UNIVERSIDADE FEDERAL DO AMAZONAS
INSTITUTO DE COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

MULTI-OBJECTIVE OPTIMIZATION IN LEARN TO PRE-COMPUTE EVIDENCE FUSION TO OBTAIN HIGH QUALITY COMPRESSED WEB SEARCH INDEXES.

Anibrata Pal

APRIL 2016

MANAUS, AM – BRAZIL

UNIVERSIDADE FEDERAL DO AMAZONAS
INSTITUTO DE COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

MULTI-OBJECTIVE OPTIMIZATION IN LEARN TO PRE-COMPUTE EVIDENCE FUSION TO OBTAIN HIGH QUALITY COMPRESSED WEB SEARCH INDEXES.

Anibrata Pal

Dissertation of Master's degree presented to the Program of Post-Graduation in Informatics, Institute of Computation - IComp, of Federal University of Amazonas, as a part of requirements necessary to obtain the degree of Master of Informatics.

Adviser: Edleno Silva de Moura

APRIL 2016

MANAUS, AM – BRAZIL

Ficha Catalográfica

Ficha catalográfica elaborada automaticamente de acordo com os dados fornecidos pelo(a) autor(a).

A597m

Anibrata, Pal Anibrata

Multi-objective optimization in learn to pre-compute evidence fusion to obtain high quality compressed web search indexes / Pal Anibrata Anibrata. 2016

76 f.: il.; 31 cm.

Orientador: Edleno Silva de Moura

Dissertação (Mestrado em Informática) - Universidade Federal do Amazonas.

1. Multi-Objective Optimization. 2. Linear Convex Combination. 3. Weighted Geometric Mean. 4. Pareto Evolutionary Algorithm. 5. Index Compression. I. Moura, Edleno Silva de II. Universidade Federal do Amazonas III. Título

FOLHA DE APROVAÇÃO

"MULTI-OBJECTIVE OPTIMIZATION IN LEARN TO PRE-COMPUTE EVIDENCE FUSION TO OBTAIN HIGH QUALITY COMPRESSED WEB SEARCH INDEXES."

ANIBRATA PAL

Dissertação de Mestrado defendida e aprovada pela banca examinadora constituída pelos Professores:

[illegible]

Prof. Edleno Silva de Moura - PRESIDENTE

[illegible]

Prof. Altigran Soares da Silva - MEMBRO INTERNO

[illegible]

Prof. Rodrygo Luis Teodoro Santos - MEMBRO EXTERNO

Manaus, 19 de Abril de 2016

To my family ...

"You have the right to work only, but never to its fruits,
Let not the fruits of action be your motive, nor let your attachment be to inaction"
- Bhagavad Geeta

Thanks

I would like this opportunity to thank my parents for raising me in their best possible capability, and that without their support it would not have been possible for me to be where I am now.

My wife Ahana, whom I thank the most, have supported me in each and every step, that have taken me to where I am now. I would like to thank my daughter Aadriya, who, in spite of being a toddler, understood my needs and always lent her little hands of support. I would like to thank my parents-in-law who helped me in every possible way they could.

I would like to thank my guide, my adviser, Prof. Edleno, who have been there always, whenever I needed any help in any kind and measure. His expertise, composure and resolve is something that I always look up to. I would also like to thank Prof. André for his incessant support during the past couple of years, patiently helping me in every possible way.

I would take this opportunity to thank all of my professors, Altigran, David Fernandes, Marco Cristo, Eulanda Santos, Eduardo Souto, João Cavalcanti, Moises Carvalho, Eduardo Feitosa, Rosiane Rodrigues for all the support and help during past two years.

I would like to thanks my friends Caio, Joyce, Urique, Diego Barros specifically here in UFAM. I would like thank each and everyone in my courses and in my laboratory who has constantly helped me in various ways in past couple of years.

I might not have mentioned the names of all of those who have by some way or other helped me in innumerable ways possible. I would like to convey my heartfelt gratitude to all of them.

Resumo

Máquinas de busca web para a web indexam grandes volumes de dados, lidando com coleções que muitas vezes são compostas por dezenas de bilhões de documentos. Métodos aprendizagem de máquina têm sido adotados para gerar as respostas de alta qualidade nesses sistemas e, mais recentemente, há métodos de aprendizagem de máquina propostos para a fusão de evidências durante o processo de indexação das bases de dados. Estes métodos servem então não somente para melhorar a qualidade de respostas em sistemas de busca, mas também para reduzir custos de processamento de consultas. O único método de fusão de evidências em tempo de indexação proposto na literatura tem como foco exclusivamente o aprendizado de funções de fusão de evidências que gerem bons resultados durante o processamento de consulta, buscando otimizar este único objetivo no processo de aprendizagem.

O presente trabalho apresenta uma proposta onde utiliza-se o método de aprendizagem com múltiplos objetivos, visando otimizar, ao mesmo tempo, tanto a qualidade de respostas produzidas quando o grau de compressão do índice produzido pela fusão de rankings. Os resultados apresentados indicam que a adoção de um processo de aprendizagem com múltiplos objetivos permite que se obtenha melhora significativa na compressão dos índices produzidos sem que haja perda significativa na qualidade final do ranking produzido pelo sistema.

PALAVRAS-CHAVE: Otimização Multi-Objetivo, Combinação Linear Convexo, Geométrico Médio Ponderado, Algoritmo Evolutivo Pareto, Compressão do Índice.

Abstract

The world of information retrieval revolves around web search engines. Text search engines are one of the most important source for routing information. The web search engines index huge volumes of data and handles billions of documents. The learn to rank methods have been adopted in the recent past to generate high quality answers for the search engines. The ultimate goal of these systems are to provide high quality results and, at the same time, reduce the computational time for query processing. Drawing direct correlation from the aforementioned fact; reading from smaller or compact indexes always accelerate data read or in other words, reduce computational time during query processing.

In this thesis we study about using learning to rank method to not only produce high quality ranking of search results, but also to optimize another important aspect of search systems, the compression achieved in their indexes. We show that it is possible to achieve impressive gains in search engine index compression with virtually no loss in the final quality of results by using simple, yet effective, multi objective optimization techniques in the learning process. We also used basic pruning techniques to find out the impact of pruning in the compression of indexes. In our best approach, we were able to achieve more than 40% compression of the existing index, while keeping the quality of results at par with methods that disregard compression.

