the queries provided for this document collection only return one or two relevant documents. Not only would this generate complications for the implementation of the system, but this system is being designed for scenarios where the user is trying to collect a collection of documents on a topic, so queries with only one relevant document are not as interesting for us. In addition to this, for run time purposes, using a subset of queries is advantageous. This particular set of queries were chosen for their divers subject topics and their high number of relevant documents in the collection.

The query and qrel files have been provided with this submission. The query file contains all the text queries and their associated query number. The qrel file is a cross reference of all query numbers and documents, indicating whether or not they are relevant. There is not a ranking of relevant documents in this particular collection, documents are simply relevant or irrelevant to a query. Importing this information is handled by the ImportQrels class.

Table 4.1: Queries

query n^o	query	n^o of relevant docs
5	'American music'	27
12	'oil petroleum resources'	33
15	'welfare reform'	17
19	'toxic waste'	40
48	'federal and state statistics'	86
58	'automobile emissions vehicle pollution'	34
104	'space exploration'	85
136	'career information'	114
153	'technology transfer'	75
157	'federal grant programs'	147

4.3.4 Feature Selection

The implementation of the feature selection function is undertaken in the TermFrequency class, and is pretty straight forward. The only element of note however is that to calculate certain probabilities the document's term frequency vectors, that were generated when the index was created, are used. As mentioned before, this might seem in contradiction with the experimental design whereby the query expansion does not interact with the information retrieval system. However in reality this choice was made for efficiency reasons, and because for storage reasons the documents are not being stored as individual files, so reading them at this stage in the program would be complicated. If this were implemented with, for example, and external search engine, you would receive the text information contained in the web pages directly and calculating these probabilities would be easy. It's for this reason that this design choice was made and why it does not impede the validity of the experimental design.

4.3.5 Rocchio Algorithm

The implementation of the Rocchio algorithm is undertaken by the Rocchio class. Again, is an application of the algorithm detailed in the experimental design section.

Similarly to the feature selection implementation though, when the algorithm refers to the sets of relevant and non relevant documents these sets are compiled from the documents returned by the initial query. The parameters a,b and c are optimized for our case as is the term threshold, as is detailed in 5 of this report.

Similarly to the section above, when implementing the Rocchio algorithm we use the term frequency vectors generated during the indexing process. Again, these could be created from the documents directly.

4.3.6 Genetic Algorithm

The implementation of the genetic algorithm is done via the functions in the GeneticAlgorithm class and different parameters have been optimized, with the results of this optimization detailed in Section 5.1. Here, we shall detail the implementation choices of the algorithm.

The first step in the genetic algorithm process is to decide how the individuals in your population are going to be represented. In this case, they are represented by query vectors based on the features selected during the feature selection phase. This query vector can contain values of 1 or 0 denoting whether or not the term the dimension in the vector refers to is included in the query.

An example of this is if your features are the terms [‘cat’, ‘dog’, ‘apple’, ‘sunshine’, ‘basket’, ‘yellow”] an individual might be represented by the vector [1,0,0,1,0,1], which would denote the query ‘cat sunshine yellow’.

When initializing the first generation of individuals in the population, the individuals were initially created completely at random. However, we also experimented with biasing generation so that the vector values were slightly more likely to be 0 to see how

this affected returns. This is discussed in 5.

When creating the next generation of individuals we have to breed parent individuals. When choosing parents to breed, tournament selection is used. This is a technique whereby two sets of three different parents are chosen at random from the population. The best candidates from each of the two groups are determined and bred to create two new children for the next generation. It is this step that causes the evolution towards better solution; some better individuals are being chosen to breed, but they are not the best of the whole group because that would create a next generation that wouldn't be diverse enough.

The mechanism for breeding parents is as follows. A random crossover point is selected and the first child is created from the first part of the first parent (before the crossover point) and the second part of the second parent (after the crossover point). Following on from our example earlier, if we were breeding individuals [1,1,1,1,1,1] and [0,1,0,0,1,0] and the random crossover point generated was 3 the children would be [1,1,1,0,1,0] and [0,1,0,1,1,1]. Additionally, before a child is put into the new generation, in 5% of cases a mutation occurs. This means that, in 5% of children, a random element of the vector will change from a 0 to a 1 or vice versa. This is done to introduce some random changes into the population for increased diversity.

Finally, we have to define how we are determining the best solutions. This is done quite simply by inputting the query into the previously explained Lucene IR system and counting the number of relevant documents returned. The query that returns the most documents is the best solution.

Chapter 5

Analysis

5.1 Optimization of Parameters

Most of the components of this implementation have some parameters that need to be defined. This section explores and analyses the different results from these optimization processes. Lastly, the final results of the best version of the query optimization techniques are presented and discussed.

5.1.1 Feature Selection

With respect to the feature selection process, the only aspect that needs to be defined is how many features are provided. Specifically, when we consider the genetic algorithm, this could be interesting. There may be terms that are not very useful when it comes to differentiating between relevant and irrelevant documents amongst those returned by the seed query. However these terms, in the wider collection, might be very useful. Providing the genetic algorithm with too few terms could deprive our solution of useful terms.

As such, figure 5.1 shows the average change in recall between the seed query and the queries generated by both the Rocchio algorithm and the genetic algorithm, ac-

cording to the number of features they were provided with. The number of features ranges from 25 to 300.

