RankGPES: Learning to Rank for Information Retrieval using a Hybrid Genetic Programming with Evolutionary Strategies

by

Mohammad Ashiful Islam

B.Sc. (Honours), University of Windsor, 2007

A thesis

presented to Ryerson University

in partial fulfillment of the

requirements for the degree of

Master of Science

in the Program of

Computer Science

Toronto, Canada, 2013

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I authorize Ryerson University to lend this thesis to other institutions or individuals for the purpose of scholarly research.

Signed:_____

I further authorize Ryerson University to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

I understand that my thesis may be made electronically available to the public.

Signed:_____

RankGPES: Learning to Rank for Information Retrieval using a Hybrid Genetic Programming with Evolutionary Strategies

Mohammad Ashiful Islam

M. Sc. in Computer Science, 2013

Ryerson University, Toronto, Canada

Abstract

In recent years, Learning to Rank has not only shown effectiveness and better suitability for modern Web Era needs, but also has proved that it outperforms traditional ranking in terms of accuracy and efficiency. Evolutionary approach to Learning to Rank such as RankGP [37] and RankDE [3] have shown further improvement over non-evolutionary algorithms. However when Evolutionary algorithms have been applied to a large volume of data, often they showed they required so much computational efforts that they were not worth applying to industrial applications. In this thesis, we present RankGPES: a Learning to Rank algorithm based on a hybrid approach combining Genetic Programming with Evolution Strategies. Our results not only showed that it outperformed both RankGP [37] by 20% and RankDE [3] by 6% in terms of accuracy but also it showed it required significant less amount of time to converge to a near-optimal result.

Acknowledgements

I would like to thank you my supervisor Dr. Marcus Santos and my co-supervisor Dr. Cherie Ding for their kind support throughout the program and Ryerson University to give me a chance to pursue my Masters Degree.

Dedication

I want to dedicate my work to my father who has always been on my side throughout my entire life and to my wife for being extremely supportive during this process.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

The central hypothesis of our research work is that, Hybrid Genetic Programming with Evolution Strategies can produce a competitive solution in Learning to Rank area while outperforming previous evolutionary based implementations such as RankGP [37] and RankDE [3] in terms of accuracy and efficiency.

1.1 Statement of the problem and objectives of the research

Learning to rank refers to a mechanism where a Machine Learning Algorithm is used to train a model using some training data, and then the trained model is applied to unseen data to rank it. More formally, it is a type of semi-supervised machine learning problem where the goal is to automatically construct a ranking model from training data [21]. In our model, this solution is achieved by using a technique from machine learning known as Genetic Programming [25]. Genetic programming attempts to search a program space for optimal solutions in a stochastic directed manner. This search is done in a manner motivated by Darwin's theory of evolution [33], in that the good programs are more likely to be combined to form new programs while poor solutions die off. Our model further enhances classical Genetic Programming model by utilizing the strong evolvability power of Evolutionary Strategies and using an optimal set of genetic programming parameters.

1.2 Motivation

Ranking is a very common problem in modern IR (Information Retrieval), NLP (Natural Language Processing) and Data Mining applications where it plays a vital role in serving data in a sorted order so that more relevant data is presented before less relevant one. Typical applications include: Web Search (Google, Yahoo, Bing), Recommender Systems (Netflix, Amazon, E-Bay), Social Media (Facebook, Twitter), Question-Answering (IBM Watson), Multimedia Information Retrieval (Google Image search, You tube), Information Organization (Text Categorization, Document Clustering), Machine Translation and so on. With the fast development of today's World Wide Web, we are experiencing a huge amount of data being created every second. According to worldwidewebsize.com, the indexed Web consists of at least 8.32 billion pages and growing exponentially. Besides data growth, our average data consumption rate is also increasing heavily. For instance, in 2008 Americans consumed information for about 1.3 trillion hours, an average of almost 12 hours per day. Consumption totaled 3.6 zetta bytes and 10,845 trillion words, corresponding to 100,500 words and 34 gigabytes for an average person on an average day. This large volume of data is also known as Big Data and unfortunately, our traditional ranking algorithms fail in many cases when it comes to process the Big Data.

Furthermore, it is not only the data volume that concerns today's Web applications but also increasing complexity of data or unstructured information. These issues lead to a creation of too many ranking features or parameters when it comes to ranking process. Failing to find out an optimal set of ranking parameters or a relation between them leads to irrelevant results. With traditional ranking approach, manual ranking parameter tuning is extremely difficult, especially when there are many complex ranking parameters and the evaluation measures are non-smooth. Manual ranking parameter tuning sometimes leads to over fitting and it is non-trivial to combine a large number of models to obtain an even more effective model [21]. To cope with these issues Machine Learning-based approaches have been preferred which have been successful in learning

optimal or near-optimal ranking function from training data.

Learning to Rank documents retrieved for a user query has gained much attention and several initiatives have taken place such as the Yahoo's Learning to Rank Challenge and Microsoft's LETOR project where a significant number of Machine Learning Algorithm have been proposed of which Evolutionary approaches such as RankGP [37] or RankDE [3] showed reasonable improvement over non-evolutionary algorithms. However, they not only require very long training/processing time but also need huge computational resources. Moreover, the work introduced in [37] did not get experimented with different combination of parameters of Genetic Programming and recommended a few algorithm optimization tasks as future tasks such as: experimenting with a better fitness function, use of different evolutionary parameters/setting or applying better evolutionary operators and applying to a different dataset. In our work, we had explored these areas not only by applying our model to an additional dataset (*i.e.* OHSUMED (Oregon Health and Science University Medical Dataset) dataset [23]) which enabled us to see how well the algorithm performs on different sets of data or whether the algorithm shows a similar pattern of result on different data sets or not, but also by experimenting with different parameters and discovered an enhanced parameter setting.

Other motivation to propose this Hybrid approach to Genetic Programming is: Evolutionary Strategies can help Genetic Programming to evolve aggressively towards a solution and reduce bloating problem (*i.e.,* generation of a lot of useless solutions that soon get deleted as they are not good enough to survive, therefore unnecessarily overloads the system) by not generating unnecessary candidates which could increase the likelihood of success in Genetic Programming problems [9].

Our proposed model not only concentrates on better accuracy by using an enhanced form of Genetic Programming but also focuses on efficiency for large scalability. We have experimented with NDCG (Normalized Discounted Cumulative Gain) [29] as our fitness function as a substitute of MAP (Mean Average Precision) [16] used by RankGP, which resulted interesting findings in our

experiments. Also, to create a broader search space, we have increased number of generations and to diversify evolution process, we utilized the selection methodology of Evolutionary Strategies [2]. Our additional parallel model enables to process the evolution task independently in parallel just like the evolutionary process occurs in nature which is a more natural solution than single population model.

1.3 Contribution and summary of results

In this work, we have presented a Machine Learned ranking algorithm based on a hybrid approach by combining Genetic Programming with Evolutionary Strategies. We have experimented with the algorithm with a few different settings such as varying initial population size, number of generations, probability of evolutionary operators, use of different fitness functions and using a parallel approach based on island model. We used LETOR 2.0 [36]: a benchmark dataset for learning to rank problem and compared against RankGP.

Analyzing the results from our experiments we found out that:

- For our problem, using a very large population size does not improve the accuracy of the algorithm but what improves is that a large number of generations.
- We discovered that a high probability of crossover and a very low probability of mutation show better results.
- Use of Evolutionary Strategies in the Genetic Programming Algorithm for Selection task adds more diversity to our solutions than classic Genetic Programming.
- NDCG as a fitness function showed higher accuracy than MAP in the best case scenarios (when training data and testing data are similar to each other), however in the worst cases (when training data and testing data are very different from each other) it resulted poorer solutions. When we averaged the result, MAP is still preferred over NDCG for higher accuracy for our dataset.

• Our parallel model showed consistent results in every run, but it created more bloating problem than single population run because of the particular migration scheme, where only the best candidates get migrated between islands and survive, that leads to a creation of more number of less diversified solutions that took huge time and computational resources to process which soon die off resulting a lot efforts get wasted unnecessarily.

• Out of the five algorithms we have experimented with our proposed algorithm tops in terms of both accuracy and efficiency. The proposed Algorithm also showed that, it is less likely to get trapped in local optima, rather it always moves towards a global optima. However it produced a solution with a complex structure and we have added some suggestion in our future work section that we think are worth investigating which may enhance solution further.

1.4 Structure of the thesis

This thesis is organized as follows:

• Chapter 2 discusses some background information about Evolutionary Computing and Learning to Rank. The chapter also talks about different types of Learning to Rank Algorithm as well as reviews previous research work that has been done in this area.

• Chapter 3 describes our proposed methodology for solving Learning to rank problem as well as the materials used in our work. It explains how the hybrid algorithm has been designed and implemented. Furthermore, it also demonstrates the algorithmic flow of the model.

• Chapter 4 is devoted to the performance evaluation of our solution approaches. The global and failure-oriented reconfiguration strategies are compared based on predefined performance metrics.

- Chapter 5 summarizes the overall contributions of our thesis while discussing possible future research directions.

Chapter 2

Background and Related Work

In the work presented here, we present a hybrid Genetic Programming with Evolutionary Strategies Algorithm to generate or discover a near-optimal ranking function automatically for Learning to Rank problem. In this chapter, we begin by providing an overview of background concepts, such as general Evolutionary Algorithms work and how the problem inputs are represented in Genetic Programming. We also provide details about Learning to Rank mechanism and different types of Learning to rank approaches. We conclude this chapter by reviewing research works related to this area.

2.1 Background

2.1.1 Evolutionary Computation

Evolutionary Computing is the collective name for a range of problem-solving methods based on Darwinian principles of biological evolution, such as natural selection and genetic inheritance. These techniques are being increasingly widely applied to a variety of problems, ranging from practical applications in the industry and commerce to leading-edge scientific research [13]. A typical flow of an Evolutionary algorithm is shown in Algorithm 1. Evolutionary Algorithms start with a set of initial populations called candidates, where each can candidate could be a potential solution. A fitness function plays a key role in Evolutionary Systems which measures how good a candidate is. New solutions are selected according to their fitness the more suitable they are the

more chances they have to reproduce. Through a number of generations, the candidates get evolved to improve their fitness by applying some evolutionary operations such as crossover, mutation, selection etc. and finally the best solution which has the best fitness measure is outputted after meeting some termination conditions.

Algorithm 1: A typical Evolutionary Algorithm

INPUT: Input Data, Evolutionary Parameters

OUTPUT: An optimal solution

1. Create an initial population of individuals
2. Evaluate the population of individuals using Fitness function

while *termination condition not reached* **do**

3. Select individuals using a selection operator.
4. Apply evolutionary operators to selected individuals to obtain offspring.
5. Evaluate offspring
6. Assimilate offspring in the population

7. Output the best result

2.1.2 Genetic Programming

Genetic Programming [25] is an algorithmic resolution of problems based on mechanisms observed on the nature and formalized on Darwin's Natural Selection Theory. In nature, the individuals that better adapt to the environment that surrounds them have a greater chance to survive. They pass their genetic characteristics to their descendants and consequently, after several generations, this process tends to select the best individuals naturally.

2.1.3 Elements of Genetic Programming

The main elements of Genetic Programming are:

2.1.3.1 Problem solution representation

A tree is the most used structure for represent programs in GP. Each node can be a function or a terminal. A function has to be evaluated considering its parameters while a terminal has its own value. The user needs to provide functions and terminals sets according to the problem.

2.1.3.2 Fitness

In the GP, the entity that reflects the degree of adaptation is the fitness function. The programs that better solve the problem at hand will receive a better fitness value, and will consequently have a better chance of being selected. An adequate choosing according to the domain of the problem is essential to provide good results.

2.1.3.3 Evolutionary Operators

Once the individuals are selected it is time to apply one of the three basic genetic operators. They are:

a. **Selection:** An individual is replicated to the next generation.

b. **Crossover:** Two programs are recombined to generate two offspring.

c. **Mutation:** New sub-tree replaces a randomly selected part of a program.

2.1.3.4 Parameters of Algorithmic run

The behavior of the algorithm is determined by a set of parameters that define and control how the search is performed. Some of them are: genetic operator's rate (reproduction, crossover, and mutation), population size and number of generations.

2.1.4 Common Genetic Programming Operators

2.1.4.1 Selection

Selection is an important part of any evolutionary algorithm. Without selection, directing the algorithm towards fitter solutions there would be no progress. There is no selection operator that is the best for all problems, but listed below are some of the most common methods. Some of these methods are mutually exclusive, while others are often used in combination:

a. **Elitist selection:** In Elitist selection- sometimes called just elitism, a specific number of most fit candidates goes from one generation to another unmodified. This can sometimes have a dramatic impact on performance by ensuring that the GP does not waste time re-discovering previously discarded partial solutions. Usually elitism is used as an additional selection operator.

b. **Fitness-proportionate selection:** In Fitness-proportionate selection gives every individual a chance of being selected to breed, but fitter candidates get more priority to be chosen than weaker individuals.

c. **Roulette-wheel selection:** The most common form of fitness-proportionate selection in which the chance of an individual's being selected is proportional to the amount by which its fitness is greater or less than its competitors' fitness using an imaginary roulette-wheel. Unlike a real roulette wheel, the sections are different sizes, proportional to the individual's fitness, such that the fittest candidate has the biggest slice of the wheel and the weakest candidate has the smallest. The wheel is then spun and the individual associated with the winning section is selected. The wheel is spun as many times as is necessary to select the full set of parents for the next generation. Using this technique it is possible that one or more individuals is selected multiple times.

d. **Tournament selection:** Tournament Selection is among the most widely used selection strate-

gies in evolutionary algorithms. Like a tournament, subgroups of individuals are randomly chosen from the larger population, and members of each subgroup compete against each other. Only one individual from each subgroup is chosen to reproduce.

e. **Rank selection:** Each individual in the population is assigned a numerical rank based on fitness, and selection is based on this ranking rather than absolute differences in fitness. The advantage of this method is that it can prevent very fit individuals from gaining dominance early at the expense of less fit ones, which would reduce the population's genetic diversity and might hinder attempts to find an acceptable solution [26].

2.1.4.2 Crossover

The goal of Crossover operation is to create diversified and potentially promising new chromosomes by combining genetic material from the parents. Some popular crossover methods include:

a. **Single-point Crossover:** Single-point crossover works by selecting one common crossover point in the parent candidates and then swapping the corresponding subtrees as shown in Figure 2.1.

b. **Multi-point Crossover:** There are two cases in this type of crossover. One is even number of cross-sites and the other odd number of cross-sites. In the case of an even number of cross-sites, cross-sites are selected randomly around a circle, and information is exchanged. In the case of an odd number of cross sites, a different cross-point is always assumed at the root node. Advantage of having more crossover points are that the problem space may be searched more thoroughly.