KEY-WORDS: Multi-Objective Optimization, Linear Convex Combination, Weighted Geometric Mean, Pareto Evolutionary Algorithm, Index Compression

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Search engines are among the main sources of information on the Web. These systems receive huge number of queries, while having to deal with massive datasets at the same time. As per current statistics by *internetlivestats*[1] on an average, Google now processes over 40,000 search queries every second, which translates to over 3.5 billion searches per day and 1.2 trillion searches per year worldwide. Besides catering to large number of queries, the web search engines also have to deal with the huge size of the world wide web. As quoted by *WorldWideWebSize*[2], the estimated size of Google's global index is little over about 45 billion web pages. The fundamental aspect of these systems is the quality of their results, especially the first ones. Users rarely tend to go beyond the first page of results [33], in the best case issuing new queries (increasing the load of the system) or in the worst case trying a competitor system. Moreover, modern search engine users are used to experience fast query processing times, regardless of the size of the dataset, and any noticeable increase in waiting time can be a fatal blow to the their perception of the quality of the system.

[1] *http://www.internetlivestats.com/*
[2] *http://www.worldwidewebsize.com/*

A principal aspect of modern search engines is the use of a large number of distinct sources of relevance evidence. Relevance evidence can be defined as the features which can be extracted from the pages of text documents which collectively makes the document relevant to the query furnished. In other words everything from the page which can contribute to the ranking of the page can be called relevance evidence. Examples of such sources of evidence are frequencies of terms, urls, titles, and other parts of speech, term frequency-inverse document frequency metrics, web link graph analysis, use of anchor text of the received links of a web page, their HTML structures, such as titles and headings, url tokenization etc [1]. The distinct sources of relevance evidence are used in order to compute the query result rankings. To do so, they need a method to unite all those sources to calculate the best location for each answer in the final ranking. In the past few decades, most of the work on this fusion of sources has been done with the deployment of learning to rank methods such as RankBoost [14], Genetic Programming based methods [5],[34], RankSVM [18] and other learn to rank methods like [39]. These methods use examples of queries, answers and their selected features (sources of relevance evidence regarding to specific queries), to train a supervised learning model to determine the relative position of the results of a query. After training, the model can be used during query processing to determine the final results ranking. This approach, however, inadvertently adds computational costs to query processing, which may lead to a drop in performance.

An alternative approach, Learn to Precompute Evidence Fusion (LePrEF) [3], also based on supervised learning, proposes to implement the bulk of the feature fusion during indexing time, generating a single inverted index containing unified entries regarding all features, called Unified Term Impacts (UTIs). Contrary to the traditional systems it doesn't maintain separate inverted indexes for each relevance evidence, rather creates a single inverted index containing fused values for all relevance evidences. An UTI can be seen as a combined value or score of all features, which are available during indexing, of a word in the text. This largely helps to reduce the overload of the use of multiple sources of relevance evidence in query processing. The results of LePrEF [3] show that the method achieved a ranking quality on par with the state-of-the-art while leading to faster query processing times.

The results attained by the authors of LePrEF [3] were at par with those of the baselines regarding quality and precision. The quality result of LePrEF with real UTIs was closely behind the GP learn to rank method (with selection procedure as per the authors of LePrEF) and RankBoost [14], and at par with RankSVM [18], whereas the result of LePrEF with integer UTIs was at par with AdaRank [41] and slightly behind RankSVM [18]. The quality achieved by LePrEF with real UTIs was obviously better than that of LePrEF with integer UTIs. This is expected, as real numbers, being floats, would contain more information than integer UTIs which are actually the truncated version of the real numbers. Although there is a loss of quality from real to integer UTI versions of LePrEF, the loss is within acceptable limits to bring this method at par with the baselines. The loss of quality is being compensated by the efficiency of LePrEF, as they were much faster in response as compared to the baselines. Further experiments have uncovered a minor bug in the existing code. The correction of the existing code led LePrEF quality metric to improve a great deal, and thus, with the current level of quality and as much more expressive performance gains over other baselines, LePrEF opens doors for further research in this direction.

As we have already mentioned earlier regarding the enormous sizes that the world wide web can attain, the principal concern is the technology that should be used to handle the needs. The omnipresent solution to this are the inverted indexes which have been efficiently catering to the needs of information retrieval systems for decades. Inverted indexes are used to efficiently access text across such huge data collections. The size of the inverted indexes also turn out to be huge since the data itself is massive. Compression thus can be adopted as the next step in the optimization of the systems. It has a threefold advantage over information retrieval tasks. The primary benefit of adopting compression of indexes is imminent enough, which is reduction in disk usage or storage. Very basic compression schemes have proven to be useful by achieving around 75% or more compression in disk storages [25].

The secondary advantage of compression is in the increased usability of caching during query processing [25]. A very common strategy for search systems to quicken the query processing is to cache the entries of frequently used terms. While the entries of a one term query are cached, all response and ranking related computations for the query are carried out in memory, and the number of disk seeks is reduced. When compression is applied in such scenarios, more information can be processed in main memory. Thus, the number of disk seeks is reduced and in turn compensated by decompression of data in the memory, which has been proven to be more than effective with various high efficiency data compression techniques available in the literature. The penalty for decompression of inverted lists in the memory is much smaller than the disk seeks and hence is preferred. Such arrangements have substantially reduced query processing costs.

Compression also helps in faster disk to memory data transfer [25]. The time taken to transfer a compressed data chunk from disk to the memory and decompress it, combined, is much lesser than the time taken to transfer the same data chunk in an uncompressed format. Thus, it is possible to load much smaller compressed postings or inverted lists and decompress them in the memory in a much faster time, even when they need to be decompressed to carry out ranking related computations. Recent researches have comprehensively proven that information retrieval from compressed indexes are much faster than their uncompressed counterparts [49].

While the creators of LePrEF [3] show that it can lead to high quality results, their experiments also show that there can be a large variation in the compression rate of the indexes created by it. The authors implemented variable size encoding schemes such as Elias-δ [9] to compress the indexes. Search engines usually compress their indexes in order to be memory efficient, and to produce indexes that allow for high compression rates is an important feature. LePrEF uses only quality as its objective function and as such may produce non-compressible indexes, with properties that are not desired in search engines. An example is the necessity of producing index entries that are compression friendly, in the sense that they need to allow high compression rates.

In an extension to the work of LePrEF [3] we propose MOL, a Multi-Objective variant of LePrEF, a method that is not only able to find great ways to fuse evidences, but also can be used to maximize other objectives, which in this case, is the compression rate of its generated indexes. In this work, we study the use of various multi-objective optimization methods and attain a quality level on par with our current baselines and LePrEF [3], while being able to maintain good compression levels also. We continue using the same platform as that of LePrEF [3] and same dataset LETOR [24]. We adopted Genetic Programming(GP) in our implementation since the authors of LePrEF [3] have used GP successfully in developing the method and it challenges most baselines convincingly. We would like to mention here that, other traditional learn-to-rank methods like RankBoost [14], RankSVM [18] and AdaRank [41] can also be used here to implement a multi-objective approach.