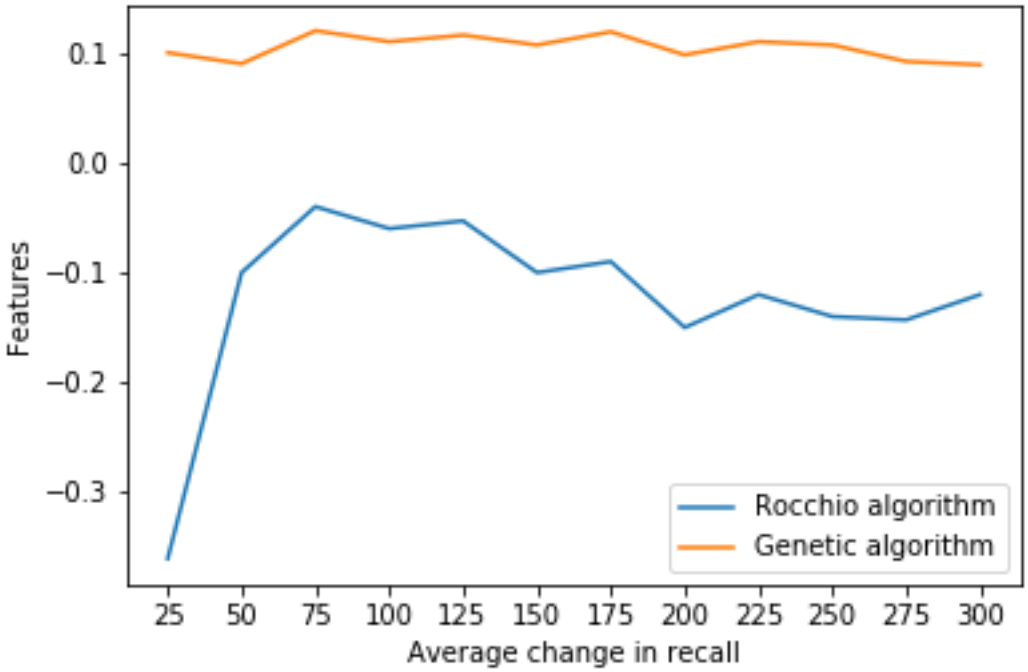

Figure 5.1: Increase in recall compared to seed query, according to number of features

As far as the genetic algorithm is concerned, the best number of features is 75, although the number of features doesn't effect recall significantly. However it is in our best interests to pick the smallest number of features since the larger the number of features the longer the algorithm takes to run.

It would seem that the best number of features with respect to the Rocchio algorithm is also around 75. However what is striking about these results is how poorly the Rocchio algorithm is performing, consistently retrieving fewer results than the seed query. This is concerning and explored further in the next section.

5.1.2 Rocchio Algorithm

Mindful of the appauling results obtained when trying to optimize the number of features to use, we can look to how to improve the results of the Rocchio algorithm by changing the parameters. The first way to optimize the algorithm is by altering the three constants; a, b and c. When trying to optimize the number of features, the parameters a, b and c had been set to 1, 0.8 and 0.2 respectively but we will now explore whether it would prove effective to change them. Different values were tested for these three variables using values between 0 and 1 in increments of 0.2. They are evaluated using the average increase in recall between the query returned by the Rocchio algorithm and the seed query (the average o 5 queries out of the 10, for efficiency). However unfortunately none of the combinations generated solutions that performed better that the initial query. These results were abysmal and have not been reported in this paper.

At this point, to try and remedy this, the threshold parameter for term selection was experimented with. The Rocchio algorithm returns a query vector and initially the optimized query was built by taking the associated term of any vector element above 0. However, with the Rocchio algorithm performing so poorly and the proposed queries being fairly long, it was conjectured that perhaps altering this threshold could help. Different threshold values were tested between 0 and 1 in increments of 0.1, and between 1 and 5 in increments of 1. The results are in tables 5.1 and 5.2 where we see the average increase in recall compared to the seed query. We can see that when the threshold takes larger values there is no increase. This means that the query vectors proposed were empty and the system defaulted to the initial seed query. However for threshold values around 0.5, the recall and net increases in retrieved results are improved. As such the threshold was set to 0.5 for the Rocchio algorithm.

Table 5.1: Testing the threshold for term selection in Rocchio algorithm (average over 5 queries)

Threshold	Recall increase	Net increase
0.0	-0.076	-3.5
0.1	-0.078	-2.9
0.2	-0.062	-1.8
0.3	-0.023	0.0
0.4	0.043	2.9
0.5	0.061	2.9
0.6	0.045	2.1
0.7	0.005	0.2
0.8	0	0
0.9	0	0
1.0	-0.033	-0.9
2.0	0	0
3.0	0	0
4.0	0	0
5.0	0	0
10.0	0	0

Table 5.2: Testing the threshold for term selection in Rocchio algorithm (average over 10 queries)

Threshold	Recall increase	Net increase
0.0	-0.043	-2.05
0.1	-0.032	-1.52
0.2	-0.020	-0.95
0.3	0.005	0.23
0.4	0.010	475
0.5	0.236	11.3
0.6	0.160	7.6
0.7	0.015	0.71
0.8	0	0
0.9	0	0
1.0	-0.023	-1.1
2.0	0	0
3.0	0	0
4.0	0	0
5.0	0	0
10.0	0	0

After selecting this 0.5 threshold, the testing of a, b and c parameters was performed again and the results table is in the Appendix. This time the parameters give more of a range of results and crucially in many cases the Rocchio algorithm was an improvement compared to the seed query. The best weights found were a = 1, b = 0.8 and c = 0.2, coincidental what had been chosen at the beginning! These parameters are used for the implementation of the Rocchio algorithm. The table in the appendix shows only the average results ove 5 queries (for efficiency's sake), this is why Table 5.1 features in this report, so the values can be compared.