2.1.4.3 Mutation

Mutation prevents the algorithm to be trapped in a local minimum. Mutation plays the role of recovering the lost genetic materials as well as for randomly disturbing genetic information. Some

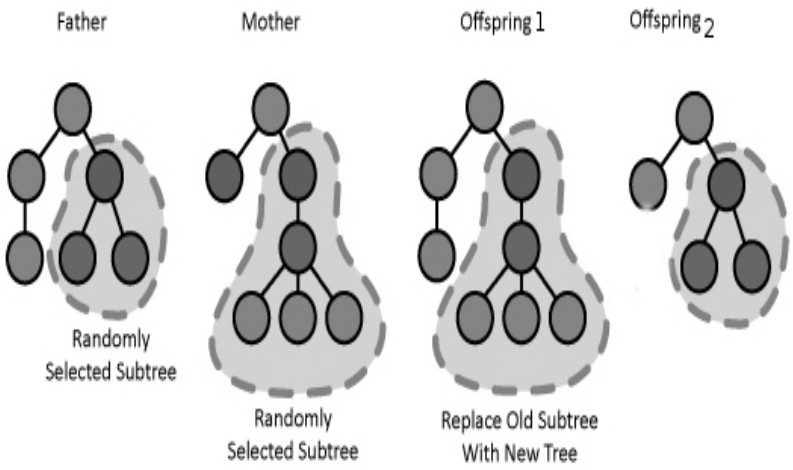

Figure 2.1: Crossover

popular methods of mutation include:

a. **Subtree mutation:** Like Crossover, subtree mutation works between an individual from the population and a newly generated individual. A mutation point is chosen and the subtree connected to that point is removed. It is then replaced with a newly generated subtree as shown in Figure2.2. The new subtree is usually constructed via the grow method, with a maximum depth near the depth of the tree being replaced.

b. **Shrink mutation:** Shrink mutation was created with the intent of reducing the complexity of populations. Shrink works by selecting a random mutation point and removing the subtree associated with that point. A terminal is then selected from the node factory and inserted into the empty space in the tree. Subtree mutation proves effective in reducing complexity, particularly by removing large subtrees of ineffective code.

c. **Node replacement mutation:** Node replacement mutation picks a mutation point from the individual. A new node with the same signature (arity, return type, etc.) is selected from the

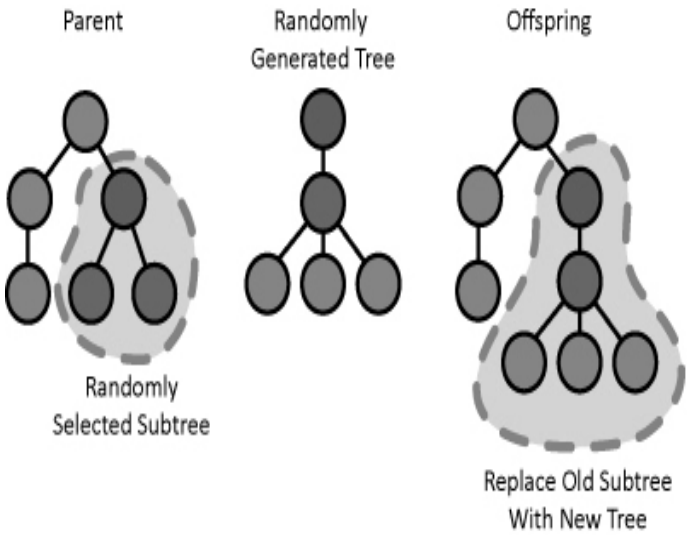

Figure 2.2: Mutation

node factory and the node at the mutation point is replaced with the new node [19]

2.1.5 Parallel Genetic Programming Models

Parallel Genetic Programming is an advanced form of Genetic Programming, where multiple genetic programs run in parallel for a number of generations, independent of each other, and at the end all the results are combined. Some popular Parallel Genetic Programming Implementation Models are: Island Model, Master-Slave Model and Grid Model [5]. In our work, the parallelism behavior of Genetic Programming is achieved by using asynchronous Island Model [27] where individuals are divided up into subpopulations (islands) which evolve in parallel asynchronously.

Each island act as an independent evolutionary algorithm. Individuals are periodically exchanged between the islands over the course of the GP run. An obvious advantage of having parallel model is that if one island prematurely converges on a sub-optimal solution it does not affect the evolution happening on the other islands; they follows their own paths. A single large population does not have this feature. The other benefit is that, each island can have completely different parameter settings such as different crossover/mutation probability, use of different selection operator or different evolution strategy which results more diverse solutions. The isolation of

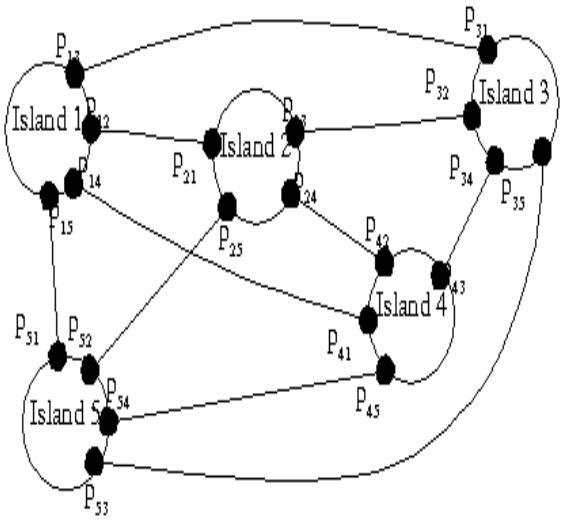

Figure 2.3: Genetic Programming Island Model

the separate populations often leads to different traits originating on different islands. Migration operation brings these diverse individuals together occasionally to see what happens when they are combined.

In summary, a parallel GP system has the following additional elements:

a. **Island:** In the island model, the population is partitioned into sub-populations to evolve independently, each of the partition is known as an island.

b. **Migration:** While evolving, a number of individuals moves from one island to another periodically which is known as migration. There are different types of migration topologies such as ring migration, grid migration, random migration etc.

c. **Epoch:** In each island, a sub-population evolves independently with respect to other islands for a number of iterations before migration occurs. Each iteration is called an epoch.

2.1.6 Evolutionary Strategies

Evolutionary Strategies are inspired by the theory of evolution by means of natural selection, mostly used as a sibling technique to other Evolutionary Algorithms such as Genetic Algorithms and Genetic Programming. It is an optimization technique created to optimize parameters by Ingo Rechenberg and Hans-Paul Schwefel [2], where the selection schema differs from typical Genetic Programing. The two most simple forms of ES are:

2.1.6.1 ($\mu + \lambda$) - ES

($\mu + \lambda$) - ES specifies that μ parents produce λ descendants, where $\lambda > \mu$. The descendants compete with their parents in the selection of the best μ individuals to the creation of the next generation. It is an elitist strategy.

2.1.6.2 (μ , λ) - ES

(μ , λ) - ES is very similar to ($\mu + \lambda$)-ES with the exception that only descendants survive and go through next generation. This strategy is more greedy than ($\mu + \lambda$) and it allows for more diversity in the population, thus avoiding the algorithm to get trapped in local optima.

2.1.7 Learning to Rank Framework

A typical Learning to rank architecture is shown in Figure 2.4 . Training data consists of query-document pairs represented as vectors with some features and relevance judgments. Learning to rank algorithm uses the training data to learn the ranking model, such that the output of the ranking model can predict the ground truth label in the training set as accurately as possible, in terms of a loss function. In the test phase, when a new query comes in, the model learned in the training phase is applied to sort the documents and return the corresponding ranked list to the user as the response to the query.

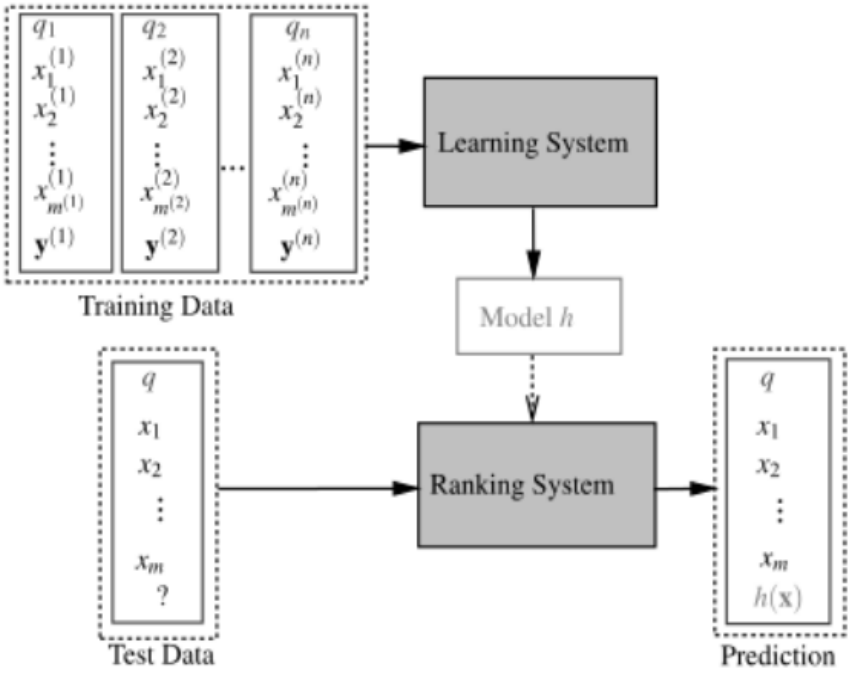

Figure 2.4: Learning to Rank Architecture [6]

There are three types of Learning to Rank:

2.1.7.1 Pointwise approach

In the pointwise approach, the loss function is defined on the basis of single objects and ranking score which could be based on regression, classification. One of the problems with these approaches is that the training model might be biased towards queries with more document pairs. Example: McRank [20], Subset Ranking [8], Pranking [10] etc.

2.1.7.2 Pairwise approach

In the pairwise approach, the loss function is defined on the basis of pairs of objects whose labels are different. The ranking problem is then transformed into the binary classification problem. One advantage of this approach is Predicting relative order is closer to the nature of ranking than predicting class label or relevance score. But the major problem with these methods is that the objective function is formalized as minimizing errors in classification rather than minimizing errors in ranking of documents. Example: RankSVM [17], RankBoost [14], RankNet [4], and FRank [28].

2.1.7.3 Listwise approach

Listwise approaches consider document lists instead of document pairs as instances in learning. This approach has two branches: Direct optimization of IR measures which tries to optimize IR evaluation measures and Listwise loss minimization which minimizes a loss function as an indirect way to optimize the IR evaluation metrics. Example: RankGP [37], AdaRank [35], ListNet [6], and ListMLE [34].

2.2 Literature Review

2.2.1 Related work on Learning to Rank

C. Manning and other researchers from University of California, Berkeley started research in 1990s but the required technology was unavailable back then to implement such Learning to Rank model. RankSVM [17] was the first successful initiative of MLR model based on Support Vector Machine in 2000. Other successful remarkable implementations are: RankBoost [14] in 2003 based on Ada boosting algorithm, GBRank [38] in 2003 based on Gradient Boosting and RankNet [4] in 2005 based on Neural Network, of which GBRank is currently being used by Yahoo in its search engine and RankNet is currently being used by Microsoft in its popular search engine: Bing.

RankGP [37] was the first implementation of Genetic Programming based Learning to Rank Model. The authors showed in that work that an evolutionary based model can outperform non-evolutionary models in terms of accuracy (using MAP evaluation method). However the computation cost was much huge that its not worth comparing it to other non-evolutionary based approaches.

In 2009 Wang et al. proposed RankIP [32], a ranking function discovery approach using Immune Programming. The authors claimed in their experiment that IP is more diversified than GP thus produces better results. They conducted their experiment on two datasets: OSUMED and TREC 2003 which are part of LETOR 2.0 dataset and concluded that both RankIP and RankGP produced similar accuracy on OSUMED (claiming OSUMED dataset is simple) dataset but on TREC 2003 dataset RankIP was superior in terms of accuracy.

RankDE [3] was another implementation of MLR model in 2011 based on a popular Evolutionary Algorithm: Differential Evolution. The authors pointed out one fundamental problem associated with the pair-wise approach (used by non-evolutionary models) is that it only considers two documents at a time and ignore the other documents retrieved for a query. The authors used a list-wise approach in their implementation like RankGP and showed it outperforms previous RankBoost, RankNET, SwarmRank and even RankGP models using MAP evaluation method. However, like RankGP the authors did not tackle one major problem of evolutionary models which is huge computation cost issue.

To solve the computation cost issue Verma et al. proposed a parallel model of Genetic Algorithms using Google's Map-Reduce technology [30]. The authors only applied their model for the OneMax problem and claimed that their model was capable of handling much larger scale problems

In 2011, Wang et al. produced CCRank [31]: A Cooperative Co-Evolution based solution to MLR problem focusing not only accuracy (which was the primary concern of existing research in MLR up until then) but also efficiency due to the dynamic nature of this kind of problem. The

authors also discussed potential ways of achieving parallelization for learning to rank, such as MapReduce, however they only applied the decomposition and combination parallelization technique provided by the CC framework. The authors conducted their research using MQ2007 and MQ2008 datasets, compared CCRank with AdaRank [35], RankBoost [14], RankSVM [17] and ListNet [4], and showed that CCRank is slightly better than other algorithms in terms of accuracy. But when it comes to efficiency their parallel evolution in CCRank leads to a significant speedup in the ranking process.

2.2.2 Related work on Genetic Programming with Evolutionary Strategies

Genetic Programming has been successfully combined with Evolutionary Strategies to solve mainly complex mathematical problems . The authors of [1] used GP algorithm to discover the shape of a mathematical function while the evolution strategy was used to find its coefficients in a Symbolic Regression problem. The experiment was compared against classic GP, and showed that their combined approach was capable of generating better solutions. Authors of [9] also applied (μ+λ) based ES to a number of solutions such as: Binomial-3 problem, Time Series problem and Santa Fe Artificial Ant problem. They found excellent improvements over traditional GP for the first two problems. However, that did not happen with Artificial Ant problem, which generated poor result. In both papers, it was concluded that the concepts of the Evolutionary Strategies techniques can be aggregated to the classical GP, which can perform in general in a better or equivalent way and expressed for further investigation to apply the combined approach to a wide variety of problems.

In our research work, we have extended the work of [9] by investigating and implementing a GP based on ES for a different type of problem which is Learning to Rank. Furthermore, we have extended the work of [37], by digging deep into a few un-explored tasks such as experimenting with OHSUMED dataset [36], more effective algorithmic setting and a different fitness function in order to increase accuracy as well as to increase efficiency by parallelizing the process of evolution which is achieved by utilizing the power of multi-core processors, mentioned in the future

work section. In our experiment, we directly compare our result with RankGP and review some interesting findings.

Chapter 3

Materials and Methods

The aim of this research work is to investigate the applicability of our Hybrid Genetic Programming-Evolutionary Strategies methodology for the Learning to Rank problem (here called RankGPES). 0This chapter continues by describing how the hybrid model is designed and implemented. Moreover, an outline of the algorithm is presented with the experimental setup in later sections, followed by the evaluation criteria for each experiment are described at the end of the chapter.

3.1 The Hybrid GP System

Motivated by the success of Genetic Programming in a wide range of Machine Learning tasks in the industrial area, we propose RankGPES: an evolutionary-based learning to rank algorithm for information retrieval where we have combined Genetic Programming with Evolutionary Strategies. In order to find a more efficient system, we have also experimented with multiple RankGPES running in parallel and thus introduced RankPGPES. The goal of the hybrid system is to discover optimal ranking functions adapted to the properties of a given query-document collection, which are also able to generalize to unseen new queries and documents retrieved at test times.

RankGPES takes a training dataset T as input that contains query-document pairs *(q, d)* with their corresponding feature vectors *f(q, d)*. As an instance of GP, RankGPES learns feature vectors from dataset and constructs ranking functions using those vectors, which evolve and get optimized using a fitness function that evaluates how well the rank scores assigned by the learned ranking

function agree with those assigned by the human evaluators. The final output of Algorithm 2 is an optimal ranking function that maximizes the fitness function. The main steps of RankGPES are summarized in Section 3.1.3.1.