Our research on the feasibility of multi-objective optimization dwells on the applicability of multiple objectives in the execution of LePrEF [3]. As stated before we used compression as an auxiliary objective to achieve multi-objective optimization, along with the principal objective, quality, of the search system. We have implemented Elias-γ [9] codes to compress the indexes generated by Multi-Objective-LePrEF (MOL), but other alternative compression methods can also be easily applied on this scenario. We should emphasize on the fact that the principal objective of this work is to achieve higher compression of inverted indexes without sacrificing any quality of the search results.

In our research we also implemented pruning of indexes to assist in compression and thereby improve search engine efficiency. Pruning can be considered as a lossy compression technique of the search indexes. Compression already reduces the index sizes, and now an efficient pruning should reduce the index size greatly. The objective of such pruning strategies is to reap the benefits of index compression while maintaining the loss of information to an acceptable limit. As discussed before the readable data is indexed and impacts related to response is stored in inverted lists or postings. Pruning techniques are directed to these lists which are responsible for measuring the relative importance of the individual terms in the documents or queries. The terms which eventually have impacts on the ranking results or, in other words, which are relevant to the search query, are maintained in the index, whereas the ones which do not contribute to the ranking are discarded.

Various pruning techniques in the literature, for example [7],[6], perform careful scrutiny of every term for the provision of useful information, and then determine which terms should be in the index and which should not.

Through our work we came to know about the non-dependency of quality of ranking and compression rate of inverted indexes on each other during the implementation of multi-objective optimization. At first glance, it was expected that quality on ranking and compression rates of inverted indexes were contradictory features for a textual search system. During the course of research it surfaced that, what appeared at first glance, was not entirely correct. We were actually able to generate some very high quality indexes which also compress well.

The remainder of this dissertation is organized as follows: In Section 2 we discuss the related works and an overview of the basic concepts involved. Section 3 describes the proposed method describing how we use LePrEF [3] with multi-objective optimization. Section 4 presents the experimental setup followed by experimental results in comparison to current baselines in Section 4.3. Finally, Section 5 presents the conclusive remarks and directions to future research.

Chapter 2

Background and Related Work

In this chapter we present some of the basic concepts to understand the paradigm adopted in our experiments and the corresponding related works in the literature.

2.1 Application of Machine Learning

Machine learning is a broader application of *Artificial Intelligence* where it studies the ability of the computers to learn without them being programmed explicitly.This field of computation is a machine implementation of how human behavior is towards a situation. It learns from experience and applies it to the current situation. The input to such systems is data, in quantities, which depends on specific scenarios, which is then used to learn *patterns* present in it. These *patterns*, which are based on the features which generated the data; can be leveraged to make predictions on unknown data sets. Machine learning thus generates models which can learn patterns from sample data and use the knowledge gained to do the required job. Owing to its remarkable success, machine learning is used in wide range of fields like medical diagnosis, fraud detection, information retrieval, natural language processing and others.

The machine learning algorithms can be subdivided into three major groups: *Supervised Learning*, *Unsupervised Learning* and *Semi-supervised Learning*, based on the type of learning that is being adopted [1]. *Supervised learning* can be described as a model which is provided with input data, known as *training data*, labeled by human data specialists. *Unsupervised learning*, can be explained as that class of learning where none of the *training data* is labeled. In *Semi-supervised learning* a small amount of labeled data is used to make predictions about large amounts of unlabeled data. The authors of LePrEF [3] deployed supervised techniques for their implementation and so is being followed by us.

In LePrEF [3], the authors used an application of supervised learning. The implementation of this is little different from classification of elements, which is a principal application of supervised learning. Here, *learning to rank* methods are adopted, where a fitness score is generated for each element and then all elements are ranked in descending order of their scores. The best elements based on specific criteria are then selected for further processing; this paradigm being known as *Genetic Programming*.

Learn to Pre-Compute Evidence Fusion & Genetic Programming

A key concept in any search engine architecture is relevance evidence. We have already talked about it in the Introduction. We define relevance evidence as any information which can be extracted from the documents in the data-set, e.g. in Web search engines its pages' content, anchor text, url, title, image description, etc. that might contribute to infer that a webpage is relevant to the user's query.

This information is translated into numerical values and stored in search engine indexes, and can be roughly divided into two sets: Query dependent and query independent. The former is relevance information that depend on the query terms in order to infer webpage relevance, and is usually stored in inverted indexes, with one entry per term present in the webpage (the concept of term might vary according to the retrieval model adopted, with single words used as terms in traditional bag-of-words based models). The latter is usually information pertaining webpage characteristics, such as PageRank values, number of received links, number of slashes in the URL etc.

On the contrary to the traditional textual search systems LePrEF [3] adopts the concept of a single numerical value to represent the whole set of relevance evidences, both query dependent and independent, storing Unified Term Impacts (UTIs) in the inverted index. Instead of storing several values for each isolated source of evidence it fuses the sets of index entries for all relevance evidences into a single value. While in the traditional search systems, we have seen that after the extraction of features from the documents/-pages the documents are indexed to generate separate inverted indexes for every relevance evidence, LePrEF [3] involves another step at this stage, where it fuses all available relevance evidences into a single value called UTI and generates an inverted index of UTIs. At this stage, the cost of the processing in LePrEF [3] is high but these costs never impact the users' experience as it occurs during indexing phase. The next stage is query processing where on one hand traditional systems followed a two-phase query processing with sophisticated ranking functions like BM25 [32] to generate rankings, on the other hand LePrEF [3] simply has to add the UTI scores to generate the ranking. Thus, it makes faster query processing at the cost of high indexing costs.

The authors of LePrEF [3] have used Genetic Programming [GP] as a tool to learn unified precomputed impacts from various distinct sources of relevance evidence. In the recent past many researchers have used GP successfully in the fields of text search and learn to rank functions [5], [10], [12], [34], [39], [43]. As compared to the works mentioned here, LePrEF [3] used GP to learn to fuse separate indexes of different relevance evidences into a single inverted index, and that too during indexing, unlike earlier approaches where the learning was applied at the query processing time. The inspiration to use GP as a tool to learn weights of each term of a document is attained from [12]. GP also proved to be very useful in [11] where it automatically generates ranking strategies for different contexts.

Various learning methods have been used successfully applied to Information Retrieval problems such as [5], [10], [14], [18], [39]. All of the aforementioned cases showed promising results but they used the learning function during query processing. The results of [5], [34] nurtured our belief in GP as they achieved at par results in search systems for textual databases. Further GP uses mathematical operators which result in simpler computation of impacts for document terms. To provide more insight regarding the GP process and how it is used in LePrEF [3] to generate the UTIs, we explain the basics of it in this section.