5.1.3 Genetic Algorithm

There are a few components of the genetic algorithm that can be optimized. The size of the initial population, the number of generations to go through, and how the population is initialized in the first place. We look first at the number of individuals in a population at the start. Figure 5.3 shows the recall and net number of returned documents for initial population sizes of 50, 100 and 150, 200, 250 and 300.

Table 5.3: Testing the initial population size in a genetic algorithm

Population Size	Recall	Net increase
50	0.08	3.5
100	0.12	5.3
150	0.10	4.5
200	0.09	3.9
250	0.08	3.4
300	0.09	3.9

We can see that a population size of 100 yields the best recall and net retrieval on average across the 10 test queries.

In a similar way, we can test the number of generations the algorithm will go through before stopping and returning the best solution. Figure 5.4 is similar to figure 5.3 and we can see that the best number of generations appears to be 150. However it seems

that the results begin to plateau after 100, so we will use 100 generations, as it will be more efficient to keep the generations at a lower number for runtime's sake.

Table 5.4: Testing the initial population size in a genetic algorithm

Population Size	Recall	Net increase
50	0.09	3.9
100	0.12	5.2
150	0.13	5.7
200	0.12	5.2
250	0.12	5.1
300	0.13	5.7

Finally, it was noted that a lot of the final queries were considerably long, similarly to the Rocchio algorithm, so we tested using a bias when initializing the population. Instead of initializing individuals in the population with a random assortment of 1's and 0's, we tested making it more likely that the elements be 0's than 1's. It was found that on average across the 10 queries we were testing, having a greater bias improved recall considerably. The results can be seen in Figure 5.2.

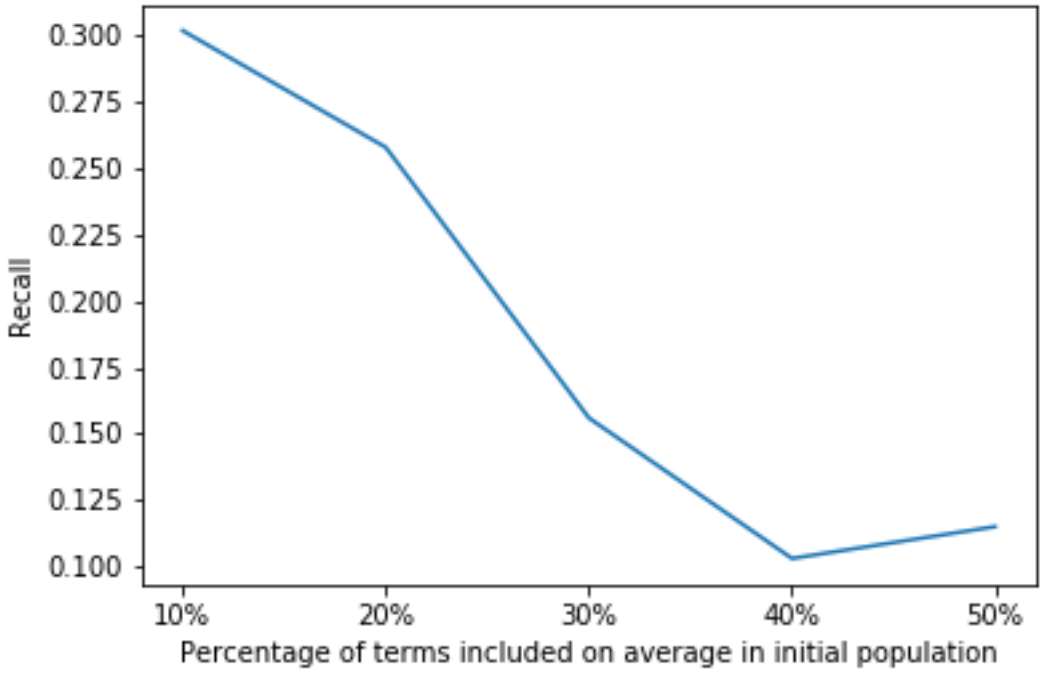

Figure 5.2: Recall according to the bias during the population initialization

5.2 Results

The best parameters found for the genetic algorithm are as follows: an initial population size of 100, 100 generations to be bred before returning a solution and a bias when generating the initial population whereby the individuals will have on average 10% of all terms in the vocabulary. Concerning the Rocchio algorithm, there is a threshold for term selection of 0.5, and the variables a, b and c are set to 1, 0.8 and 0.2 respectively. Finally, the best number of features to be selected in both algorithmic cases is 75.

The results outlining the performance of the two algorithms and the seed query are detailed in the following three tables. The final number of relevant documents returned per query by basic retrieval, the Rocchio query optimization and the genetic algorithm are in table 5.5. Table 5.6 shows the recall, per query, on each of these for each of the

three query types. Finally table 5.7 shows the increase in recall for both the query expansion techniques compared to the initial query.