As mentioned in the previous chapter, Fitness function plays the most important part in any Evolutionary Algorithm, which defines how good a solution is and the algorithm tries to optimize this fitness value over the course of generations. In other words, a solution is more accurate if it's fitness function is more effective. Since we are concentrating on an information retrieval problem, the fitness function is defined as a widely-used effective information retrieval measure, Normalized Discounted Cumulative Gain (NDCG). Although (MAP) is more frequently used to measure the accuracy of a document ranking algorithm in information retrieval systems and authors of both RankGP and RankDE used MAP as a fitness function in their implementations, but it is worth experimenting the algorithm with NDCG, as in many cases it proved to be more effective than MAP [29]. Because NDCG directly works with calculating cumulated gain of ranking positions based on document relevancy of a query result, it emphasizes more on the result relevancy at top positions than MAP.

3.1.1 Population Representation

In RankGPES, each individual is a potential solution 'i.e' ranking functions for our problem. An individual I is represented as:

$$I = \{S_c, S_{op}, S_v\} \tag{3.1}$$

Where, S_c=$\{0.1,0.2,0.3.....1.0\}$

S_{op}=$\{+,-,*\}$

S_v=$\{f|f \in F\ \}$,where F is all features

Like RankGP [37], the proposed algorithm only concentrates on linear function as a solution, therefore works with only simple arithmetic operators in S_{op} to avoid huge computational cost and bloating problem caused by generation of unnecessary or wasteful solutions during evolution process. Furthermore, it has been shown that these operators are sufficient to achieve good results in classification problems [18].

An individual is represented as a binary tree structure, in which internal nodes are operators S_{op} and leaf nodes are either variables S_v or constants S_c. The maximum number of available nodes of an individual is determined by the depth of the tree, which is defined before the learning process. A tree structure example is provided in Figure 3.1 for the function, $f : ((f2 \times 0.9) - (0.3 + f1))$:

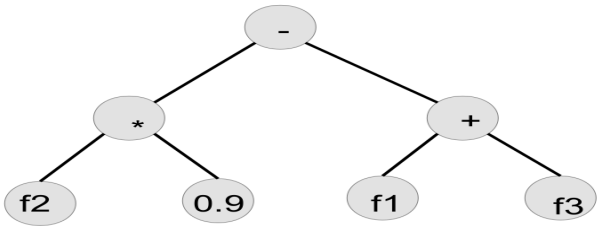

Figure 3.1: Individual Representation of $f : ((f2 \times 0.9) - (0.3 + f1))$

3.1.2 Population Generation

In order to create a diverse initial population both in structure and computational complexity, initial population of RankGPES is generated using Ramped Half and Half [25] method where half of the individuals are created using the "full method" and the other half are generated using the "grow method". In "full method" the structure is generated from root node recursively until depth reaches

to the maximum depth. The "grow method" is similar to "full method" with the exception of tree depth which is chosen randomly between minimum depth to maximum depth. It has been shown in [7] that, Ramped Half and Half is more effective than other population generation methods.

3.1.3 Algorithm

3.1.3.1 RankGPES

In RankGPES (Algorithm 2), we preferred (μ, λ) ES over ($\mu + \lambda$) ES as (μ, λ) ES is capable of generating more diversified solutions than ($\mu + \lambda$) [2]. The Algorithm starts with generating an initial set of individuals with Ramped half and half method [19], where half of the population guarantees to be of full depth and half of the population has the flexibility to be of any depth (within maximum length). Then each of the candidate is evaluated using a fitness function and only μ best candidates are selected to evolve. The evolution process begins with selecting candidates with Tournament Selection operator. Then crossover and mutation operations get applied on those candidates until λ children gets generated. Newly generated candidates are evaluated using fitness function and best μ children are selected to proceed for next iteration, while remaining candidates gets deleted. At each generation the most fit solution is added to output set. This evolution process runs until it satisfies one or more termination conditions. We want to make sure the optimal solution is not only good for training data, but also works well on validation data, therefore each of the solution from output set is evaluated on validation data. After validation, the best solution is outputted. All the parameters used for RankGPES is described in Genetic Programming Parameters section.

3.1.3.2 RankPGPES

RankPGPES (Algorithm 3) is implemented by using an asynchronous island model [27], where each island runs in parallel independently. To diversify our solution we have initialized each island with a different setting than other as per [15]. Each island runs the evolution process for a number

Algorithm 2: RankGP-ES(μ, λ)

INPUT: Training Data
OUTPUT: An Optimal Ranking Function
TRAINING:
1: Initialization:
(a) Generate the initial population.
(b) Calculate fitness.
(c) Select the μ best individuals.
while *the maximum number of generations was not reached* **do**
    2. Using the Tournament Selection method select parents
    3. Apply Crossover and Mutation until λ descendants generated
    4. Calculate Fitness.
    5. Add the best ranking function to Output Set
    6. Select μ best descendants and update population with those for the next iteration.
end
VALIDATION:
7. Each of the solutions in Output Set is evaluated on validation data.
8. The best solution is resulted.

of iterations called epochs. After each epoch, migration process takes place within all islands where n best solutions migrates from one island to other. The algorithm uses ring migration scheme to process the migration task. This particular choice of migration scheme is chosen arbitrarily and because of it's implantation popularity. Experimenting with different migration scheme would be another research question. This evolution process is run through a number of generations and at each generation the most fit solution is added to output set. The process ends when it meets one or more termination conditions. Like the single population method, we want to make sure the best solution not only works well on training data but also produce a good result on validation data. In validation process each solution from output set is evaluation on validation data and finally the best solution gets outputted. Algorithm 3 shows how island model works.

As mentioned in previous paragraph, we have used different algorithms for different island inspired by ideas of [15], Algorithm 4 is another Hybrid algorithm combined with Genetic Programming and (μ+λ)-ES.

Algorithm 3: Algorithm:RankPGPES

INPUT: Training Data
OUTPUT: An Optimal Ranking Function
TRAINING:
1. Create multiple islands
while *the maximum number of generations was not reached* **do**
  foreach *island* **do**
    2. (a) Generate the initial population
    2. (b) Calculate fitness
    while *the maximum epoch size not reached* **do**
      3. (a) Apply Genetic Operators: Selection, Crossover and mutation
      3. (b) Calculate Fitness
      3. (c) Add the best candidate to output set 3. (d) Pass n best candidates to next iteration
    end
    4. Apply Migration: Migrate n individuals to another island
  end
  end;
end
VALIDATION:
4. Each of the solutions in Output Set is evaluated on validation data.
5. The best solution is resulted.

Algorithm 4: RankGP-ES($\mu + \lambda$)

INPUT: Training Data
OUTPUT: An Optimal Ranking Function
1: Initialization:
(a) Generate the initial population.
(b) Calculate fitness.
(c) Select the μ best individuals.
while *the maximum number of generations was not reached* **do**
  2. Using the Tournament Selection method select parents
  3. Apply Crossover and Mutation until λ descendants generated
  4. Calculate Fitness.
  5. Select best μ parents between ($\mu + \lambda$)
end
6. Output the best ranking function

3.1.4 Implementation

In order to implement our GP algorithm we used one of the most popular and powerful Evolutionary Frameworks: "Watchmaker Framework", which comes with Apache 2.0 license. There are both advantages and disadvantages using an existing framework, but in our case the main reason we chose to use it because it comes with a friendly GUI for setting up and testing run parameters, and rich repertoire of tools to collect and interpret run statistics.

Choosing the appropriate framework for the right problem is the key to better productivity and efficiency. There are a number of quality frameworks available such as "ECJ [22]" and "EpochX [24]" which we tested but ended up choosing "Watchmaker Framework [12]" because of it's extensibility, efficient object-oriented design and support for parallel processing of using Island model approach. However, the framework only comes with Genetic Algorithm implementation. Therefore, our implementation includes tree based representations of the candidates, fitness calculations, Genetic Programming operations such as crossover, mutation etc. from scratch, but the key part we got benefited from is re-using the evolution engine which defines the flow of the algorithm. We chose Java to be our preferred language of implementation as it offers a good balance of performance, ease-of-use and a rich standard library.

The algorithms were run on a single computer with quad core processors and 6GB ram. On average, it took approximately 1-2 hours to conduct a single run of RankGPES with 5 fold cross-validation on TREC[1] dataset and 30-50 minutes on OHSUMED[2] dataset. On the other hand, the parallel implementation i.e. RankPGPES took approximately 7 hours to finish a single run with 5 fold cross-validation on TREC dataset and 2-2.5 hours on OHSUMED dataset. The reason RankPGPES takes more time is that it generates a lot of complex and unnecessary solutions which requires extra resources and time to process.

The parameters used for RankGPES and RankPGPES are described in Tables 3.1 and 3.2,

[1] Text REtrieval Conference

[2] Oregon Health and Science University Medical Dataset

Table 3.1: Parameters of RankGPES

RankGPES Parameters	
No. of generations	500
Initial population size	100
Tree Depth	8
Tournament Selection probability	.75
Crossover rate	.9
Crossover type	Single point
Mutation rate	.1
Mutation type	Subtree mutation
Elitism	1
Evolutionary Strategies	(μ, λ)

respectively.

3.2 The Datasets

We have conducted our experiments using LETOR dataset [36], released by Microsoft Research Asia for research on learning to rank for IR, which is standard dataset in learning to rank area. Letor 2.0 contains 2 datasets: OHSUMED and TREC.

3.2.1 OHSUMED

OHSUMED (Oregon Health and Science University Medical Dataset) is a subset of MEDLINE (Medical Literature Analysis and Retrieval System Online) [23] database used for medical publications. The collection consists of 348,566 records from 270 medical journals during the period of 1987-1991. The fields of a record include title, abstract, Medical Subject Headings (MeSH) [23], indexing terms, author, source, and publication type. There are 106 queries. For each query, there are a number of documents associated. Each query is about a medical search need, and thus is also associated with patient information and topic information. The relevance degrees of documents with respect to the queries are judged by humans, on three levels: definitely(2), possibly(1), or not relevant(0). There are a total of 16,140 query-document pairs with relevance judgments. There are

Table 3.2: Parameters of RankPGPES

RankPGPES Parameters	
No. of generations	100
Initial population size	200
Tree Depth	8
Tournament Selection probability	.75
Crossover rate	.9
Crossover type	Single point
Mutation rate	.1
Mutation type	Subtree mutation
Elitism	1
No. of Islands	2
Island 1 Algorithm	GP with (μ, λ) ES
Island 2 Algorithm	GP with ($\mu + \lambda$) ES
Epoch size	50
Migration Type	Ring Migration

25 features or attributes in OHSUMED:

Table 3.3: OHSUMED Features

Feature ID	Descriptions
1	L1 for the T field(Title)
2	L2 for the T field(Title)
3	L3 for the T field(Title)
4	L4 for the T field(Title)
5	L5 for the T field(Title)
6	L6 for the T field(Title)
7	L7 for the T field(Title)
8	L8 for the T field(Title)
9	L9 for the T field(Title)
10	L10 for the T field(Title)
11	L1 for the W field(Abstract)
12	L2 for the W field(Abstract)
13	L3 for the W field(Abstract)
14	L4 for the W field(Abstract)
15	L5 for the W field(Abstract)
16	L6 for the W field(Abstract)
17	L7 for the W field(Abstract)
18	L8 for the W field(Abstract)
19	L9 for the W field(Abstract)
20	L10 for the W field(Abstract)
21	H1 for the joint of T(Title) and W(Abstract) fields
22	H2 for the joint of T(Title) and W(Abstract) fields
23	H3 for the joint of T(Title) and W(Abstract) fields
24	H4 for the joint of T(Title) and W(Abstract) fields
25	H5 for the joint of T(Title) and W(Abstract) fields

Table 3.4: OHSUMED Feature Descriptions

Features	Descriptions	References
L1	Term Frequency(TF)	[36]
L2	Feature in SIGIR paper	[36]
L3	Normalized Term Frequency(TF)	[36]
L4	Feature in SIGIR paper	[36]
L5	Inverse Document Frequency(IDF)	[36]
L6	Feature in SIGIR paper	[36]
L7	Feature in SIGIR paper	[36]
L8	Feature in SIGIR paper	[36]
L9	TF*IDF	[36]
L10	Feature in SIGIR paper	[36]

Table 3.5: OHSUMED Feature Descriptions 2

Features	Descriptions	References
H1	BM25 Score	[36]
H2	log(BM25 Score)	[36]
H3	LMR with DIR Smoothing	[36]
H4	LMR with JM Smoothing	[36]
H5	LMR with AMS Smoothing	[36]

Table 3.6: OHSUMED Record Format

<label> <query id>:<value> <feature id>:<value> ..... <feature id>:<value> # <info>

Identifiers: (L1,L2...L10) and (H1,H2...H5) used in Table 3.3 are explained in Table 3.4 and 3.5 respectively. The structure of each query-document pair is specified in Table 3.6 where <label> takes values from (0, 1, 2),<query id> is an integer, <feature id> is as shown in Table 3, <value> is a float value of the corresponding feature, and document id is given at the end of each line as <info>. An example of a record is shown in Table3.7 It means that for query:1, document:40626 was returned where the label is 2 (definitely relevant). The 25 features extracted for the query-document pairs are (3.00000000, 2.07944154, ... -3.87512000) meaning weight of the first feature is 3.00000000, weight of the second feature is 2.07944154 and so on.

3.2.2 TREC

TREC (Text REtrieval Conference) dataset contains features extracted from query-document pairs in the topic distillation task of TREC 2003 and TREC 2004. TREC 2003 and TREC 2004 used the .GOV collection, which is based on a January, 2002 crawl of .gov Web sites. There are in total

Table 3.7: OHSUMED Record example

2 qid:1 1:3.00000000 2:2.07944154 ..... 25:-3.87512000 # docid = 40626

Table 3.8: TREC Features(1 of 2 tables continued in Table3.9)

Feature ID	Descriptions
1	BM25
2	dl of Body
3	dl of Anchor
4	dl of Title
5	dl of URL
6	HITS Authority
7	HITS Hub
8	HostRank
9	IDF of Body
10	IDF of Anchor
11	IDF of Title
12	IDF of URL
13	Sitemap based feature propagation
14	PageRank
15	LMIR.ABS of Anchor
16	BM25 of Anchor
17	LMIR.DIR of Anchor
18	LMIR.JM of Anchor
19	LMIR.ABS of extracted title
20	BM25 of extracted title
21	LMIR.DIR of extracted title
22	LMIR.JM of extracted title
23	LMIR.ABS of title
24	BM25 of title
25	LMIR.DIR of title

1,053,110 html documents in this collection, together with 11,164,829 hyperlinks. There are 50 queries and 75 queries in topic distillation tasks of the Web track in TREC 2003 and 2004. TD2003 contains 49,171 query-document paris and TD2004 contains 74,170 query-document pairs. The extracted features cover most of the standard IR features, such as classical features (e.g., term frequency, inverse document frequency, and BM25 [24]), and the features proposed in recent SIGIR papers (e.g., HostRank [27], Feature Propagation [23][25], and Topical PageRank [21]). There are in total 44 features were extracted in both TD2003 and TD2004. The relevance judgments are on two levels in TD2003 and TD2004-relevant(1) and not relevant(0).