Genetic Programming framework as depicted in listing 2.1 follows an iterative process over a given number of generations and individuals until it generates the best individual based on certain end criteria. In case of our implementation the end criterion is the maximum number of generations of the evolutionary process. The GP process runs in two phases: *training* Lines(1-10) and *validation* Lines(11-14). In each of these phases a set

Listing 2.1: General GP process used both in learn to rank and in LePrEF [3]

```
1  Let $\mathcal{T}$ be a training set of queries ;
2  Let $\mathcal{V}$ be a validation set of queries ;
3  Let $N_g$ be the number of generations ;
4  Let $N_b$ be the number of best individuals ;
5  $\mathcal{P} \leftarrow$ Initial random population of individuals ;
6  $\mathcal{B}_t \leftarrow \varnothing$;
7  **For** each generation $g$ of $N_g$ generations **do** {
8      $\mathcal{F}_t \leftarrow \varnothing$;
9      For each individual $i \in \mathcal{P}$ do
10         $\mathcal{F}_t \leftarrow \mathcal{F}_t \cup \{g, i, fitness(i, \mathcal{T})\}$;
11     $\mathcal{B}_t \leftarrow \mathcal{B}_t \cup getBestIndividuals(N_b, \mathcal{F}_t)$;
12     $\mathcal{P} \leftarrow applyGeneticOperations(\mathcal{P}, \mathcal{F}_t, \mathcal{B}_t, g)$;
13 }
14 $\mathcal{B}_v \leftarrow \varnothing$;
15 **For** each individual $i \in \mathcal{B}_t$ **do**
16     $\mathcal{B}_v \leftarrow \mathcal{B}_v \cup \{i, fitness(i, \mathcal{V})\}$;
17  BestIndividual $\leftarrow applySelectionMethod(\mathcal{B}_t, \mathcal{B}_v)$;
```

of queries and documents are selected from distinct collections, which are called *training set* and *validation set*[5].

The GP process starts with the creation of a random population of individuals (Line 5), which are essentially mathematical expressions derived from the combination of relevance evidences. This random population starts evolving with every passing generation using genetic operations like reproduction, mutation and crossover until some specified stopping criterion is met (Line 12), which in this case is the number of generations. In the training phase each training individual is assigned a score by the fitness function (Line 7-10) and the individuals are ordered as per their fitness scores. The fittest individual is then allowed to evolve (Line 11). The next step is to use the best individuals from training on the *validation set* of data to remove *overfitting* (Line 15-16). The fittest individual from this phase is considered as the best individual (Line 17).

In LePrEF [3] a learn to rank approach is used, wherein the individuals which are basically, the evidence fusion functions, generate an UTI value for each term in the inverted index. The training queries are then processed, based on these UTI values to generate a ranking for the *training set*. Thus, for every term occurring in the data collection an UTI value is generated with respect to the document where the term exists. The sum of the UTIs for query terms are calculated with respect to each document to rank the all documents in the collection. The final ranking, hence, is then used to calculate the fitness of the individuals.

Individual

In genetic programming, an individual can be represented as a mathematical function organized in a tree structure as shown in 2.1. These mathematical formulas can be obtained from the various ranking formulas in the *information retrieval* literature. In the experiments of LePrEF [3] the authors used the values of the features as mentioned in section 4.1, which are the information obtained from the relevance evidence, for the terminals or the leafs, also combining them with random number constants ranging from 0 to 100. In the inner nodes, the authors use division(/), multiplication(*), addition(+) and logarithm(log) as corresponding mathematical functions.

Fitness Function

The fitness function is designed so as to measure the quality of rankings generated by the individuals. Normalized Discounted Cumulative Gain (NDCG) [17] and Mean Average Precision (MAP) [1] were used by the authors of LePrEF [3] in their initial experiments,

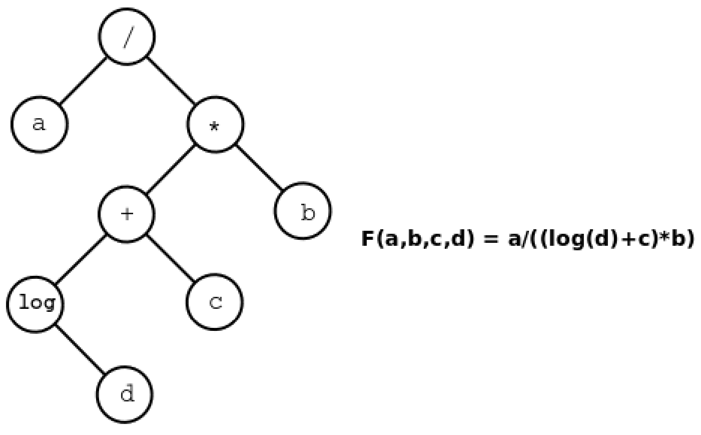

Figure 2.1: A sample individual of GP process

but later the best experiments which resulted in better quality, used mean NDCG [17] as the fitness function. NDCG measures the performance of a recommendation/retrieval system based on the graded relevance of the recommended entities. It varies from 0.0 to 1.0, with 1.0 representing the ideal ranking of the entities.

Selection of the Best Individual

The selection of the best individual is done from the *validation* of LePrEF [3] as suggested in [5]. In this method both the training and validation performance is being taken into consideration while calculating the final performance. For example if t_i is the training performance and v_i is the validation performance of the individual i, then the final performance f_i is $f_i = (t_i + v_i) - \sigma_i$, where σ_i is the standard deviation of t_i and v_i. The best individual is thus given by:

$$\underset{i}{argmax}((t_i + v_i) - \sigma_i) \tag{2.1}$$

This method is known as AVG$_\sigma$, and the individual with the highest value of AVG$_\sigma$ is selected as the fittest individual. The selection procedure for the best individual clearly implies that a smaller value of σ_i would contribute more in the selection of the individual, thereby making it sure that the individuals which perform uniformly over *training* and *validation* data set are selected, whereas $(t_i + v_i)$ stresses on getting those individuals selected, which simultaneously performs very good in both the sets.

As we all know that Genetic Programming is associated with some kind randomness in the process, the authors of LePrEF [3] have run ten processes with different random seeds instead of a single process. Finally the best individual is selected based on the AVG_σ values of the individuals generated over the ten seeds. Through this approach the chances of generating a low performing individual with only a single seed is replaced with an option of multiple seeds where the probability of generating a good individual increases. The results of LePrEF [3] proves the same.