Table 5.5: Raw number of documents returned for each query

Query n^o	No expansion	Rocchio expansion	Genetic algorithm
5	16	21	24
12	8	8	7
15	13	15	17
19	2	3	8
48	6	11	9
58	7	8	13
104	4	5	8
136	13	13	21
153	17	17	20
157	13	24	27
average	9.9	12.5	15.4

Table 5.6: Recall for each query

Query n^o	No expansion	Rocchio expansion	Genetic algorithm
5	0.59	0.77	0.88
12	0.24	0.24	0.21
15	0.76	0.88	1
19	0.05	0.07	0.2
48	0.07	0.13	0.10
58	0.21	0.23	0.38
104	0.05	0.06	0.09
136	0.11	0.11	0.18
153	0.22	0.22	0.27
157	0.09	0.16	0.18
average	0.24	0.29	0.35

Table 5.7: Increase in recall compared to initial query

Query n^o	Rocchio expansion	Genetic algorithm
5	30.5 %	49.1 %
12	0 %	-12.5 %
15	15.8 %	31.6 %
19	40 %	300 %
48	53.8 %	42.8 %
58	9.5 %	80.9 %
104	20 %	80 %
136	0 %	63.6 %
153	0 %	22.7 %
157	77.7 %	100 %
average	24.7%	75.8%

Table 5.5 shows that in almost all of the queries the genetic algorithm returns the most relevant documents. The same is seen for recall in table 5.6. These results are very promising, in particular the difference in recall shows how much the query optimization techniques are working and returning more relevant documents.

It can said that the actual quantity of relevant documents being returned is not particularly high, which is a fair criticism. Having said this, if we look at the increase in recall in figure 5.7 we can see that compared the number of relevant documents being returned by the seed queries, the optimized queries are functioning very effectively, in particular those optimized via the genetic algorithm.

Chapter 6

Conclusion

6.1 Final Results

The final results are very promising. Both query expansion techniques yield improved retrieval results, with the genetic algorithm proving the most effective. The success of the implementation of the genetic algorithm allows us to confirm the first hypothesis ("A genetic algorithm can be used as a tool for query optimization"). The fact that the genetic algorithm returned more relevant documents and better recall that the Rocchio algorithm and the seed query allow us to confirm the second hypothesis ("A genetic algorithm method of query expansion will perform better than other query expansion methods").

The third hypothesis ("Query optimization can be effectively performed using a genetic algorithm with-out input from the information retrieval system itself"), sought to explore whether a genetic algorithm query optimization technique would be effective when applied to an information retrieval system which modeled a search engine, where you have no access to the back end mechanics. This hypothesis can also be confirmed due to the combination of the particular experimental design used and the successful results produced for the genetic algorithm implementation.

6.2 Future Work

Despite the positive results there is much future work for this system, existing in two broad categories. The first category is all the ways in which this particular system could be further optimized and tested, and the second category is applications for these kinds of techniques in the future.

Starting with the former, the first techniques that could be explored are novel ways of handling the terms that are used by the query expansion techniques. For example, perhaps adding synonyms of important terms to the list of features could be useful. This would explicitly tackle the problem of synonymy in the query formulation process. In addition to this, an implementation could be designed when additional terms are added to the vocabulary as more documents are discovered. This would be more of an iterative query expansion technique but intuitively it would seem this could be an effective method. There would be other ways to apply genetic algorithms to this field as well. In the paper by Araujo et al. (2008) a genetic algorithm is used to alter the query using a morphological thesaurus which means that the query is altered by using words with the same root as those in the original query.

There are also different ways to handle the vocabulary in general that could be explored. For example innovative ways of stemming terms (Araujo & Pérez-Agüera 2008). It might also be worth experimenting with different feature selection methods, either more complex methods to test how they perform, or choosing ones specifically adapted for information retrieval tasks citepgeng2007feature. A more complicated model specifically tailored to query searching feature selection might help further improve the recall results.

If we were to test new feature selection models, it might be worth testing another query expansion optimization techniques too. In particular since the Rocchio algo-

rithm proved to be a bit temperamental when optimizing the parameters, future work could explore more comparison techniques.

In this implementation, the parameters optimized were done so for all the queries, taking the best parameters on average. However it is arguable that this should be done on a query by query basis. Especially given that the application would likely be to specific themes or domains, and that a few of the see queries were not improved by the genetic algorithm. Optimizing the parameters for each query would most likely generate better accuracy, however there is something to be said for creating a generalized model.

Another way to make the implementation here more robust would be to use more divers evaluation tools. For example, it might be useful to have some kind of measure of how well ranked the documents returned are. For example the system could be returning lots of relevant documents but if they are all on the second page of the search engine interface, the user is very unlikely to see them. To tackle this we could use recall at different ranks instead of just overall recall. Or indeed institute mechanisms specifically to improve the ranking, like, for example, a genetic algorithm!

However, it can be argued there are two major issues with the implementation that would need to be addressed first. The biggest problem with this implementation, and the first aspect that would need to be addressed in future work, is the runtime. The genetic algorithm in particularly is slow to run. Arguably this isn't too major of an issue since speed is not essential at this point, and even in applications we are probably not facing a scenario where the query is needed immediately. However it is worth baring in mind. Additionally the faster the runtime the easier to test and improve all other parameters. The second issue is that the basic retrieval system is not very good, there are still very small number of document's being returned by the seed query. Improving this system would allow us to determine the effect on the other techniques since they

depend on the initial documents returned.