Table 3.9: TREC Features(2 of 2 tables)

Feature ID	Descriptions
26	LMIR.JR of title
27	Sitemap based feature propagation
28	TF of Body
29	TF of Anchor
30	TF of Title
31	TF of URL
32	TFIDF of Body
33	TFIDF of Anchor
34	TFIDF of Title
35	TFIDF of URL
36	Topical PageRank
37	Topical HITS Authority
38	Topical HITS Hub
39	Hyperlink base score propagation:wighted in link
40	Hyperlink base score propagation:wighted out link
41	Hyperlink base score propagation:uniformed out link
42	Hyperlink base feature propagation:wighted in link
43	Hyperlink base feature propagation:wighted out link
44	Hyperlink base feature propagation:uniformed out link

Table 3.10: TREC Record Format

<label> <query id>:<value> <feature id>:<value> ..... <feature id>:<value> # <info>

Table 3.11: TREC Record Example

1 qid:41 1:5.022967 2:213.000000 3:0.000000 4:8.000000 5:3.000000 6:0.000000 7:0.000000 8:0.000567 9:0.000013 10:0.000456 11:0.000212 12:0.000678 13:4.642564 14:0.000000 15:-12.201200 16:0.021918 17:-11.595500 18:-8.585960 19:-6.167350 20:1.583980 21:-6.106140 22:-6.104610 23:-6.461410 24:1.341180 25:-6.379590 26:-6.377600 27:4.686737 28:0.000000 29:0.000000 30:0.000000 31:0.000000 32:0.000000 33:0.000000 34:0.000000 35:0.000000 36:0.000000 37:0.000000 38:0.000000 39:0.000000 40:0.000000 41:0.000000 42:0.000000 43:0.000000 44:0.000000 # docid = 96

The structure of each query-document pair is specified in Table 3.10 where <label> takes values from (0, 1, 2),<query id> is an integer, <feature id> is as shown in Table 3, <value> is a float value of the corresponding feature, and document id is given at the end of each line as <info>. An example of a record is shown in 3.11 It means that for query:41, document:96 was returned where the label is 1(relevant). The 44 features extracted for the current query-document pair are (5.022967, 213.000000, 0.000000,....., 0.000000) meaning weight of the first feature is 5.022967, weight of the second feature is :213.000000 and so on.

3.2.3 Dataset Partitioning

Each dataset is partitioned into five subsets: *S1, S2, S2, S4* and *S5* in order to conduct 5-fold cross validation. For each fold, 3 subsets are used for training, 1 subset is used for validation and 1 used for testing as shown in Table 3.12:

3.2.4 Data Normalization

Before applying the algorithm to the training set, a query-based normalization on features is performed in order to normalize all feature values into a range of [0, 1]. For a query q_i, the normalized

Table 3.12: TREC Dataset Partition Scheme

SUBSETS	Training	Validation	Testing
FOLD1	(S1, S2, S3)	S4 Set	S5
FOLD2	(S2, S3, S4)	S5	S1
FOLD3	(S3, S4, S5)	S1 Set	S2
FOLD4	(S4, S5, S1)	S2 Set	S3
FOLD5	(S5, S1, S2)	S3	S4

value of $f_k(q_i, d_j)$ is calculated by following equation:

$$f_k(q_i, d_j) = \frac{f_k(q_i, d_j) - min(f_k(q_i, d_i))}{max(f_k(q_i, d_i)) - min(f_k(q_i, d_i))} \tag{3.2}$$

Where f_k is the feature vector of *k-th* item, q_i is the *i-th* query, d_j is the *j-th* document, $max(f_k(q_i, d_i))$ is the maximum value of $f_k(q_i, d_i)$ and $min(f_k(q_i, d_i)$ is the minimum value of $f_k(q_i, d_i)$ for all $d_i \in D$ and D is all documents.

3.3 Evaluation Metrics

We have utilized the standard evaluation tool provided by LETOR to measure accuracy. The evaluation tool provides following IR evaluation measures:

3.3.1 Mean Average Precision (MAP)

Mean average precision (MAP) [21] is the most frequently used accuracy measure of a ranked retrieval run. It is computed by measuring average for each query also known as Average Precision (AP) first, then Average Precisions for all queries are added and divided by the total number of queries as follows:

$$MAP = \frac{\sum_{q \in Q}^{Q} AP(q)}{|Q|} \tag{3.3}$$

Where, q is a query such that $q \in Q$, Q is the set of all the queries and $AP(q)$ is the Average Precision of query q which is calculated by:

$$AP(q) = \frac{\sum_{n=1}^{N} P@n * rel(n)}{\#rel(q)} \tag{3.4}$$

Where N is total number of results returned by the query q, $rel(n)$ is relevancy at position n , the $\#rel(q)$ is total number of relevant documents found for this query q and $P@n$ is the precision at position n is computed by:

$$P@n = \frac{\#rel(n)}{n} \tag{3.5}$$

Where n is the position, and $\#rel(n)$ is the number of relevant documents found in top n results.

3.3.2 Normalized Discounted Cumulative Gain (NDCG)

NDCG [21] is an effective accuracy measure of a ranked retrieval run where the focus is more on relevancy of the results in a way that more relevant results should appear top of less relevant ones. It is computed by dividing the Discounted Cumulative Gain (DCG) by the Ideal Discounted Cumulative Gain (IDCG), where IDCG refers to the idealized DCG: the DCG of the perfect ranking of the result and DCG is measured by:

$$DCG(p) = rel(i) + \sum_{i=2}^{p} \frac{rel(i)}{log_2(i)} \tag{3.6}$$

Where, p is the result position of a query, $rel(i)$ is the relevancy of a particular result at position i. $NDCG$ of a query is measured by the expression below:

$$NDCG(p) = \frac{DCG(p)}{IDCG(p)} \tag{3.7}$$

Where, p is the result position of a query. The higher the NDCG of a query, the more accurate it's ranked list.

3.4 Experimental Design

In our experiments, we aim to obtain satisfactory answers to the following research questions for which we ran 6 experiments, briefly described in Section 3.4.2. Our experimental setup provided in Section 3.4.1:

- Does the hybrid approach (combining Genetic Programming with Evolutionary Strategies) evolves solutions more effectively than classical approach on TREC and OHSUMED dataset?
- Can the hybrid algorithm produce better results by outperforming RankGP and RankDE in terms of accuracy and efficiency?
- Can we solve learning to rank problem more effectively if we distribute the tasks into multiple islands and run them in parallel?
- Can NDCG be a better fitness function than MAP for Learning to Rank problem?
- Higher population or higher number of generations, which parameter is more important that leads to a more accurate solution?

3.4.1 Setup

For simplicity, all of our algorithms target linear solutions, therefore initial maximum tree depth is set to 8 in order to cover the case that leaf nodes contain 44 features and 44 constants. A full

binary tree with a depth of 8 can have 128 leaf nodes, which is enough to cover 44+44=88 nodes. Each algorithm is run 5 times. In each run, 5-fold cross validation was conducted and an average score was computed. The reported score of each algorithm is the average of those in the 5 runs.

3.4.2 Experiments

In our study, the following tests were performed:

3.4.2.1 Accuracy Test on TREC dataset

In this test, Both classic GP algorithm (RankGP [37]) and Hybrid GP algorithm (RankGPES) is run by varying 2 fitness functions: MAP and NDCG, which we referred as RankGP-MAP (RankGP with MAP), RankGP-NDCG (RankGP with NDCG), RankGPES-MAP (RankGPES with MAP) and RankGPES-NDCG (RankGPES with NDCG). However, The parallel algorithm (RankPGPES) is run with only one MAP fitness function due to time constraints. Each of the algorithms is run on TREC dataset and in the end their accuracy, measured in terms of MAP (Mean Average Precision) and NDCG (Normalized Discounted Cumulative Gain), is evaluated using LETOR evaluation tool mentioned in Section 3.3. As mentioned in Section 3.4.1, the algorithms are run in 5-fold cross validation way, where we captured the highest score, the lowest score and averaged out all scores to determine average score of each of the algorithm. Then we compared against each of the algorithms to determine on average which algorithm provides most accurate result on TREC dataset. This test gives us the answer if the hybrid approach generates a better ranking function that ranks documents more accurately than classical approach on TREC dataset. Furthermore, by comparing MAP-fitness based algorithms: RankGP-MAP and RankGPES-MAP with NDCG-fitness based algorithms: RankGP-NDCG and RankGPES-NDCG respectively, this experiment answers whether NGDC-fitness based implementations can outperform their MAP-fitness based implementations in terms of accuracy, therefore if NDCG can be a better fitness function than MAP that results be more effective solutions.

3.4.2.2 Average Fitness Growth Test on TREC dataset

In this experiment, we compared Classical GP algorithms: RankGP-MAP and RankGP-NDCG with Hybrid GP algorithms: RankGPES-MAP and RankGPES-NDCG respectively and inspected each of their fitness values over the course of generations on TREC dataset. RankPGPES is not considered for this test as it's number of generations is different from our other algorithms. The fitness growth pattern is carefully analyzed and therefore we try to answer if the hybrid approach can evolve populations more effectively than classical approach. This experiment also tells us, how many generations do the algorithms take to reach to a converging point and if their fitness growths are interrupted by any early convergence which leads to non-optimal result.

3.4.2.3 Accuracy Test on OHSUMED dataset

In this test, we perform similar experiment as mentioned in Section 3.4.2.1 (so the description is not repeated) but on OHSUMED dataset. This experiment answers us how accurate the algorithm results on OHSUMED dataset and how different are the results from TREC dataset.

3.4.2.4 Average Fitness Growth Test on OHSUMED dataset

Here, we conduct similar experiments as mentioned in Section 3.4.2.2 but on OHSUMED dataset therefore the description is not repeated. This test helps us to answer, if the fitness growth patterns are similar or different on OHSUMED dataset. Also, this experiment helps us to decide the hybrid approach is capable of evolving solutions more effectively than classical approach.

3.4.2.5 Comparison Test of RankGPES with RankDE and RankGP

In this experiment, RankGPES is compared with RankDE [3] and RankGP [37] using their final MAP (Mean Average Precision) score on TREC dataset. Unlike RankGP, RankDE is not implemented in our work and the results are obtained directly from published work. In our work, RankGP and RankDE, the same officially released LETOR 2.0 dataset and it's evaluation tools

Table 3.13: Parameters of RankGPES-high-pop, RankGPES-mid-pop and RankGPES-low-pop

Parameters	RankGPES-mid-pop.	RankGPES-low-pop.	RankGPES-high-pop.
No. of generations	250	500	100
Initial population size	200	100	400

were used, which enable us to make fair and direct comparison. In this test, we also determine how fast RankGPES generates the final solution compared to RankGP and RankDE.

3.4.2.6 Population vs Generation

In this experiment, we intend to answer the following research question:"higher population or higher number of generations, which parameter is more important that results to a better solution?". We ran RankGPES-MAP using three different settings: one with higher initial population and lower number generations referred as "RankGPES-high-pop", one with higher number of generations but lower initial population referred as "RankGPES-low-pop" and another with medium number of initial population and number of generations referred as "RankGPES-med-pop" in this experiment. All other GP parameters such as crossover rate, mutation rate, selection method etc. were kept the same for every setting. Table 3.13 shows RankGPES-high-pop's, RankGPES-mid-pop's and RankGPES-low-pop's number of initial population and number of generations setting.

Chapter 4

Results and Discussion

This chapter reports the results of work with reference to the methodology mentioned in the previous chapter. Here, all algorithms that have been used in our experiment are compared and analyzed. The chapter concludes with a discussion about what we have learned from our experiments.

4.1 Accuracy Test on TREC dataset

In this section, an overview of each Algorithmic run for this test is briefly described first and then the analysis is summarized in Section 4.1.6.

4.1.1 RankGP-MAP

RankGP is the classical GP implementation using MAP as the fitness function. The highest MAP and NDCG was reported to be 0.3869 and 0.6778 respectively and the lowest MAP and NDCG was reported to be 0.2542 and 0.5125 respectively. On average, our experiment showed a similar result as those in [37], that is: average MAP was 0.2995 and average NDCG was 0.5618. Table 4.1 shows the best, worst and average results in more detail. The Final ranking function generated by RankGP-MAP was: ((0.4 - f0) * (0.4 - (((f38 + (((((f28 * (f36 + (((f15 * f6) - f38) - f38))) * (f26 + f0)) - (((f15 * f6) - f38) + ((f15 * f6) - f38))) * f23) - f6)) * f38) + (((((f28 * (f36 + (((f15 * f6) - f38) - f38)))

Where, $f0$ represents feature#1, $f1$ represents feature#2 and so on.

Table 4.1: RankGP-MAP: The highest, the lowest and the average results

	MAP Score	NDCG Score
Highest	0.3869	0.6778
Lowest	0.2542	0.5125
Average	0.2995	0.5618

Table 4.2: RankGP-NDCG: The highest, the lowest and the average results

	MAP Score	NDCG Score
Highest	0.3818	0.6661
Lowest	0.1574	0.5037
Average	0.2349	0.5625

4.1.2 RankGP-NDCG

RankGP-NDCG is the classical GP implementation but using NDCG as the fitness function. The highest MAP and NDCG was reported to be 0.3818 and 0.6661 respectively and the lowest MAP and NDCG was reported to be 0.1574 and 0.5037 respectively. One thing to notice here, in best case scenario the NDCG value is higher than MAP but in worst case scenario it is much lower. On average, our experiment resulted: average MAP:0.2349 and average NDCG: 0.5625. Table 4.2 shows the best, worst and average results in more detail. The final ranking function generated by RankGP-NDCG is:

((((0.9 - f13) + (f15 - f20)) - ((f37 * f19) - (f35 * 0.2))) + (((f5 * 0.9) + (f36 * f25)) + 0.1)) + ((((f6 - f0) + (f26 + f42)) - ((f9 * f2) - 0.1)) - f6))

Where, $f0$ represents feature#1, $f1$ represents feature#2 and so on.