2.2 Compression

A lot of research has been done in the past and present on data compression in inverted indexes. A classical approach to such compression that has yielded consistently good results in this task is done by using *universal codes* to represent index entries, where each integer is encoded with a uniquely decodable variable length code. An interesting fact about these codes is that they are not dependent on the input.

The most common and trusted codes among these are the *Elias Gamma/Delta codes* [9]. The *Delta codes* being little more efficient for compression but loses to *Gamma codes* on account of decompression. *Gamma codes* are the simplest of the Elias codes. For example if a number $x \in \mathbb{N} = \{1, 2, 3, ...\}$ is encoded with Elias Gamma coding scheme, then the encoded number is represented by its binary form preceded by $\lfloor log_2(x) \rfloor$ zeroes. This encoding scheme is quite lucid and we may note that the total number of bits required to encode x are $(2\lfloor log_2(x) \rfloor + 1)$, wherein $(\lfloor log_2(x) \rfloor + 1)$ bits are required for the binary representation of x.

The Elias Delta codes are built on top of *Elias Gamma* and *Binary* representation of integers. Suppose we have an integer $x \in \mathbb{N} = \{1,2,3,...\}$. The first part of the delta code comes from the *Elias Gamma* representation of $(\lfloor log_2(x) \rfloor + 1)$. The rest of the number consists of the binary representation of $(x - \lfloor log_2(x) \rfloor)$, which is coded in $\lfloor log_2(x) \rfloor$ bits. Thus, the total number of bits needed for *Delta codes* are $(\lfloor log_2(x) \rfloor + 2\lfloor log_2(\lfloor log_2(x) \rfloor + 1) \rfloor + 1)$ bits.

Golomb code [15] is run-length encoding also known as variable-length codes. The length of the encoded number changes as per the selection of a variable, which is used in the encoding. To compute *Golomb code* for an integer $x \in \mathbb{N} = \{1,2,3,...\}$, we need to compute three values:

$$q = \lfloor \frac{x-1}{b} \rfloor, \; r = x - qb - 1 \text{ and } c = \lfloor log_2 b \rfloor,$$

where, q is the quotient, r is the remainder and b is a constant based on which *Golomb codes* are generated. The encoding is done in two parts; the first part consists of the *Unary* representation of $(q+1)$, succeeded by the binary representation of b. In the second part r is represented in its *binary* format using c bits for c successive numbers with a most significant bit of 0, and for the rest the same is represented using $c+1$ bits with a most significant bit of 1.

Another example of run-length encoding is *Rice code* [31], which is a very specialized form of *Golomb coding*. This code is preferable when the numbers in the data collection are small. Encoding pattern for *Rice codes* are exactly the same to that of *Golomb coding*, the only difference being $b \in 2^n$ where, $n \in \mathbb{N} = \{1, 2, 3, ...\}$.

We adopt Elias codes in our experiments but it is possible to implement other state-of-art compression schemes like PFOR,PFOR-DELTA and PDICT [50], varint-G8IU [37], NewPFD [42], PFOR2008 [44], Partitioned Elias-Fano Indexes [29] and SIMD-BP128 [23].

2.3 Pruning

Pruning is another strategy which plays an important role in the field of information retrieval. We have already discussed regarding pruning and strategies in the Introduction 1. Literally index pruning techniques remove entries from the inverted indexes which do not contribute to the ranking of the documents. There are more complex strategies for computing those values which needs to be removed from the inverted indexes based on various factors like proximity, relevance and so on. The final goal towards of pruning strategies are to facilitate indexes more towards compression, thereby saving space and time.

In [6] the authors propose a pruning idea which enhances the time and space efficiency without tinkering with the search effectiveness. They have successfully claimed that the experiments have proven 60% cost cutting over reduction in index storage, while keeping the loss in the retrieval precision to a minimum. The have also proven that indexes were 88% compressed while comparing with uncompressed indexes. To achieve this feat they have used two kinds of indexes: *The frequency index* and *the positional index*. The *frequency index* contains the frequency of the term *t* for all documents where it exists, which is used to measure the relative importance of terms in documents and queries. The *positional index* contains the information of the position of occurrence of the term *t* in those documents and thus facilitate processing positional queries.

In another approach towards pruning [7] proposed a static index pruning strategy that takes into account the locality of occurrences of words in the text. These strategies are best suited for fast changing document databases such are large web search engines. The authors achieved around 30% reduction of index sizes in their experiments with almost no change in the top query results. Locality information is very important in this approach and a simple approach involving locality can be very well competitive to more complex strategies that do not rely on locality information.

2.4 Multi-Objective Optimization in Genetic Programming

The Multi-objective Optimization problem (also known as multi-criteria, multi-performance or vector optimization problem) can be defined as the problem of finding [28]:

a vector of decision variables which satisfies constraints and optimizes a vector function whose elements represent the objective functions. These functions form a mathematical description of performance criteria which usually conflict with each other. Hence, the term "optimize" means finding such a solution which would give the values of all the objective functions acceptable to the decision maker.

Multi-objective optimization in genetic programming is a very active field of research in the last decade or more. Generally, in a multi-objective optimization problem, multiple objectives, or goals, $f_1, f_2, ..., f_n$ are optimized(maximized or minimized) to find a range of solutions. These solutions are essentially and necessarily acceptable for all criteria simultaneously. The problem arises when these objectives or criteria, directly or indirectly oppose each other while we try to achieve our goal, thereby reducing the overall performance of the system. Multi-objective optimization algorithms offer specialized care in these situations, thus, achieving optimal results.

A general multi-objective optimization problem can be described as a vector function f that maps a tuple of m parameters (decision variables) to a tuple of n objectives, formally [47],[36],[13]:

$$\begin{aligned} &\text{minimize/maximize} && \mathbf{y} = f(\mathbf{x}) = (f_1(\mathbf{x}), f_2(\mathbf{x}), ..., f_n(\mathbf{x})) \\ &\text{subject to} && \mathbf{x} = (x_1, x_2, ..., x_m) \in X \\ & && \mathbf{y} = (y_1, y_2, ..., y_n) \in Y \end{aligned} \tag{2.2}$$

where, $\mathbf{x}$ is called the *decision vector* and $\mathbf{y}$ is called the *objective vector*, and $n \geq 2$.