In conclusion of this paper, we will touch upon the possible future applications of the techniques developed over the course of this research paper. As has been mentioned, the query expansion techniques could be used as part of bigger applications, to develop systems that would search through the web, using a search engine of the user's choice, collecting information on a topic a user wants to research. In fact, so long as you had some kind of classifier able to identify relevant web pages, any topic could be researched.

We can think back to the example in chapter 1 about collecting information about social innovation projects and the paper by Milosevic et al. (2018). If you wanted to build a system that used a search engine to identify web pages about social innovation projects, even though those projects might be very different, you could use a genetic algorithm to evolve towards a query that would return the most relevant pages.

In fact it could be interesting to incorporate some of the other technologies mentioned in the introduction that have been developed as a result of the desire to harness some of the abundant information online. In fact, this has already been done for the example above. Milosevic et al. used natural language processing techniques to extract information from different web pages about social innovation projects, they also used internet crawlers to crawl through web domains collecting web pages and classifying them to see if they were relevant to their research topic. If you could pair that technology with research via a search engine, you could use query expansion to find other relevant web pages and collect more information about the topic you want to research. In fact, to go further you could not only find other relevant web pages but potentially find web pages that simply contain a lot of links to the types of projects you are interested in. In a case like that it might be difficult for a user to construct a query that would characterize those kinds of pages, however a classifier would be

able to identify those pages and the genetic algorithm could evolve a query to return more of them. This scenario would be extremely relevant to the type of query expansion we have been discussing, as the user wouldn't need to understand anything about what such web pages would look like or contain, the query expansion would be able to evolve to a query that would return the web pages with the most links, thus maximizing the amount of information you can collect for the purposes of your research.

In conclusion, the research done in this paper allowed us to confirm our hypotheses about using genetic algorithms to optimize queries in information retrieval systems. Indeed it is possible to return more relevant documents when using a query optimized using a genetic algorithm. In fact this is a more efficient technique than using the Rocchio query expansion technique, a tried and tested method. Additionally the design of our system is such that these techniques could be applied to any information retrieval system or search engine, as it does not depend on any information about the workings of the retrieval system or the broader document collection. This means there are many practical applications for these techniques to collecting data and information on topics using the web.

Bibliography

Abdelmgeid, A. (2007), 'Applying genetic algorithm in query improvement problem', *Information Technologies and Knowledge* **1**(4).

Al Mashagba, E., Al Mashagba, F. & Nassar, M. O. (2011), 'Query optimization using genetic algorithms in the vector space model', *International Journal of Computer Science Issues (IJCSI)* **8**(5), 450.

Araujo, L. & Pérez-Agüera, J. R. (2008), Improving query expansion with stemming terms: a new genetic algorithm approach, *in* 'European Conference on Evolutionary Computation in Combinatorial Optimization', Springer, pp. 182–193.

Carpineto, C. & Romano, G. (2012), 'A survey of automatic query expansion in information retrieval', *Acm Computing Surveys (CSUR)* **44**(1), 1.

Chen, H. (1995), 'Machine learning for information retrieval: neural networks, symbolic learning, and genetic algorithms', *Journal of the American society for Information Science* **46**(3), 194–216.

Chen, H., Shankaranarayanan, G., She, L. & Iyer, A. (1998), 'A machine learning approach to inductive query by examples: an experiment using relevance feedback, id3, genetic algorithms, and simulated annealing', *Journal of the American Society for Information Science* **49**(8), 693–705.

Cui, H., Wen, J.-R., Nie, J.-Y. & Ma, W.-Y. (2003), 'Query expansion by mining user logs', *IEEE transactions on knowledge and data engineering* **15**(4), 829–839.

Cuna Ekmekcioglu, F., Robertson, A. M. & Willett, P. (1992), 'Effectiveness of query expansion in ranked-output document retrieval systems', *Journal of Information Science* **18**(2), 139–147.

Davis, M. (2019), 'Unicode text segmentation', `https://unicode.org/reports/tr29/`. Accessed: 31 July 2019.

Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T. K. & Harshman, R. (1990), 'Indexing by latent semantic analysis', *Journal of the American society for information science* **41**(6), 391–407.

Geng, X., Liu, T.-Y., Qin, T. & Li, H. (2007), Feature selection for ranking, *in* 'Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval', ACM, pp. 407–414.

Glover, E. J., Flake, G. W., Lawrence, S., Birmingham, W. P., Kruger, A., Giles, C. L. & Pennock, D. M. (2001), Improving category specific web search by learning query modifications, *in* 'Proceedings 2001 Symposium on Applications and the Internet', IEEE, pp. 23–32.

Haiduc, S., Bavota, G., Marcus, A., Oliveto, R., De Lucia, A. & Menzies, T. (2013), Automatic query reformulations for text retrieval in software engineering, *in* 'Proceedings of the 2013 International Conference on Software Engineering', IEEE Press, pp. 842–851.

Horng, J.-T. & Yeh, C.-C. (2000), 'Applying genetic algorithms to query optimization in document retrieval', *Information processing & management* **36**(5), 737–759.

Ingber, L. & Rosen, B. (1992), 'Genetic algorithms and very fast simulated reannealing: A comparison', *Mathematical and computer modelling* **16**(11), 87–100.

Kraft, D. H., Petry, F. E., Buckles, B. P. & SADASIVAN, T. (1997), Genetic algorithms for query optimization in information retrieval: relevance feedback, *in* 'Genetic al-

gorithms and fuzzy logic systems: Soft computing perspectives', World Scientific, pp. 155–173.

Kramer, O. (2017), *Genetic algorithm essentials*, Vol. 679, Springer.