4.1.3 RankPGPES

RankPGPES is the island model where multiple islands run in parallel. As mentioned in previous chapter, we ran 2 islands: 1 island with GP with (μ+λ) and another GP with (μ,λ).For this experiment the highest MAP and NDCG was reported to be 0.3503 and 0.6511 respectively and

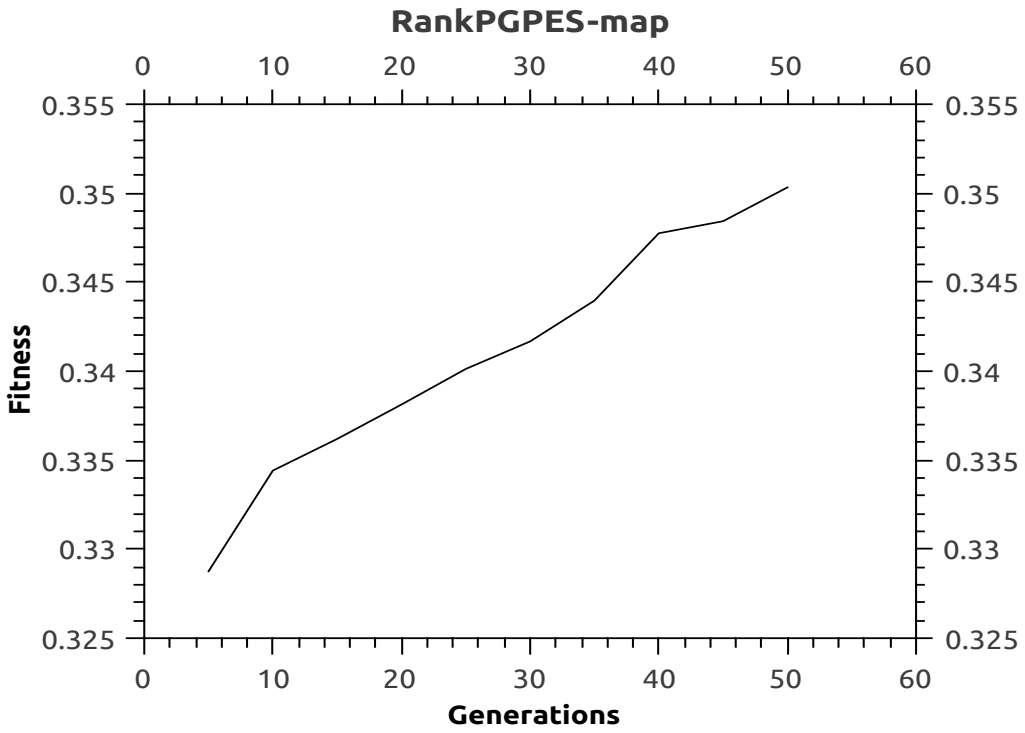

Figure 4.1: RankPGPES-map fitness vs generation graph

the lowest MAP and NDCG was reported to be 0.3288 and 0.6121 respectively. On average, this experiment resulted: average MAP: 0.3395 and average NDCG: 0.6417. Figure 4.1 shows fitness growth by generation and Table 4.3 shows the best, worst and average results in more detail. The final ranking function generated by RankPGPES is:

(((f38 * ((0.4 + f42) - f12)) + (f37 - (0.4 + f0))) + ((f17 - f15) + f18)) * (((0.4 + f0) + ((0.4 + f0) - 0.4)) - (f32 * (f12 + f26)))) + (((f0 + (f26 + (0.3 - ((f5 - ((f32 - f37) + f34)) + f34)))) + ((((f25 * ((0.4 + f0) - 0.2)) - (((f34 * f15) + ((f5 - f12) + (f9 - 0.9))) + ((f43 + (0.4 + f32)) + (f37 - (1.0 + f0))))) - (f43 - f12)) + (f23 - ((f43 + 1.0) + ((0.4 + f0) * f32))))) + ((1.0 - f42) + f5))) + ((((f0 + 0.4) + ((f12 + 1.0) + ((f2 * f32) + f5))) - (0.2 * ((f2 * f32) + f23) + ((((f2 * (f7 * f0)) * (0.2 + f6)) - (0.2 * (f43 - f19))) * ((f3 * (0.3 + f6)) - f12)))))

Where, $f0$ represents feature#1, $f1$ represents feature#2 and so on.

Table 4.3: RankPGPES: The highest, the lowest and the average results

	MAP Score	NDCG Score
Highest	0.3503	0.6511
Lowest	0.3288	0.6121
Average	0.3395	0.6417

Table 4.4: RankGPES-MAP: The highest, the lowest and the average results

	MAP Score	NDCG Score
Highest	0.42035	0.7115
Lowest	0.3394	0.5778
Average	0.360	0.6323

4.1.4 RankGPES-MAP

RankGPES-MAP is the Genetic Programming with Evolutionary Strategies algorithm using MAP as the fitness function.

The highest MAP and NDCG was reported to be 0.42035 and 0.7115 respectively and the lowest MAP and NDCG was reported to be 0.3394 and 0.5778 respectively. On average, the experiment resulted: average MAP:0.3605 and average NDCG: 0.6323 as shown in Table 4.4. The final ranking function generated by RankGPES-MAP is:

((((f0 - 1.0) + ((f31 * (((((((f40 - f31) * (f36 + f21)) * (f15 - (0.4 - f27))) - 0.8) + ((((f40 - f31) * (f40 - 0.4)) + ((f6 * f18) - (1.0 - f40))) + f37)) + ((((f40 + (0.4 - f19)) * f36) - f19) - (f15 + f14))) - (((f14 - 0.8) - (f11 - f42)) * ((((f31 * (f36 + f0)) * f27) * (f27 + (f19 + (f36 + f40)))) * f34)))) * f15)) * ((f40 - 0.8) + (((f6 - 0.4) + ((f6 * f26) - (1.0 - f5))) + 1.0))))

Where, $f0$ represents feature#1, $f1$ represents feature#2 and so on.

4.1.5 RankGPES-NDCG

RankGPES-NDCG is the Genetic Programming with Evolutionary Strategies algorithm using NDCG as the fitness function. This algorithm showed similar aggressive fitness growth per generation as

Table 4.5: RankGPES-NDCG: The highest, the lowest and the average results

	MAP Score	NDCG Score
Highest	0.3828	0.6450
Lowest	0.1554	0.4762
Average	0.2600	0.5813

RankGPES-MAP, however like classical RankGP-NDCG it showed fairly high improvement for best case scenario and very low in worst case scenario. The highest MAP and NDCG was reported to be 0.3828 and 0.6450 respectively and the lowest MAP and NDCG was reported to be 0.1554 and 0.4762 respectively. On average, the experiment resulted: average map:0.2600 and average ndcg: 0.5813 as shown in Table 4.5. The final ranking function generated by RankGPES-NDCG is:

((f41 + (f41 * (f15 * (f6 * (f26 * (f15 * (f6 * (f12 + f15)))))))) - (f15 * (f6 * ((((f2 * (f26 * (f6 * f41))) * (f26 * (f6 * f26))) * ((((f2 * (f26 * (f6 * f15))) + ((f34 * (f42 * f41)) + f26)) + ((f6 * (f15 * (0.3 * f30))) * f13)) * (f6 + f32))) + (((f23 * ((((f41 * f43) - f6) + f17) * ((f34 * (f15 * f15)) + 0.3))) * 0.3) - ((f1 * ((0.9 + f34) + ((f34 * (f42 * f41)) + f30))) - f6)))))) + (f26 + (f23 * (f41 * f12) - f26))

Where, $f0$ represents feature#1, $f1$ represents feature#2 and so on.

4.1.6 Analysis of Accuracy Test on TREC dataset

The analysis of this test is divided in to two evaluation measures:

- Evaluation by MAP Score (Section 4.1.6.1)
- Evaluation by NDCG Score (Section 4.1.6.2)

4.1.6.1 Evaluation by MAP Score

Figure 4.2 shows the highest, the lowest and the average MAP score comparison. As we can see, Hybrid GP algorithms such as RankGPES-MAP and RankGPES-NDCG resulted better MAP

score than non hybrid GP algorithms such as RankGP-MAP and RankGP-NDCG respectively. This was due to RankGPES-MAP and RankGPES-NDCG generated ranking functions that could rank documents more accurately than RankGP-MAP and RankGP-NDCG, respectively. However, NDGC-fitness based algorithms such as RankGP-NDCG and RankGPES-NDCG generated both very high and very low MAP score, therefore on average resulted worse than MAP-fitness based algorithms such as RankGP-MAP and RankGPES-MAP respectively. This behavior can be explained by the fact that, the ranking function that RankGP-NDCG or RankGPES-NDCG resulted could accurately rank some documents only in the best case scenario where training and testing documents are much similar but also failed to rank some documents properly in the worst case scenario where training and testing documents are less similar, hence on average case scenario, the NDCG-fitness based algorithms did not rank documents properly. The Hybrid parallel algorithm, RankPGPES, resulted not very high but not very low MAP score, therefore averaging a moderate MAP score. The reason behind the moderate score is due to the particular migration scheme, where only the best candidates get migrated between islands and survive, resulting each island to have similar solutions that resulted similar ranking functions at the end in almost every run of folds.

4.1.6.2 Evaluation by NDCG Score

Figure 4.3 shows similar comparison of algorithms as seen in Section 4.1.6.1 but using NDCG score. We observe similar pattern of result here as well, which is Hybrid GP algorithms RankGPES-MAP and RankGPES-NDCG outperformed non hybrid GP algorithms RankGP-MAP and RankGP-NDCG respectively. On average, NDCG-fitness based algorithms resulted worse than MAP-fitness based algorithms. However only difference in this comparison is the hybrid parallel algorithm. RankPGPES actually resulted the best NDCG score on average among all 5 algorithms and its result is better than RankGPES-MAP. The reason could be the presence of outliers in generated solutions in our experiments, as this is only scenario where RankPGPES slightly outperformed RankGPES-MAP.

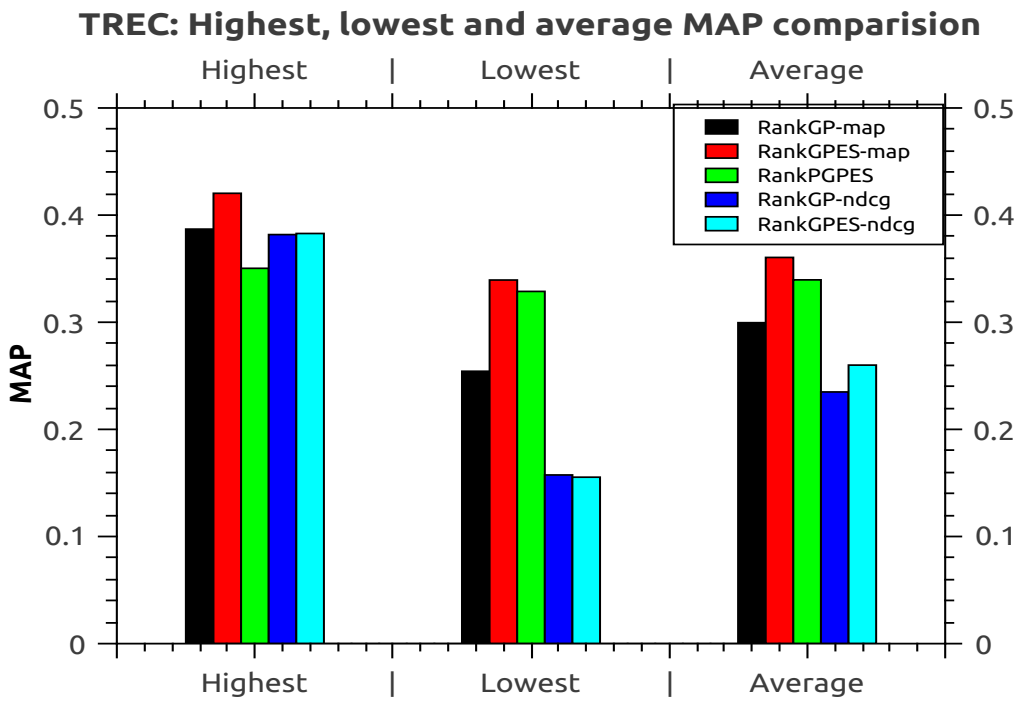

Figure 4.2: TREC:the highest,lowest and average MAP comparison

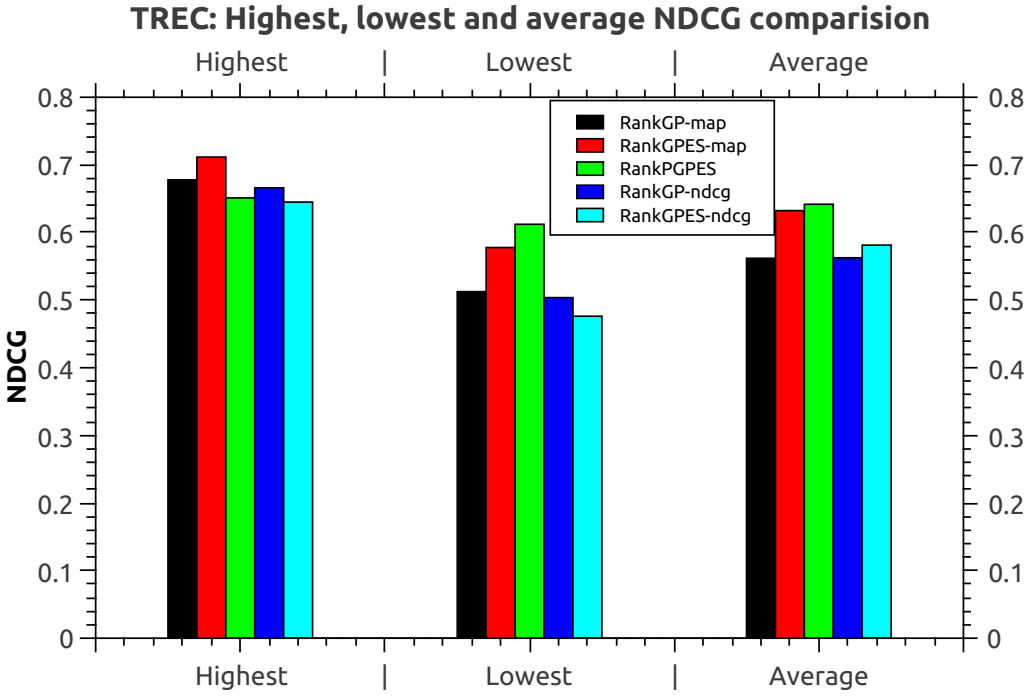

Figure 4.3: TREC:the highest,the lowest and average NDCG comparison

4.2 Average Fitness Growth Test on TREC dataset

Here, we compare RankGP-MAP with RankGPES-MAP described in Section 4.2.1 and RankGP-NDCG with RankGPES described in Section 4.2.2 and analyze each of their fitness growth over the course of generations on TREC dataset.

4.2.1 RankGP-MAP vs RankGPES-MAP

Figure 4.4 shows average fitness growth comparison between RankGP-MAP and RankGPES-MAP. As we can see RankGP-MAP started with a decent growth in early generations (between generation 1 and 100) but after some time (after generation 200) it became steady and barely showed any growth. This behavior was due to the algorithm converged prematurely or got trapped in local optima and could not evolve efficiently to get out of the trap. On the other hand, RankGPES-MAP displayed more growth in early generations (between generation 1 and 100), seemed to got trapped during generation 150 but quickly got out of the trap and showed nearly constant growth in later generations. This behavior can be explained as RankGPES-MAP evolved more efficiently than RankGP-MAP, generated more diversified solutions with better fitness values that helped to get out local optima if got trapped and moved towards global optima more aggressively.

4.2.2 RankGP-NDCG vs RankGPES-NDCG

Figure 4.5 shows average fitness growth comparison between RankGP-NDCG and RankGPES-NDCG.As we can see RankGP-NDCG showed a moderate growth in early generations but that growth rate was slowed down heavily in later generations. However, RankGPES-NDCG showed similar aggressive fitness growth per generation as seen with RankGPES-MAP, although the growth slowed down slightly in mid generations but it resumed to grow sharply at the end. Again, this can be explained as RankGPES-NDCG evolved more efficiently and generated more diversified solutions with better fitness value than RankGP-NDCG to get recovered if the growth slowed down

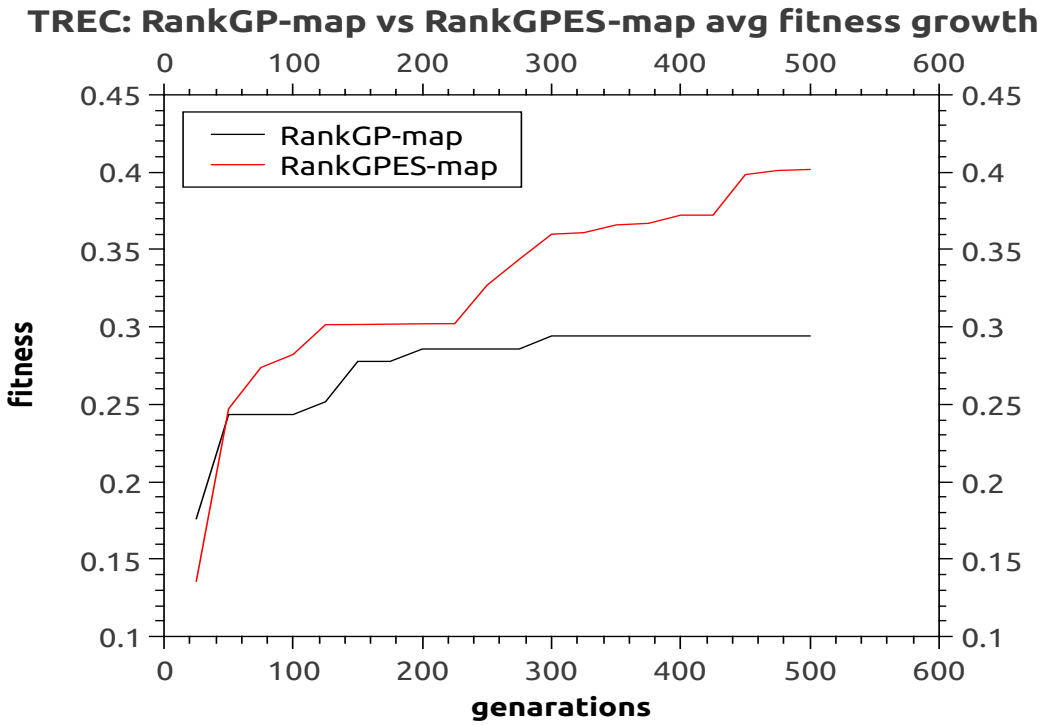

Figure 4.4: TREC:RankGP-MAP vs RankGPES-MAP

during the evolution process.