We have mentioned the term "*optimal*" earlier in our discussion regarding multi-objective optimization. This actually means some value for the *decision vector* which is either maximum or minimum, based on requirements. In this case we have multiple optimal solutions leading to a set of solutions. Such a set of solutions consists of all *decision vectors* which cannot be improved in any objective without degrading any other *decision vector*. These solutions or vectors are known as *Pareto Optimal*. Pareto im-

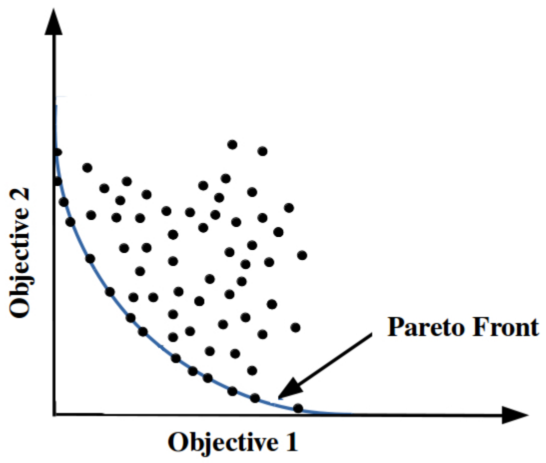

Figure 2.2: Example of pareto distribution of solutions with two objective functions.

provement consists of improving a solution without making any other solution worse, and when we reach the optimal solutions, then, no further improvements are possible. Pareto optimality can be formally proposed as [47]: Suppose we have a minimization problem with two *decision vectors* $a, b \in X$ with n objectives. We state that a dominates b or $a \succ b$ iff

$$\forall i \in \{1, 2, ..., n\} : f_i(\mathbf{a}) \leq f_i(\mathbf{b}) \quad \wedge \quad \exists j \in \{1, 2, ..., n\} : f_j(\mathbf{a}) < f_i(\mathbf{b}) \tag{2.3}$$

The set of optimal solutions or the best solutions from the search space, which is also known as the non-dominated solutions, constitutes the pareto frontier of the collection. Figure 2.2 shows an example of a pareto frontier and the distribution. In the figure 2.2 we show a minimization scheme for optimization and hence the curve is concave, whereas this can be put forth for a maximization of the objectives also, and in that case the curve in the figure would be convex.

A very common idea to use multi-objective optimization with genetic programming is to combine several fitness functions or objectives by using an aggregate scalar fitness function. Suppose we have n objective functions $f_1, f_2, ..., f_n$, then we can can use linear combination of the form $f = \Sigma_{i=1}^{n} w_i.f_i$, where $w_1, w_2, ..., w_n$ are appropriate constants. A normalized form of the same idea is to use weights $w_i \ \exists \ \Sigma_{i=1}^{n} w_i = 1$.

Langdon and Poli [22] proposed a semi-linear combination of fitness and speed to improve the performance of genetic programming on the Santa Fe trail ant problem. A threshold was applied there to limit the impact of speed to avoid an excessive bias for the ants which were fast but not able to complete the trails.

Langdon, Barrett and Buxton [21] also used linear combination of two related objectives, the sum of squared errors and the number of hits, where a hit is a case in which the error is below a pre-defined threshold value. This was used for biochemical predictions in pharmaceutical industry.

Zhang and Bhowan [45], O'Reilly and Hemberg [27] and Koza, Jones, Keane, and Streeter [20] also present similar ideas of multi-objective optimization to improve genetic programming performance by doing linear combination of multiple objectives with carefully chosen weights, which were derived only after considerable experimentation.

Another way of calculating an aggregate of multiple objectives or fitness functions is to calculate the weighted geometric mean of the objectives. Suppose we have n objectives $f_1, f_2, ..., f_n$ and $w_1, w_2, ..., w_n$ as the corresponding weights assigned to each objective, then the geometric mean of the objectives can be computed by

$$f = \sqrt[W]{(f_1^{w_1}.f_2^{w_2}...f_n^{w_n})}\ , where\ W = \Sigma_{i=1}^{n} w_i$$

Iba, Paul and Hasegawa [16] use equal weights for two objectives, sensitivity and specificity to compute the geometric mean based fitness as

$$G_{fitness} = \sqrt{(sensitivity.specificity)}$$

In an alternative approach the objectives can also be kept separate while they are being optimized. The prime idea in multi-objective optimization here is based on *pareto dominance*. Given a set of objectives to minimize, for instance, a solution is said to *pareto dominate* another, if the first is not strictly inferior to the second in all objectives and is superior to the second in at least one objective. Zitzler et al.[48], [46] and Deb et al.[8] used similar approaches to propose evolutionary algorithms which are currently used widely in various multi-objective approaches in a wide range of applications.

Strength Pareto Evolutionary Algorithm (SPEA2):

One of the most successful approaches to multi-objective optmization is the second version of the Strength Pareto Evolutionary Algorithm (SPEA2) [46]. The objective of SPEA2 is to locate and maintain a front of non-dominated solutions, ideally a set of pareto optimal solutions. Finding pareto optimal solutions is an strategy adopted while combining conflicting objectives. At a first glance, finding a solutions that provides good compression and high quality might be two conflicting objectives, since as compression rates grow, less information is present to generate the rankings. Thus, using a pareto based approach could be a natural option for combining such objectives.

The SPEA2 is an evolutionary process which uses genetic operators such as reproduction, crossover and mutation to explore the search space. The selection process uses a combination of the degree to which a candidate solution is dominated (strength) and an estimation of density of the pareto front as an assigned fitness. The non-dominated set of individuals is maintained separately from the population of candidate solutions used in the evolutionary process, providing a form of elitism. More details about SPEA2 can be found in the work of Zitzler et al. [46]. The SPEA2 algorithm is described briefly below:

Input : ***N*** *(population size),* ***T*** *(Maximum number of generations)*

Output: ***A*** *(nondominated set)*

Step 1: Initialization: Generate an initial population P_0. Set t=0.

Step 2: Fitness Assignment: Calculate fitness values of individuals in P_t. Each individual i in population P_t is assigned a strength value $S(i)$, representing the number of solutions it dominates.

$$S(i) = |\ \{j \mid j \in P_t \wedge i \succ j\}\ |$$

where, $\succ$ corresponds to the pareto dominance relation. Based on the values of S the raw fitness $R(i)$ of an individual i is calculated:

$$R(i) = \sum_{j \in P_t, i \succ j} S(j)$$

Fitness here needs to be minimized and a value of 0 corresponds to the non-dominated individuals. When there are many individuals which do not dominate each other. Density Estimation is used to assign different values to these individuals. The density estimation is done using k^{th} nearest neighbour method proposed by Silverman [35]. The value of k is given by:

$$k = \sqrt{N}$$

For each individual i the distances (in objective space) to all individuals j in population are calculated and stored in a list. After sorting the list in increasing order, the k^{th} element gives the distance sought, denoted as σ_i^k. Thus, density is given by:

$$D(i) = \frac{1}{\sigma_i^k + 2}$$

Finally $D(i)$ is added to the raw fitness value $R(i)$ of an individual i to give the fitness $F(i)$:

$$F(i) = R(i) + D(i)$$

Step 3: Environmental Selection: The concept of allowing the best individuals to go through to the next generation untouched is known as *elitism*. We apply reproduction to x percent of the population to select the best individuals from the population and promote them to the next generation. The new generation thus now has only x percent individuals, which are the best from the previous generation. Formally we can present this as:

$$P_{t+1} = \{ j \mid j \in P_t \wedge F_j(i) < 1 \}$$

where, j is the top x percent of the population P_t. The rest of the individuals in the new generation are generated from the new population by crossover and mutation, thus generating P_{t+1}. The value of x is carefully chosen after considerable experiments.