Lei, S. (2012), A feature selection method based on information gain and genetic algorithm, *in* '2012 International Conference on Computer Science and Electronics Engineering', Vol. 2, IEEE, pp. 355–358.

Lewis, D. D., Schapire, R. E., Callan, J. P. & Papka, R. (1996), Training algorithms for linear text classifiers, *in* 'SIGIR', Vol. 96, pp. 298–306.

Lucene (2000), 'Apache lucene', `https://lucene.apache.org/`. Accessed: 8 August 2019.

Manning, C., Raghavan, P. & Schütze, H. (2010), 'Introduction to information retrieval', *Natural Language Engineering* **16**(1), 100–103.

McCue, T. (2018), 'Seo industry approaching $80 billion but all you want is more web traffic', `https://www.forbes.com/sites/tjmccue/2018/07/30/seo-industry-approaching-80-billion-but-all-you-want-is-more-web-traffic/#556a7ec47337`. Accessed: 31 July 2019.

Milosevic, N., Gok, A. & Nenadic, G. (2018), Classification of intangible social innovation concepts, *in* 'International Conference on Applications of Natural Language to Information Systems', Springer, pp. 407–418.

Ng, H. T., Goh, W. B. & Low, K. L. (1997), Feature selection, perceptron learning, and a usability case study for text categorization, *in* 'SIGIR', Vol. 97, pp. 67–73.

NIST (2000), 'Trec overview', `https://trec.nist.gov/overview.html`. Accessed: 8 August 2019.

NodeGraph (2017), 'Big data facts – how much data is out there?', `https://www.nodegraph.se/big-data-facts/`. Accessed: 25 August 2019.

Pathak, P., Gordon, M. & Fan, W. (2000), Effective information retrieval using genetic algorithms based matching functions adaptation, *in* 'Proceedings of the 33rd annual Hawaii international conference on system sciences', IEEE, pp. 8–pp.

Qiu, Y. & Frei, H.-P. (1993), Concept based query expansion, *in* 'Proceedings of the 16th annual international ACM SIGIR conference on Research and development in information retrieval', ACM, pp. 160–169.

Robertson, S. E. (1990), 'On term selection for query expansion', *Journal of documentation* **46**(4), 359–364.

Rocchio, J. J. (1971), 'The smart retrieval system: Experiments in automatic document processing', *Relevance feedback in information retrieval* pp. 313–323.

Seymour, T., Frantsvog, D. & Kumar, S. (2011), 'History of search engines', *International Journal of Management & Information Systems (IJMIS)* **15**(4), 47–58.

Singhal, A. et al. (2001), 'Modern information retrieval: A brief overview', *IEEE Data Eng. Bull.* **24**(4), 35–43.

Ted (2009), 'Tim berners-lee: The next web of open, linked data', `https://www.youtube.com/watch?v=OM6XIICm_qo&t=338s`. Accessed: 31 July 2019.

Tutorialspoint (2019), 'Lucene - overview', `https://www.tutorialspoint.com/lucene/lucene_overview.htm`. Accessed: 05 August 2019.

Uğuz, H. (2011), 'A two-stage feature selection method for text categorization by using information gain, principal component analysis and genetic algorithm', *Knowledge-Based Systems* **24**(7), 1024–1032.

Vrajitoru, D. (1998), 'Crossover improvement for the genetic algorithm in information retrieval', *Information processing & management* **34**(4), 405–415.

Yang, Y. & Pedersen, J. O. (1997), A comparative study on feature selection in text categorization, *in* 'Icml', Vol. 97, p. 35.

Appendix

Below is a table from the results section which was too large to include in the main text body. It contains the results from an exploration of the a, b, and c parameters of the Rocchio Algorithm implementation.

a	b	c	Recall increase	Net retrieval increase
0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.2	0.0	0.0
0.0	0.0	0.4	0.0	0.0
0.0	0.0	0.6	0.0	0.0
0.0	0.0	0.8	0.0	0.0
0.0	0.0	1.0	0.0	0.0
0.0	0.2	0.0	0.0	0.0
0.0	0.2	0.2	0.0	0.0
0.0	0.2	0.4	0.0	0.0
0.0	0.2	0.6	0.0	0.0
0.0	0.2	0.8	0.0	0.0
0.0	0.2	1.0	0.0	0.0
0.0	0.4	0.0	0.0	0.0
0.0	0.4	0.2	0.0	0.0
0.0	0.4	0.4	0.0	0.0
0.0	0.4	0.6	0.0	0.0
0.0	0.4	0.8	0.0	0.0
0.0	0.4	1.0	0.0	0.0
0.0	0.6	0.0	4.889073317840178E-4	0.9
0.0	0.6	0.2	0.0	0.0
0.0	0.6	0.4	0.0	0.0
0.0	0.6	0.6	0.0	0.0
0.0	0.6	0.8	0.0	0.0
0.0	0.6	1.0	0.0	0.0
0.0	0.8	0.0	0.03696202888014496	2.1
0.0	0.8	0.2	0.020546160207094346	1.3
0.0	0.8	0.4	-0.12624641775902282	-3.6