4.3 Accuracy Test on OHSUMED dataset

In this section, an overview of each algorithmic run for this test is briefly described first and then the analysis is summarized in Section 4.3.6.

4.3.1 RankGP-MAP

The highest MAP and NDCG of RankGP-MAP on OHSUMED dataset was reported to be 0.7386 and 0.7420 respectively and the lowest MAP and NDCG was reported to be 0.5552 and 0.6805 respectively. On average, RankGP-MAP resulted average MAP:0.6720 and average NDCG:0.7287 as shown in Table 4.6. Final ranking function generated by RankGP-MAP is:

(((0.5 * (f6 + 0.9)) + (((0.1 - f9) * 0.1) + ((0.1 + ((f7 + f4) * (f24 + (f9 + f4)))) + (f24 * (f13 + f10))))) * (f24 + (f7 + f4)))

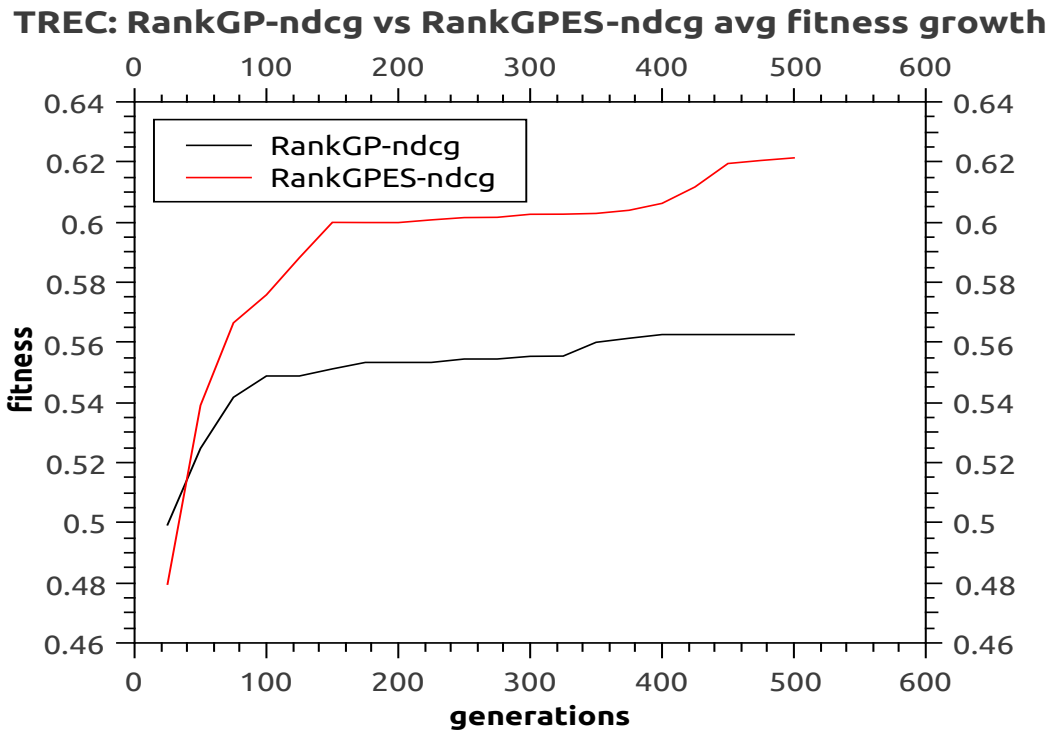

Figure 4.5: TREC:RankGP-ndcg vs RankGPES-ndcg avg fitness growth

Table 4.6: OHSUMED RankGP-MAP: The highest, the lowest and the average results

	MAP Score	NDCG Score
Highest	0.7386	0.7420
Lowest	0.5552	0.6805
Average	0.6720	0.7287

Where, $f0$ represents feature#1, $f1$ represents feature#2 and so on.

4.3.2 RankGP-NDCG

As seen on TREC dataset, RankGP-NDCG implementation followed similar pattern which was: a good improvement in early generations and less improvement in later generations on OHSUMED dataset. The highest MAP and NDCG of RankGP-NDCG on OHSUMED dataset was reported to be 0.7596 and 0.7471 respectively and the lowest MAP and NDCG was reported to be 0.5580 and 0.6083 respectively as shown in Table 4.7. On average, our experiment resulted: average

Table 4.7: OHSUMED RankGP-NDCG: The highest, the lowest and the average results

	MAP Score	NDCG Score
Highest	0.7596	0.7471
Lowest	0.5580	0.6083
Average	0.6691	0.6687

Table 4.8: OHSUMED RankPGPES: The highest, the lowest and the average results

	MAP Score	NDCG Score
Highest	0.7235	0.7411
Lowest	0.6888	0.6941
Average	0.6939	0.7175

MAP:0.6691 and average NDCG: 0.6687. The final ranking function generated by RankGP-NDCG is:

(((((f9 - (f4 - f3)) - f11) - (f4 - f3)) - f11) + (f3 + ((f20 * f5) * ((f2 * f9) + 0.3))))

Where, $f0$ represents feature#1, $f1$ represents feature#2 and so on.

4.3.3 RankPGPES

On OHSUMED dataset, RankPGPES reported the highest MAP and NDCG to be 0.7235 and 0.7411 respectively and the lowest MAP and NDCG to be 0.6888 and 0.6941 respectively. On average, this experiment resulted: average MAP: 0.6939 and average NDCG: 0.7175 as shown in Table 4.8. Figure 4.6 shows fitness vs generation graph in more detail. Final ranking function generated by RankGP-NDCG is:

((((((f22 + 1.0) - (f2 + f18)) - ((f19 - f8) * (f17 - f2))) - (((f1 - 0.7) * (f8 + ((f15 - f19) * (f16 + f15)))) + ((f6 * f17) + (f8 - f2)))) + (((0.2 * ((f23 - f17) - f16)) * ((f15 - f19) * (f16 + f15))) - (((f7 + f19) * (0.3 - f5)) * ((f14 * f24) * 0.2)))))

Where, $f0$ represents feature#1, $f1$ represents feature#2 and so on.

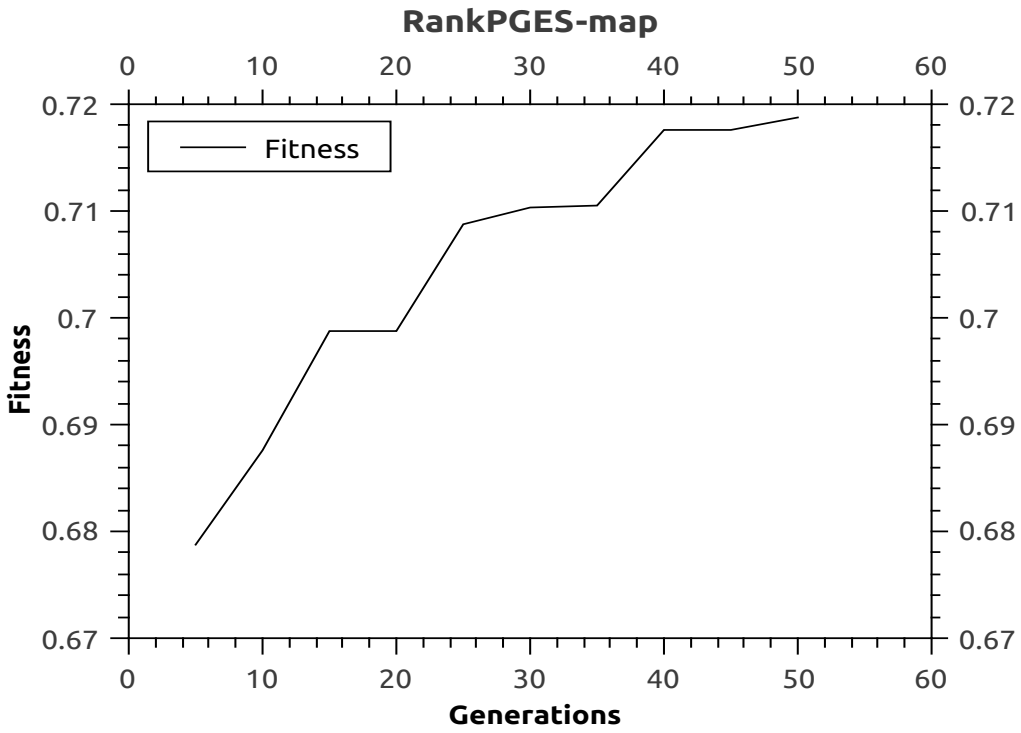

Figure 4.6: OHSUMED RankPGPES-map fitness vs generation graph

4.3.4 RankGPES-MAP

The highest MAP and NDCG of RankGPES-MAP on OHSUMED dataset was reported to be 0.7434 and 0.7632 respectively and the lowest MAP and NDCG was reported to be 0.6432 and 0.6783 respectively. On average, the experiment resulted: average MAP:0.6921 and average ndcg: 0.7425 as shown in Table 4.9. The final ranking function generated by RankGP-NDCG is:

((((f20 * f7) - 0.3) - f3) * (((f14 - f17) * f9) * (((f8 * f2) + 0.1) * ((f5 - 0.7) * (f20 * ((((f8 - 0.3) * f2) * ((f20 - 0.2) - f10)) + ((f7 * f20) - (f15 + 0.4)))))))

Where, $f0$ represents feature#1, $f1$ represents feature#2 and so on.

4.3.5 RankGPES-NDCG

The highest MAP and NDCG of RankGPES-NDCG on OHSUMED dataset was reported to be 0.7596 and 0.7699 respectively and the lowest MAP and NDCG was reported to be 0.5581 and

Table 4.9: OHSUMED RankGPES-MAP: The highest, the lowest and the average results

	MAP Score	NDCG Score
Highest	0.7434	0.7632
Lowest	0.6432	0.6783
Average	0.6921	0.7425

Table 4.10: OHSUMED RankGPES-NDCG: The highest, the lowest and the average results

	MAP Score	NDCG Score
Highest	0.7596	0.7699
Lowest	0.5581	0.6383
Average	0.6692	0.6888

0.6383 respectively. On average, the experiment resulted: average MAP:0.6692 and average NDCG:0.6888 as shown in Table 4.10. The final ranking function generated by RankGP-NDCG is:

(((((((f9 - f16) + f1) + ((0.6 - ((f21 + f14) - f22)) + f7)) + f7) + (f22 + ((f3 * (f1 - f16)) + f7))) - (((0.9 + f16) * (f13 - (f22 + f7))) + f3)) - (f22 - (((f7 + ((f22 * f22) + (f7 + f9))) + f1) - 1.0)))

Where, $f0$ represents feature#1, $f1$ represents feature#2 and so on.

4.3.6 Analysis of Accuracy Test on OHSUMED dataset

The analysis of this test is divided in to two evaluation measures:

- Evaluation by MAP Score (Section 4.3.6.1)
- Evaluation by NDCG Score (Secion 4.3.6.2)

4.3.6.1 Evaluation by MAP Score

Figure 4.7 shows the highest, the lowest and the average MAP score comparison on OHSUMED dataset. As we can see, Hybrid GP algorithms: RankGPES-MAP and RankGPES-NDCG resulted

better MAP score than non hybrid GP algorithms: RankGP-MAP and RankGP-NDCG respectively. However, NDGC-fitness based algorithms RankGP-NDCG and RankGPES-NDCG generated both very high and very low MAP score, therefore on average resulted worse than MAP-fitness based algorithms:RankGP-MAP and RankGPES-MAP respectively. The Hybrid parallel algorithm RankPGPES resulted not very high but not very low MAP score, therefore averaging a moderate MAP score showing a similar result as TREC dataset. As explained in Section 4.1.6.1, this behavior is due to the particular migration scheme, where only the best candidates get migrated between islands and survive, resulting each island to have similar solutions that resulted similar ranking functions at the end in almost every run of folds. In comparison with the accuracy test on TREC dataset mentioned in Section 4.1.6.1, two different patterns were observed in OHSUMED dataset which are:

- the average MAP score generated by each of the algorithms were much closer to each other than the average MAP scores were in TREC dataset.

- RankGP-NDCG and RankGPES-NDCG scored much higher than RankGP-MAP and RankGPES-MAP respectively, but only in the best case scenario.

The more closer average MAP score between algorithms in OHSUMED dataset than TREC dataset seem to be generated because, OHSUMED dataset has much smaller number of feature parameters and overall dataset size is much smaller, *i.e.*, total number of (query,document) pairs are much smaller than TREC, which enable the algorithms to generate more accurate solutions much easily. Furthermore, The reason of very high MAP score of RankGP-NDCG and RankGPES-NDCG only in the best case scenario is: in the best case scenario training documents and testing documents are much similar to each other than in other cases, which is completely opposite for the worst case scenario where these Algorithms scored much lower. This behavior can be summarized as NDCG-fitness based algorithms can learn from training data and predict more accurately on testing data

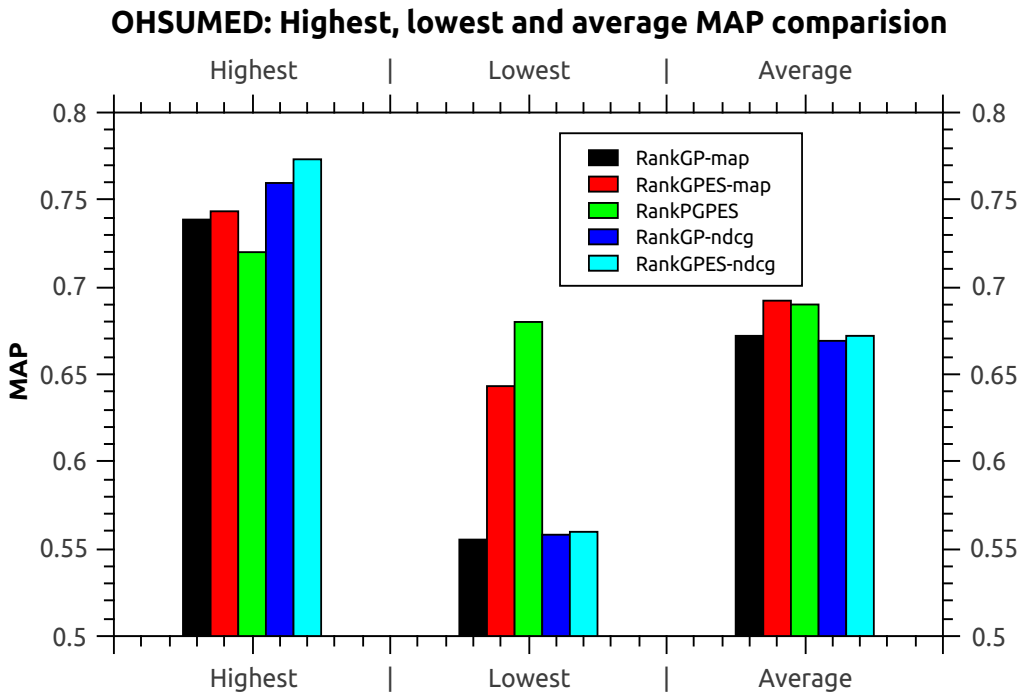

Figure 4.7: OHSUMED:highest lowest and average MAP comparison

than MAP-fitness based algorithms if both testing and training data are similar and specially when the dataset is much smaller and less complex.