Step 4: Termination: If $t \geq T$ or any other stopping criterion is met then set A is said to be the set of decision vectors represented by the non-dominated individuals in P_{t+1}.

Owing to the pertaining discussion about multi-objective optimization we were encouraged to implement the idea of having multiple objectives during the indexing of LePrEF [3] which when combined with compression yielded great compression rates with very little or no change in the quality of rankings. This approach is an innovative direction to multi-objective optimization in the field of textual information retrieval where compression friendly quality indexes are very important.

Chapter 3

Multi-Objective Learn to Precompute Evidence Fusion

LePrEF [3] is a Learning to Rank method that is based on the idea of fusion of different sources of relevance at indexing time. To do so, it introduced the concept of Unified Term Impact (UTI), which is a single numerical value representing the impact that a term has in a document, taking into consideration all sources of relevance evidences.

To do this fusion, it uses Genetic Programming (GP) in order to generate individuals (mathematical formulas) to combine all sources of relevance evidence into a single value for each term-document pair. Then, the quality of the rankings generated by UTIs created by each individual is used as this individual's fitness value. After finding the best individual during the training stages, the whole dataset has to be indexed according to that individual, generating an UTI inverted index. In the UTI inverted index we have an inverted list for each term. The inverted list of the term is composed of pairs consisting of document id and corresponding UTI value, for all documents in the collection where the term occurs. A document id is an alphanumeric variable which is used to index the documents in the collection. Thus while using a single UTI inverted index, the query processing is just a simple matter of list traversals and accumulator sums, instead of multiple indexes being traversed followed by a fusion step, as occurs in traditionally indexed datasets. This advantage leads to a reduction in the number of operations performed by the query processor since it is, in practice, dealing with a single and unified representation of the sources of relevance evidence adopted.

We propose Multi-Objective LePrEF (MOL) on the similar lines, a variant of the LePrEF [3] method that can take into account more than one (which in this case is *“quality of ranking"*) objectives while generating solution. In this section we will describe MOL, with emphasis on using both the quality of the ranking and an assumed compression rate as training objective. This is further realized by means of various optimization schemes to fortify our proposal.

3.1 Introducing Compression as an Objective

In search engines, it is usual to store the impacts of the terms as integers in inverted lists, and a similar approach can be used for UTIs. The entire index produced by LePrEF [3] requires the storage of one UTI for each term that occur in each document of the dataset, and such task usually requires a large amount of storage space. Thus, in order to minimize the impact that those potentially vast datasets might have on the search system, it is usual to implement compression schemas in the indexed data, like *Elias-*γ/δ *codes* [9] or *Golomb/Rice codes* [15],[31] to encode the integer entries.

As in LePrEF [3] we adopted *Elias-*γ [9] coding in our experiments. Since we are interested in adding the compression potential of UTI indexes as an objective function in the learning process, we adopted the compression rate of data as one of our objective functions. While processing each generation in the GP process, MOL computes the NDCG[17] values computed by each individual over a set of training queries, alike LePrEF [3].

While the system is indexing the data collection and computing the UTI values to generate the inverted index of UTI values, it calculates the compression rates by using *Elias-γ codes*, and the compressed size of the index, thereafter. In the next step the relative difference between the size of the uncompressed and compressed UTIs gives the space savings or Compression Rate:

$$CR = \frac{UTI\ size_{initial} - UTI\ size_{final}}{UTI\ size_{initial}}$$

where, $UTI\ size_{initial}$ is the size of the inverted index before compression and $UTI\ size_{final}$ is the final compressed size of the inverted index of UTIs. Note that the index does not need to be compressed in order to compute the compression rate, since we can compute the size of an Elias code for any given number.

The value of Compression Rate(CR) varies from 0.0 to 1.0, where 0.0 denotes “no compression" and 1.0 representing theoretical “100% compression". A one hundred percent compression, though means complete loss of information. Notice that it is also possible to achieve negative compression, where the compressed data occupies more space than the uncompressed.

3.2 Combining Quality and Compression

In order to combine both Quality and compression as our objective functions in Multi-Objective LePrEF, we experimented with three different techniques discussed in Section 2: Linear Convex Combination, Weighted Geometric Mean and Strength Pareto Evolutionary Algorithm (SPEA2) [46]. In all cases, the values of NDCG [17] and compression rate were adopted to represent how good is each individual regarding the quality of ranking and compression, respectively.

Formally this is a typical multi-objective optimization problem where we would need to maximize the *objective vector*. We have two objectives, quality of ranking and compression of inverted index, represented by q and cr respectively. While we use genetic programming, we have a *decision vector* with m parameters or decision variables, and n contradicting objectives. Thus, the problem can be stated as:

$$\begin{array}{ll} \text{maximize} & \mathbf{y} = f(\mathbf{x}) = (f_q(\mathbf{x}), f_{cr}(\mathbf{x})) \\ \text{subject to} & \mathbf{x} = (x_1, x_2, ..., x_m) \in X \\ & \mathbf{y} = (y_q, y_{cr}) \in Y \end{array} \tag{3.1}$$

where, $\mathbf{x}$ is the *decision vector* and $\mathbf{y}$ is called the *objective vector*, and $n \geq 2$. Thus, in our case we have a maximization problem with the objective vector as shown above and also $n = 2$ (two objectives: quality and compression). We have k *decision vectors*, where $k \geq 1$. Thus, if we have $\{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_k\} \in \mathbf{X}$ as the decision vector, we state that $\mathbf{x}_1$ dominates $\mathbf{x}_2$ or $\mathbf{x}_1 \succ \mathbf{x}_2$ iff

$$\forall i \in \{1,2,...,n\} : f_i(\mathbf{x}_1) \geq f_i(\mathbf{x}_2) \quad \wedge \quad \exists j \in \{1,2,...,n\} : f_j(\mathbf{x}_1) > f_j(\mathbf{x}_2) \tag{3.2}$$

Where, the aforementioned *pareto optimality* is checked between each of the *decision vectors* to lead to the best set of pareto optimal solutions.