0.0	0.8	0.6	-0.016827094474153297	-0.8
0.0	0.8	0.8	-0.004705882352941176	-0.4
0.0	0.8	1.0	0.0	0.0
0.0	1.0	0.0	0.008638120901217081	1.9
0.0	1.0	0.2	0.013509810313601595	1.7
0.0	1.0	0.4	0.007627457372425125	1.4
0.0	1.0	0.6	-0.0979023669604088	-3.4
0.0	1.0	0.8	-0.12799440377513466	-4.0
0.0	1.0	1.0	-0.15953368615475316	-4.3
0.2	0.0	0.0	0.0	0.0
0.2	0.0	0.2	0.0	0.0
0.2	0.0	0.4	0.0	0.0
0.2	0.0	0.6	0.0	0.0
0.2	0.0	0.8	0.0	0.0
0.2	0.0	1.0	0.0	0.0
0.2	0.2	0.0	0.0	0.0
0.2	0.2	0.2	0.0	0.0
0.2	0.2	0.4	0.0	0.0
0.2	0.2	0.6	0.0	0.0
0.2	0.2	0.8	0.0	0.0
0.2	0.2	1.0	0.0	0.0
0.2	0.4	0.0	-0.037068357998590554	-0.9
0.2	0.4	0.2	0.0	0.0
0.2	0.4	0.4	0.0	0.0
0.2	0.4	0.6	0.0	0.0
0.2	0.4	0.8	0.0	0.0
0.2	0.4	1.0	0.0	0.0
0.2	0.6	0.0	0.010284538057647303	1.1

0.2	0.6	0.2	-0.032068357998590556	-0.7
0.2	0.6	0.4	0.0	0.0
0.2	0.6	0.6	0.0	0.0
0.2	0.6	0.8	0.0	0.0
0.2	0.6	1.0	0.0	0.0
0.2	0.8	0.0	0.05790677041312179	2.8
0.2	0.8	0.2	0.04908324100135708	2.5
0.2	0.8	0.4	0.011415944813443348	1.1
0.2	0.8	0.6	-0.016827094474153297	-0.8
0.2	0.8	0.8	-0.004705882352941176	-0.4
0.2	0.8	1.0	0.0	0.0
0.2	1.0	0.0	0.017461650312981785	2.2
0.2	1.0	0.2	0.027251882195208365	2.4
0.2	1.0	0.4	0.019392163254778067	1.8
0.2	1.0	0.6	0.013812153473087613	1.1
0.2	1.0	0.8	-0.12799440377513466	-4.0
0.2	1.0	1.0	-0.15953368615475316	-4.3
0.4	0.0	0.0	0.0	0.0
0.4	0.0	0.2	0.0	0.0
0.4	0.0	0.4	0.0	0.0
0.4	0.0	0.6	0.0	0.0
0.4	0.0	0.8	0.0	0.0
0.4	0.0	1.0	0.0	0.0
0.4	0.2	0.0	-0.03333333333333333	-0.9
0.4	0.2	0.2	0.0	0.0
0.4	0.2	0.4	0.0	0.0
0.4	0.2	0.6	0.0	0.0
0.4	0.2	0.8	0.0	0.0

0.4	0.2	1.0	0.0	0.0
0.4	0.4	0.0	-0.03333333333333333	-0.9
0.4	0.4	0.2	-0.037068357998590554	-0.9
0.4	0.4	0.4	0.0	0.0
0.4	0.4	0.6	0.0	0.0
0.4	0.4	0.8	0.0	0.0
0.4	0.4	1.0	0.0	0.0
0.4	0.6	0.0	0.01910806746941201	1.4
0.4	0.6	0.2	-0.028333333333333332	-0.7
0.4	0.6	0.4	-0.032068357998590556	-0.7
0.4	0.6	0.6	0.0	0.0
0.4	0.6	0.8	0.0	0.0
0.4	0.6	1.0	0.0	0.0
0.4	0.8	0.0	0.05790677041312179	2.8
0.4	0.8	0.2	0.05790677041312179	2.8
0.4	0.8	0.4	0.02121157553930663	1.3
0.4	0.8	0.6	-0.03392219044065829	-0.7
0.4	0.8	0.8	-0.004705882352941176	-0.4
0.4	0.8	1.0	0.0	0.0
0.4	1.0	0.0	0.017461650312981785	2.2
0.4	1.0	0.2	0.03607541160697307	2.7
0.4	1.0	0.4	0.04195776454814954	2.8
0.4	1.0	0.6	0.034667768446349646	1.8
0.4	1.0	0.8	0.022037665900280716	1.3
0.4	1.0	1.0	-0.15953368615475316	-4.3
0.6	0.0	0.0	0.0	0.0
0.6	0.0	0.2	-0.06623600344530577	-2.2
0.6	0.0	0.4	-0.06623600344530577	-2.2

0.6	0.0	0.6	-0.06623600344530577	-2.2
0.6	0.0	0.8	-0.06623600344530577	-2.2
0.6	0.0	1.0	-0.06623600344530577	-2.2
0.6	0.2	0.0	0.0	0.0
0.6	0.2	0.2	0.0	0.0
0.6	0.2	0.4	-0.06623600344530577	-2.2
0.6	0.2	0.6	-0.06623600344530577	-2.2
0.6	0.2	0.8	-0.06623600344530577	-2.2
0.6	0.2	1.0	-0.06623600344530577	-2.2
0.6	0.4	0.0	0.0	0.0
0.6	0.4	0.2	0.0	0.0
0.6	0.4	0.4	0.002325581395348837	0.2
0.6	0.4	0.6	-0.06623600344530577	-2.2
0.6	0.4	0.8	-0.06623600344530577	-2.2
0.6	0.4	1.0	-0.06623600344530577	-2.2
0.6	0.6	0.0	0.04503399339533794	2.1
0.6	0.6	0.2	0.005	0.2
0.6	0.6	0.4	0.005	0.2
0.6	0.6	0.6	0.0012649753347427767	0.2
0.6	0.6	0.8	-0.06623600344530577	-2.2
0.6	0.6	1.0	-0.06623600344530577	-2.2
0.6	0.8	0.0	0.061610474116825485	2.9
0.6	0.8	0.2	0.061610474116825485	2.9
0.6	0.8	0.4	0.04114621606218245	1.9
0.6	0.8	0.6	0.009206773618538324	0.4
0.6	0.8	0.8	0.0024414459229780707	0.3
0.6	0.8	1.0	-0.06623600344530577	-2.2
0.6	1.0	0.0	0.024869057720389193	2.4