4.3.6.2 Evaluation by NDCG Score

Figure 4.8 shows similar comparison of algorithms as seen in Section 4.3.6.1 but using NDCG score. Similar pattern of result can be found here as well, which is Hybrid GP algorithms: RankGPES-MAP and RankGPES-NDCG outperformed non hybrid GP algorithms RankGP-MAP and RankGP-NDCG respectively. Although in the best case scenario, RankGP-NDCG and RankGPES-NDCG seemed to have very high scores, but in the worst case scenario it resulted very poor, implying the ranking function that RankGP-NDCG or RankGPES-NDCG resulted, could correctly rank some subsets of the dataset much accurately when training and testing datasets were similar but also failed to rank some subsets of the dataset properly when training and testing datasets were much different. Hence on average, NDCG-fitness based algorithms resulted worse than MAP-

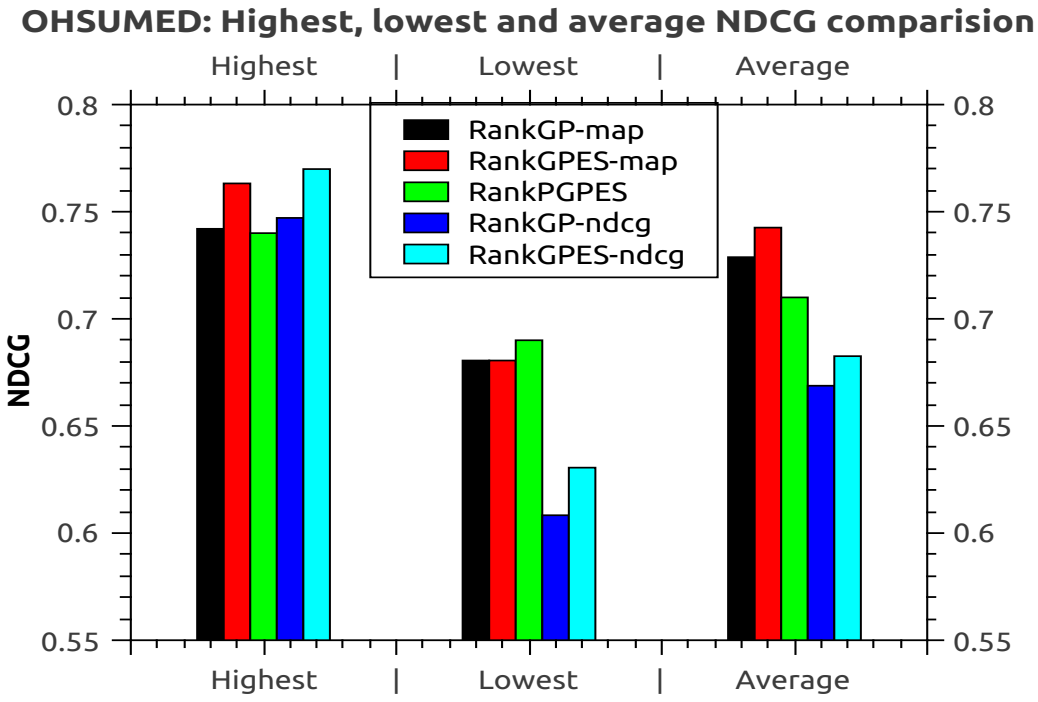

Figure 4.8: OHSUMED:highest lowest and average NDCG comparison

fitness based algorithms. RankPGPES showed similar moderate result in every case meaning not very high and not very low due to its particular migration scheme. Finally, on average RankGPES-MAP topped the NDCG score among all 5 algorithms.

4.4 Average Fitness Growth Test on OHSUMED dataset

Here, we compared RankGP-MAP with RankGPES-MAP described in Section 4.4.1 and RankGP-NDCG with RankGPES described in Section 4.4.2 and analyzed each of their fitness growth over the course of generations on OHSUMED dataset:

4.4.1 RankGP-MAP vs RankGPES-MAP

Figure 4.9 shows average fitness growth comparison between RankGP-MAP and RankGPES-MAP. As we can see both RankGP-MAP and RankGPES-MAP showed substantial growth between generation 1 and 300 but after generation 300 RankGP-MAP got trapped in local optima

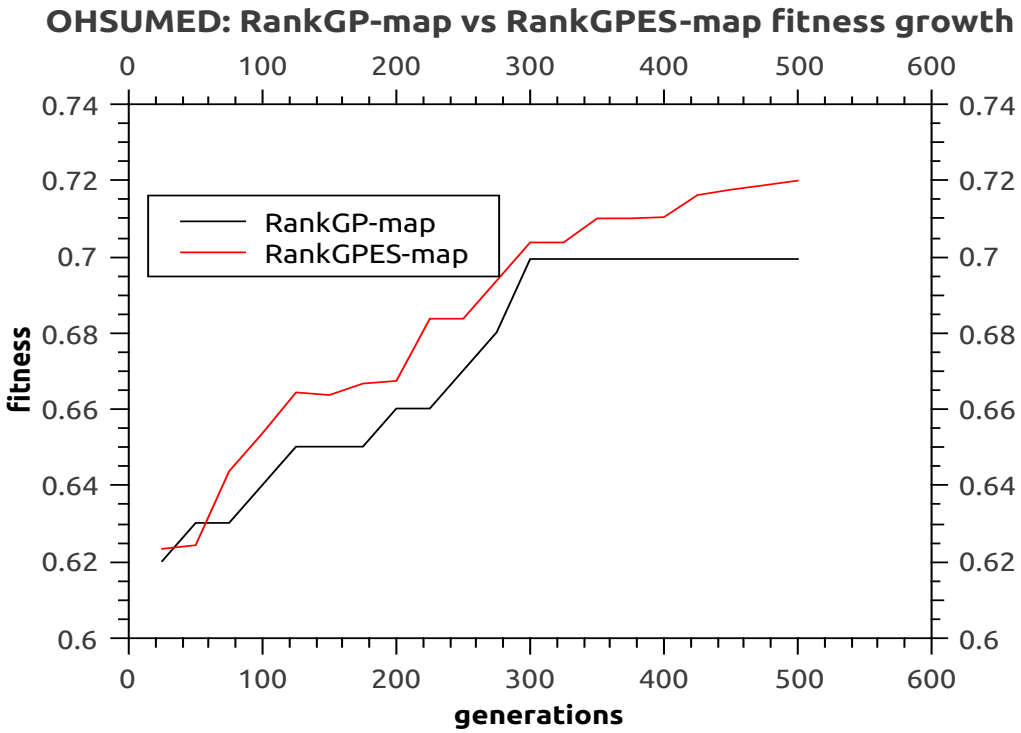

Figure 4.9: OHSUMED:RankGP-MAP vs RankGPES-MAP avg fitness growth

and could not recover. On the other hand, RankGPES-MAP seemed to have gotten trapped in local optima a number of times: during generation 250 and generation 300 but it successfully got out in every case showed gradual growth. This can be explained as RankGPES-MAP evolved more aggressively and generated solutions which were more diversified with more fitness value with than RankGP-MAP that helped to get out local optima when got trapped and moved towards global optima gradually showing no sign of convergence.

4.4.2 RankGP-NDCG vs RankGPES-NDCG

Figure 4.5 shows average fitness growth comparison between RankGP-NDCG and RankGPES-NDCG.The graph showed similar growth pattern as was shown in Section 4.4.1, which can be described as both RankGP-NDCG and RankGPES-NDCG showed moderate growth until at generation 300, then RankGP-NDCG got trapped into local optima where it was stuck for a long time (until generation 480). RankGP-NDCG finally got out after generation 480 and started to show

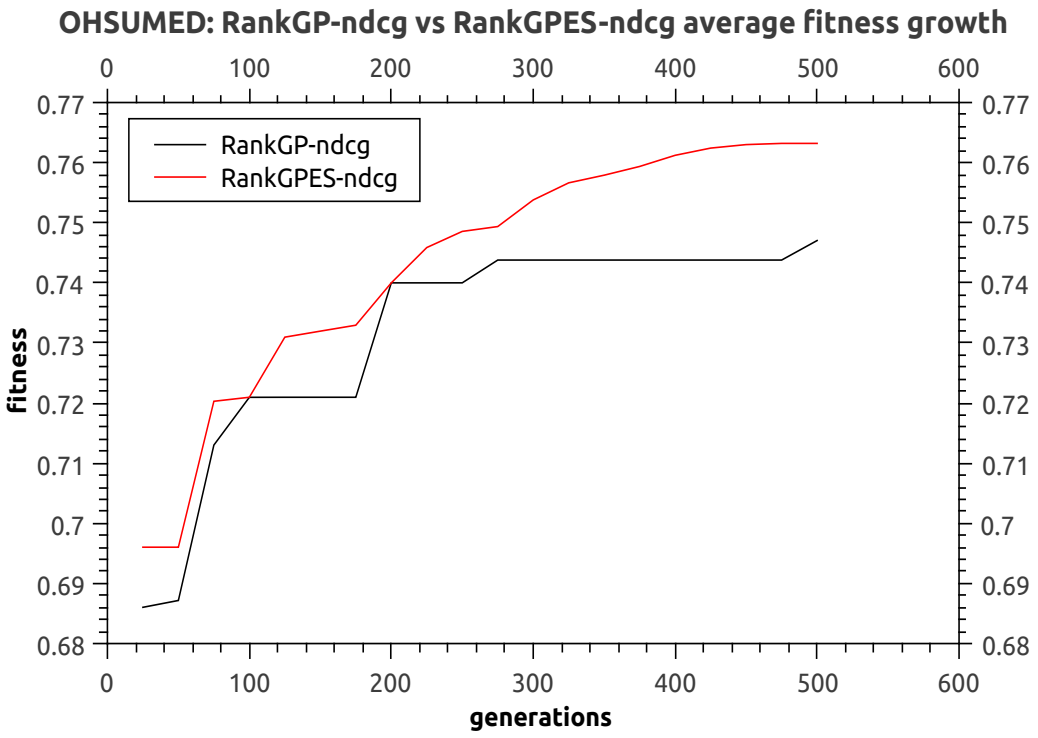

Figure 4.10: OHSUMED:RankGP-NDCG vs RankGPES-NDCG avg fitness growth

some improvement then it reaches maximum number of generations. On the other hand, although RankGPES-NDCG showed slow growth at times (during generation 80 to 110 and generation 150 to 170) but after that it resumed its moderate growth and did not seem to get stuck at any point.Again, this can be explained as RankGPES-NDCG evolved more efficiently and generated more diversified solutions than RankGP-NDCG to get recovered if the growth slowed down during evolution process.

4.5 Comparison Test of RankGPES with RankDE and RankGP

In this analysis, RankGPES is compared with RankDE [3] which is shown in Table4.11. Unlike RankGP, RankDE was not implemented in our work and the results were obtained directly from published work. In both our work and RankDE, the same officially released LETOR 2.0 (only TREC) dataset and its evaluation tools were used, which enable us to make a direct and a fair comparison. As we can see from Table 4.11, on average RankGPES outperforms RankGP by

Table 4.11: Comparison of RankGPES with RankDE and RankGP using MAP score

Algorithm	Average MAP Score
RankGPES	0.360
RankGP	0.299
RankDE	0.339

20% and RankDE by 6% in terms of accuracy. Furthermore, RankGPES is significantly more efficient than RankGP where RankGP took 3 days to complete a single training on TREC dataset, while RankGPES took only 1-2 hours to complete on the same dataset and 30 to 40 minutes on OHSUMED dataset. This is achieved by utilizing multi-threading on multi core processors. Again, RankDE took 10,000 generations to reach to a satisfactory level whereas RankGPES was able to reach to that level in significantly less number of generations (i.e., only 500 generations).

4.6 Population vs Generation Experiment

As we see in Figure 4.11, RankGPES-MAP-high-population: the algorithm with higher initial population generated solutions with almost similar fitness values as RankGPES-mid-population and slightly higher fitness values than RankGPES-low-population. This behavior can be explained as, a very high population is capable of generating better solutions than lower population, but not necessarily always. Also having more population requires extra resources to process them, therefore an optimal number of initial population should be preferred. However, as generations passed by RankGPES-MAP-mid-population outperformed RankGPES-MAP-high-population and RankGPES-MAP-low-population outperformed RankGPES-MAP-mid-population by having solutions with higher fitness value. This can be explained as, although a larger number of diversified initial Population helps to generate good solutions, but if they are efficiently evolved using an optimal fitness function, those Population are capable or generating better solutions throughout a longer period of generations until global optima has been reached.

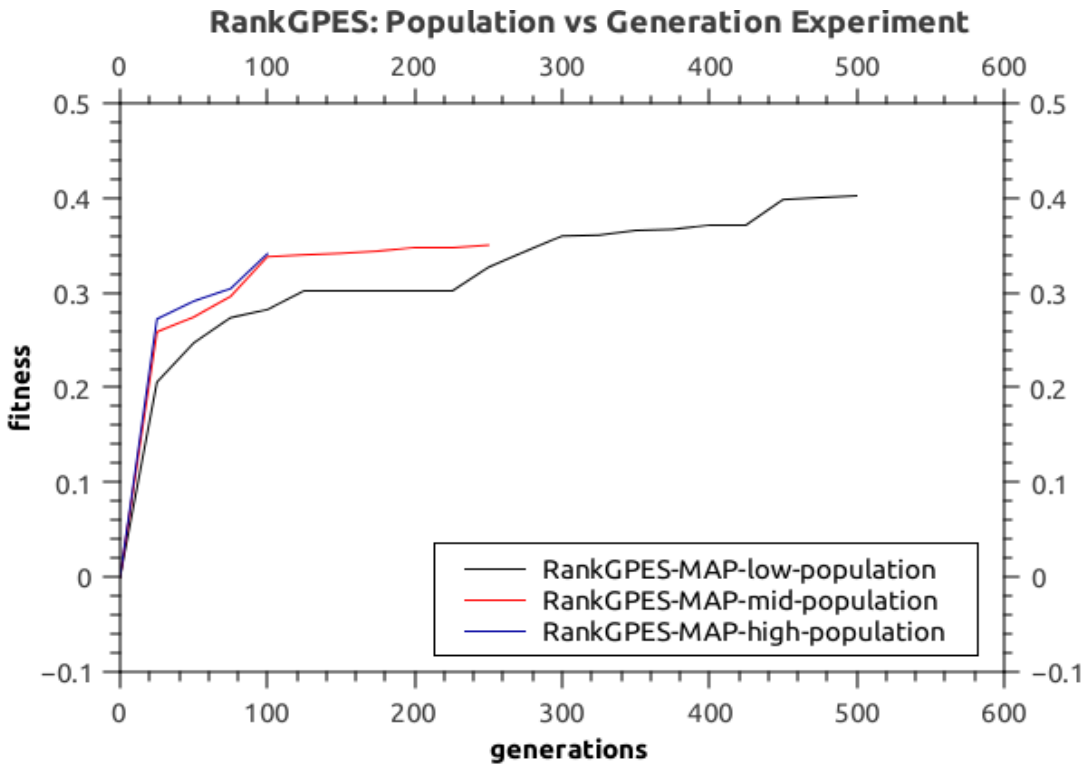

Figure 4.11: RankGPES-MAP: Population vs Generation comparison

4.7 Discussion

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

- Based on our experiments, we found out higher MAP and NDCG scores on OHSUMED dataset than TREC dataset with all our algorithms as shown in Figure 4.7, Figure 4.8, Figure 4.2 and Figure 4.3 respectively. Also, it took almost 50% less time to process Algorithms on OHSUMED dataset than on TREC dataset as shown in Section 3.1.4. That led to a conclusion that, it is easier to rank OHSUMED data than TREC data and the reason being that the TREC dataset not only has more features than the OHSUMED dataset, but also it contains more (query, document) pairs which make overall ranking task more complex than with OHSUMED dataset.