0.6	1.0	0.2	0.043482819014380475	2.9
0.6	1.0	0.4	0.049365171955556945	3.0
0.6	1.0	0.6	0.05184616435251573	2.6
0.6	1.0	0.8	0.03483227869489351	1.7
0.6	1.0	1.0	0.014390162454457938	0.4
0.8	0.0	0.0	0.0	0.0
0.8	0.0	0.2	0.0	0.0
0.8	0.0	0.4	-0.06623600344530577	-2.2
0.8	0.0	0.6	-0.06623600344530577	-2.2
0.8	0.0	0.8	-0.06623600344530577	-2.2
0.8	0.0	1.0	-0.06623600344530577	-2.2
0.8	0.2	0.0	0.0	0.0
0.8	0.2	0.2	0.0	0.0
0.8	0.2	0.4	0.0	0.0
0.8	0.2	0.6	-0.06623600344530577	-2.2
0.8	0.2	0.8	-0.06623600344530577	-2.2
0.8	0.2	1.0	-0.06623600344530577	-2.2
0.8	0.4	0.0	0.0	0.0
0.8	0.4	0.2	0.0	0.0
0.8	0.4	0.4	0.0	0.0
0.8	0.4	0.6	0.002325581395348837	0.2
0.8	0.4	0.8	-0.06623600344530577	-2.2
0.8	0.4	1.0	-0.06623600344530577	-2.2
0.8	0.6	0.0	0.04503399339533794	2.1
0.8	0.6	0.2	0.005	0.2
0.8	0.6	0.4	0.005	0.2
0.8	0.6	0.6	0.005	0.2
0.8	0.6	0.8	-0.0037350246652572234	0.0

0.8	0.6	1.0	-0.06623600344530577	-2.2
0.8	0.8	0.0	0.061610474116825485	2.9
0.8	0.8	0.2	0.061610474116825485	2.9
0.8	0.8	0.4	0.04114621606218245	1.9
0.8	0.8	0.6	0.009206773618538324	0.4
0.8	0.8	0.8	0.0061764705882352946	0.3
0.8	0.8	1.0	0.0012649753347427767	0.2
0.8	1.0	0.0	0.024869057720389193	2.4
0.8	1.0	0.2	0.043482819014380475	2.9
0.8	1.0	0.4	0.049365171955556945	3.0
0.8	1.0	0.6	0.05184616435251573	2.6
0.8	1.0	0.8	0.04830697089735589	2.4
0.8	1.0	1.0	0.02348107154536703	0.7
1.0	0.0	0.0	0.0	0.0
1.0	0.0	0.2	0.0	0.0
1.0	0.0	0.4	0.0	0.0
1.0	0.0	0.6	-0.08682423873942342	-2.9
1.0	0.0	0.8	-0.06623600344530577	-2.2
1.0	0.0	1.0	-0.06623600344530577	-2.2
1.0	0.2	0.0	0.0	0.0
1.0	0.2	0.2	0.0	0.0
1.0	0.2	0.4	0.0	0.0
1.0	0.2	0.6	0.0	0.0
1.0	0.2	0.8	-0.06623600344530577	-2.2
1.0	0.2	1.0	-0.06623600344530577	-2.2
1.0	0.4	0.0	0.0	0.0
1.0	0.4	0.2	0.0	0.0
1.0	0.4	0.4	0.0	0.0

1.0	0.4	0.6	0.0	0.0
1.0	0.4	0.8	0.002325581395348837	0.2
1.0	0.4	1.0	-0.06623600344530577	-2.2
1.0	0.6	0.0	0.04503399339533794	2.1
1.0	0.6	0.2	0.005	0.2
1.0	0.6	0.4	0.005	0.2
1.0	0.6	0.6	0.005	0.2
1.0	0.6	0.8	0.0	0.0
1.0	0.6	1.0	0.004825581395348838	0.3
1.0	0.8	0.0	0.061610474116825485	2.9
1.0	0.8	0.2	0.061610474116825485	2.9
1.0	0.8	0.4	0.04114621606218245	1.9
1.0	0.8	0.6	0.009206773618538324	0.4
1.0	0.8	0.8	0.0061764705882352946	0.3
1.0	0.8	1.0	0.005	0.2
1.0	1.0	0.0	0.024869057720389193	2.4
1.0	1.0	0.2	0.043482819014380475	2.9
1.0	1.0	0.4	0.049365171955556945	3.0
1.0	1.0	0.6	0.05184616435251573	2.6
1.0	1.0	0.8	0.04830697089735589	2.4
1.0	1.0	1.0	0.028132234336064703	1.1