• In Learning to Rank Problem, Genetic programming with Evolutionary Strategies techniques can generate a very diversified solution structurally with a very competitive result (by comparing with baseline results in [36]) in Learning to Rank area, while outperforming previous Evolutionary based implementations such as RankGP [37] by 20% and RankDE [3] by 6% in terms of accuracy as shown in Table 4.11.

• In our experiments, we discovered that, although NDCG (Normalized Discounted Cumulative Gain) based fitness functions generated a good solution in the best case, but in the worst case it generated a very poor solution shown in Figure 4.3 and Figure 4.8 for TREC and OHSUMED dataset respectively.

• By analyzing Figure 4.2, Figure 4.3, Figure 4.7 and Figure 4.8 for TREC and OHSUMED respectively it can be finalized that, on average, MAP (Mean Average Precision) based fitness functions resulted better solution than NDCG (Normalized Discounted Cumulative Gain).

• We found that in our problem, the number of generation is more important than the number of initial population. Using a large number of generations with a small population showed better results than using a large number of population with a small number of generations as shown in Section 4.6. Although it can vary to problem.

• In our experiments, we noticed that classical GP got trapped in local optima very often and showed early convergence. On the other hand RankGPES tended not to fall under any local optima and kept moving towards a continuous progress which sometimes created difficulty to find a converging point as summarized on Section 4.2.1 and Section 4.2.2 for TREC and Section 4.4.1 and Section 4.4.2 for OHSUMED.

• Both classical approach and hybrid approach showed very different results when ran on different folds of training data on each dataset, therefore found very high and very low MAP and NDCG scores at times as explained in Section 4.1.6.1,Section 4.1.6.2, Section 4.3.6.1

and Section 4.3.6.2 for TREC and OHSUMED respectively. However the parallel model was found to be the most consistent in every scenario because the algorithm's migration mechanism, using which the best solutions from each islands/clusters got transferred to other islands/clusters, resulting consistent solution at the end in almost every folds of training data.

• However parallel model may not the ideal choice of algorithm, because in our experiment it created more bloating problem than the single population run by generating more unnecessary solutions, because of the particular migration scheme, where only the best candidates get migrated between islands and survive, that led to a creation of more number of less diversified solutions that took huge time and computational resources to process which soon died off resulting a lot efforts got wasted unnecessarily.

• Overall, out of all the algorithms we have experimented with, RankGPES proved to be capable of producing results with high precision value while generating most complex relations structurally between program parameters as summarized in Section 4.1.6.1, Section 4.1.6.2, Section 4.3.6.1 and Section 4.3.6.2 for TREC and OHSUMED respectively.

Chapter 5

Conclusions

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

5.1 Summary of research

This research evaluates the effectiveness of an enhanced version of genetic programming by combining with evolutionary strategies in learning to rank problem to generate an near-optimal ranking function. In total, 5 algorithms (RankGP-MAP, RankGP-NDCG, RankGPES-MAP, RankGPES-NDCG and RankPGPES) have been tested on 2 datasets (TREC and OHSUMED). MAP fitness based algorithms have been compared against NDCG fitness based algorithms to see if NDCG can be used as a better fitness function instead of MAP. Classical GP based implementations have been compared with Hybrid GP algorithms to see if combining Genetic Programming with Evolutionary Strategies helps to evolve solutions more effectively. Moreover, multiple Hybrid GP algorithms have been run in parallel using Island model approach, to see if solutions can be generated more efficiently. Furthermore, the Hybrid algorithm RankGPES has been run in multiple settings by varying number of initial populations and number of generations to see which parameter is more important for an optimal result. In the end, RankGPES has been compared against RankGP and RankDE to see if RankGPES can outperform them in terms of accuracy and efficiency.

5.2 Summary of results and conclusions

The Hybrid approach was shown to be very effective against the tested datasets and produced high mean average precision score. However, the resulted solution was fairly long and produced a complex relationship among ranking features structurally. The poor results appeared to have generated from the worst case scenarios, where the testing query-document pairs were completely different from training data. This behavior is probably due to the small size of each fold of the training datasets, where each fold contains very different query-document pairs than other folds. A bigger fold size might have given us better results but to make the comparisons between algorithms fair, we have used standard fold size set by LETOR. The gain in ranking shown by LETOR evaluation tool [36] as an indicator of substantial performance improvement over existing RankGP and RankDE. We have seen TREC dataset is much complex and harder to learn and rank than OHSUMED dataset based on our tests. Our experiments showed although NDCG is an effective Information Retrial measure, but it is not quite effective to replace MAP as GP fitness operator because, on average case scenario NDCG-fitness based algorithms generated less accurate results (measured by MAP and NDCG) than MAP-fitness based algorithms. Moreover, parallel implementation of GP algorithms was capable of producing consistent result in almost all scenarios because of its particular migration scheme, but was not preferred due it's bloating problem, slowness and requirement of huge computational resources. Moreover, higher number of generations proved to be more important than higher number of initial populations that generated more accurate results for our algorithm. Finally, some suggestions are provided below that may also aid in yielding important performance improvements with this approach.

5.3 Future work

While some of the results presented in this report are appealing, a number of areas for future work exist that may greatly improve the effectiveness of this approach:

• Genetic Programming parameters: In general, Genetic Programming behaves differently when experimented with a different set of breeding operators or different probability used by those operators. A dynamic set of those probabilities that can change overtime automatically to output maximum fitness value or use of an additional evolutionary algorithm only for these parameters to find the best setting that outputs maximum fitness value, may significantly result better convergence and improve overall accuracy.

• Experimenting with enhanced fitness functions: We have used two commonly used Information Retrieval measures which are MAP and NDCG as fitness functions in our experiments. Some researchers already discovered some enhanced versions of these Information Retrieval measures such as Mean reciprocal rank, Structural Relevance, etc. Using these enhanced Information Retrieval measures as fitness function may also lead to a better result.

• Use of a different migration scheme for the parallel implementation may produce more diversified solutions which can result more accurate solutions.

• Increase the number of runs of each algorithm can help to reduce number outliers in the result significantly. Additionally, use of some outliers removal techniques such as One Class Support Vector Machines can be used to detect and remove outliers from results, can produce more accurate results.

• Distributed processing: Evolutionary approaches are computationally expensive in nature. The island model we presented in our experiment can be implemented effectively to process in a distributed way across multiple systems using Map/Reduce [11] technology. This can greatly reduce training time.

• Feature Selection Algorithm: Finally, a use of good feature selection algorithm (*i.e.*, the process of selecting a subset of the most relevant or important features of the problem) can reduce unnecessary ranking features and generate a concise and non-complex relationship

among parameters that impact the ranking process the most.

Bibliography

[1] C. L. Alonso, J. L. Montaña, and C. E. Borges. Evolution strategies for constants optimization in genetic programming. In *Proceedings of the 2009 21st IEEE International Conference on Tools with Artificial Intelligence*, ICTAI '09, pages 703–707, Washington, DC, USA, 2009. IEEE Computer Society.

[2] H.-G. Beyer and H.-P. Schwefel. Evolution strategies a comprehensive introduction. 1(1):3–52, May 2002.

[3] D. Bollegala, N. Noman, and H. Iba. Rankde: learning a ranking function for information retrieval using differential evolution. In N. Krasnogor and P. L. Lanzi, editors, *GECCO*, pages 1771–1778. ACM, 2011.

[4] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender. Learning to rank using gradient descent. In *Proceedings of the 22nd international conference on Machine learning*, ICML '05, pages 89–96, New York, NY, USA, 2005. ACM.

[5] E. Cant-Paz. Designing efficient and accurate parallel genetic algorithms. Technical report, 1999.

[6] Z. Cao, T. Qin, T.-Y. Liu, M.-F. Tsai, and H. Li. Learning to rank: from pairwise approach to listwise approach. In *ICML '07: Proceedings of the 24th international conference on Machine learning*, pages 129–136, New York, NY, USA, 2007. ACM.

[7] K. Chellapilla. Evolving computer programs without subtree crossover. *Trans. Evol. Comp*, 1(3):209–216, Sept. 1997.

[8] D. Cossock and T. Zhang. Subset ranking using regression. In G. Lugosi and H. Simon, editors, *Learning Theory*, volume 4005 of *Lecture Notes in Computer Science*, pages 605–619. Springer, Berlin / Heidelberg, 2006.

[9] E. Costa and A. Pozo. A new approach to genetic programming based on evolution strategies. In *Systems, Man and Cybernetics, 2006. SMC '06. IEEE International Conference on*, volume 6, pages 4832 –4837, oct. 2006.

[10] K. Crammer and Y. Singer. Pranking with ranking. In *Advances in Neural Information Processing Systems 14*, pages 641–647. MIT Press, 2001.

[11] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, Jan. 2008.

[12] D. Dyer. Watchmaker framework for evolutionary computation. `http://watchmaker.uncommons.org`, 2013.

[13] A. E. Eiben and J. E. Smith. *Introduction to Evolutionary Computing*. SpringerVerlag, 2003.

[14] Y. Freund, R. Iyer, R. E. Schapire, and Y. Singer. An efficient boosting algorithm for combining preferences. *J. Mach. Learn. Res.*, 4:933–969, Dec. 2003.

[15] S. M. Gustafson. *An Analysis of Diversity in Genetic Programming*. PhD thesis, School of Computer Science and Information Technology, University of Nottingham, Nottingham, U.K, 2004.

[16] B. He, C. Macdonald, and I. Ounis. Retrieval sensitivity under training using different measures. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '08, pages 67–74, New York, NY, USA, 2008. ACM.

[17] T. Joachims. Optimizing search engines using clickthrough data. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '02, pages 133–142, New York, NY, USA, 2002. ACM.

[18] J. K. Kishore, L. M. Patnaik, V. Mani, and V. K. Agrawal. Application of genetic programming for multicategory pattern classification. 4(3):242–258, Sept. 2000.

[19] J. R. Koza. Introduction to genetic programming. In *Proceedings of the 2007 GECCO conference companion on Genetic and evolutionary computation*, GECCO '07, pages 3323–3365, New York, NY, USA, 2007. ACM.

[20] P. Li, C. J. C. Burges, and Q. Wu. Mcrank: Learning to rank using multiple classification and gradient boosting. In J. C. Platt, D. Koller, Y. Singer, and S. T. Roweis, editors, *NIPS*. Curran Associates, Inc., 2007.

[21] T.-Y. Liu. Learning to rank for information retrieval. *Found. Trends Inf. Retr.*, 3(3):225–331, Mar. 2009.

[22] S. Luke. A java-based evolutionary computation research system. `http://cs.gmu.edu/~eclab/projects/ecj`, 2013.

[23] MEDLINE. Bibliographic database of the top medical journals. `http://www.ebscohost.com/biomedical-libraries/medline`, 2013.

[24] F. E. B. Otero, T. Castle, and C. G. Johnson. EpochX: Genetic programming in Java with statistics and event monitoring. In *Proceedings of the 2012 Genetic and Evolutionary Conference Companion (GECCO 2012)*, Philadelphia, July 2012. ACM Press.

[25] R. Poli, W. B. Langdon, and N. F. McPhee. *A Field Guide to Genetic Programming*. Lulu Enterprises, UK Ltd, 2008.

[26] K. S. N. Ripon and M. N. H. Siddique. Evolutionary multi-objective clustering for overlapping clusters detection. In *Proceedings of the Eleventh conference on Congress on Evolutionary Computation*, CEC'09, pages 976–982, Piscataway, NJ, USA, 2009. IEEE Press.

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

[28] M.-F. Tsai, T.-Y. Liu, T. Qin, H.-H. Chen, and W.-Y. Ma. Frank: a ranking method with fidelity loss. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '07, pages 383–390, New York, NY, USA, 2007. ACM.

[29] H. Valizadegan, R. Jin, R. Zhang, and J. Mao. Learning to rank by optimizing ndcg measure. In Y. Bengio, D. Schuurmans, J. D. Lafferty, C. K. I. Williams, and A. Culotta, editors, *NIPS*, pages 1883–1891. Curran Associates, Inc., 2009.

[30] A. Verma, X. Llorà, D. E. Goldberg, and R. H. Campbell. Scaling genetic algorithms using mapreduce. In *Proceedings of the 2009 Ninth International Conference on Intelligent Systems Design and Applications*, ISDA '09, pages 13–18, Washington, DC, USA, 2009. IEEE Computer Society.

[31] S. Wang, B. J. Gao, K. Wang, and H. W. Lauw. Ccrank: Parallel learning to rank with cooperative coevolution. In W. Burgard and D. Roth, editors, *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2011, San Francisco, California, USA, August 7-11, 2011*. AAAI Press, 2011.

[32] S. Wang, J. Ma, and J. Liu. Learning to rank using evolutionary computation: immune programming or genetic programming? In *Proceedings of the 18th ACM conference on In-*

formation and knowledge management, CIKM '09, pages 1879–1882, New York, NY, USA, 2009. ACM.

[33] Wikipedia. Darwinism — wikipedia, the free encyclopedia. `http://en.wikipedia.org/w/index.php?title=Darwinism`, 2012.

[34] F. Xia, T.-Y. Liu, J. Wang, W. Zhang, and H. Li. Listwise approach to learning to rank: theory and algorithm. In *Proceedings of the 25th international conference on Machine learning*, ICML '08, pages 1192–1199, New York, NY, USA, 2008. ACM.

[35] J. Xu and H. Li. Adarank: a boosting algorithm for information retrieval. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '07, pages 391–398, New York, NY, USA, 2007. ACM.

[36] T. yan Liu, J. Xu, T. Qin, W. Xiong, and H. Li. Letor: Benchmark dataset for research on learning to rank for information retrieval. In *In Proceedings of SIGIR 2007 Workshop on Learning to Rank for Information Retrieval*, 2007.

[37] J. yuan Yeh, J. yi Lin, H. ren Ke, and W. pang Yang. Learning to rank for information retrieval using genetic programming. SIGIR '07, 2007.

[38] Z. Zheng, K. Chen, G. Sun, and H. Zha. A regression framework for learning ranking functions using relative relevance judgments. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '07, pages 287–294, New York, NY, USA, 2007. ACM.