Linear Convex Combination

One of the most simple alternatives for multi-objective optimizations that we have experimented is linear convex combination. The values for quality of ranking and compression rates of inverted indexes are calculated by each individual for set of queries and combined together, leading to a fitness function containing a mixture parameter w dictating the ratio of importance given to the quality of ranking and the compression rate of inverted index:

$$score_{fitness} = w.q + (1-w).cr \tag{3.3}$$

where $0 \leq w \leq 1$, q represents the quality of ranking produced by the system using the best individual and cr is the compression rate of the UTI inverted index using the same individual.

The fitness score computed as shown here is used in the method to compute the fitness of each individual while it processes the training queries. Based on the score of the individuals the best ones are selected, and those individuals are used to generate the population for the next generation. This score is used to rank the GP individuals in the validation phase similarly, and finally the best individual is selected from the combination of scores of training and validation phase.

Weighted Geometric Mean

In another alternative approach, we also propose to use a weighted geometric mean between quality and compression, since it takes into account both mean and variance of a set of values. We implemented weighted geometric mean to calculate the fitness of an individual as:

$$score_{fitness} = \sqrt[c]{(q)^a.(cr)^b} \tag{3.4}$$

where, c is the sum of a and b; a and b are the values of the weights assigned to quality of ranking and compression rates, respectively. The variables q and cr represent the values of quality of ranking and compression rate for the inverted list, using the best individual for that generation. When the weights assigned to the two objectives are equal, it becomes *Geometric Mean*. The fitness scores computed here by using geometric weighted mean is used in the same manner as it is used while using Linear Convex Combination.

Strength Pareto Evolutionary Algorithm (SPEA2)

The third alternative adopted by us is the Strength Pareto Evolutionary Algorithm (SPEA2) [46]. We implemented a fast algorithm [26] to find the non-dominated solution sets 3.1. Unlike the methods like Linear Combination or Geometric Mean, SPEA2 [46] does not rely on any scalar values based on the fusion of the values of the objective functions. It uses the

Listing 3.1: Fast algorithm to find Non-dominated Solutions [26]

```
Sort all the solutions $(P_1, P_2, ..., P_N)$ in decreasing order of their first objective
    function $(F_1)$ and create a sorted list $(O)$
Initialize an empty set $S_1$ and add the first element of the list $O$ to $S_1$
For every solution $O_i$ (other than first solution ) of list $O$, do
    Compare solution $O_i$ with the solutions of $S_1$, for both first $(F_1)$ and second
        $(F_2)$ objectives
        If any element of set $S_1$ dominate $O_i$
            then delete $O_i$ from the list
        If $O_i$ dominate any solution of the set $S_1$
            then delete that solution from $S_1$
        If $O_i$ is non dominated to set $S_1$
            then update set $S_1 = S_1 \cup O_i$
        If set $S_1$ becomes empty
            then add the immediate solution to $S_1$
Get non−dominated set $S_1$.
```

aforementioned concept of pareto optimality to determine the non-dominated solutions.

In this implementation we use the same basic infrastructure as of LePrEF [3] and introduce the SPEA2 [46] code. The values of quality of ranking(NDCG [17]) and compression(using Elias-γ [9] codes) for the training queries are processed using SPEA2 [46] as the selection function, to generate the best individuals for each generation. The fittest individuals from the training phase are used to process the validation dataset, again using SPEA2 [46] to get the best individual in each generation.

Although we have decided to experiment with SPEA2, we notice that compression and high quality ranking are in fact not necessarily conflicting objectives. Our experiments show that in fact it is possible to add compression in the goal with almost no loss in quality of results. Due to this non-conflicting property of the two objective functions studied, the results achieved with SPEA2 were in fact not superior to the ones achieved by the other two simpler combination approaches adopted, the linear convex combination and the weighted geometric mean. There were cases where, for a solution, both quality and compression are good, or both are worse. So there may not be a direct tradeoff between these features. We decided to report the results with SPEA2 anyway, given the importance of this approach and to report that this possible solution to the problem was considered in our research.

Our goal with the combination of objective functions is to achieve an UTI index produced by LePrEF with as large a compressibility as possible, but without any or with very little loss of quality in the ranking results produced by it.

Chapter 4

Experiments

In this section, we present the experiments done to study the behavior of Multi-Objective LePrEF (MOL) as compared to the original LePrEF regarding the quality of its query results and the compression of indexes generated by it.

4.1 Datasets

We used LETOR benchmark [24] to evaluate the quality of ranking results produced by MOL similar to its predecessor LePrEF [3]. It was initially created from the GOV2 document collection which contains roughly 25 million web pages. Precisely, we have chosen MQ2007 subset of LETOR4 as it has large number of queries. MQ2007 has 1692 queries.

Every query-document pair in this dataset is represented by 46-feature vector, since the dataset MQ2007 contains 46 features for documents, which are sampled from the top 1000 retrieved documents by using BM25 [32] on the GOV2 corpus for each query. Some of these textual features are available during the indexing time and some can be obtained during query processing only.

For *MQ*2007, we have access to 26 features during the indexing phase. These features are **TF**, **IDF**, **TF** x **IDF** and length in number of words. Each of these features are then applied to five different areas of the documents: the body of the text, the anchor text, the title, the URL, and the whole document, thereby generating 20 features. Apart from these there are six other features: PageRank, InLink Count, OutLink Count, Number of Slashes on URL, Length of URL and Number of Children which are also available during indexing. The details of the features may be looked into in the LETOR documentation [24]. The features **TF**, **IDF**, **TF** x **IDF** are not used in MOL as they are in LETOR; instead of taking a sum of features values obtained with each query term, we compute the individual values of frequency of each query term in each document as features.

LETOR also has features which are not available during indexing as we have already mentioned before. These features are the similarity scores assigned by ranking function BM25 [32], and by three variations of Language Models based functions. Each function was applied to five areas of the documents as mentioned earlier. Thus, they constitute those 20 features which cannot be adopted by MOL.

LETOR was created by using BM25 [32] to select an initial set of documents from the whole data collection as an initial ranking of the best documents. Thus, while using LETOR this initial set of best ranked documents are then analysed upon by applying the whole set of features as discussed early in this section again to generate the final results [4],[2].

We use the results for LETOR4 by LePrEF [3] as the baseline method to compare the quality effectiveness results of MOL.

4.2 Experimental Design

The LETOR [24] is split into 5 folds by default, where 3 folds are used for training queries, 1 fold for validation queries and the other one for test. This allows for a 5 fold cross validation scheme to evaluate the results. In our results we have always presented and used the average of the results at each testing fold.

We have adopted mean NDCG [17], MAP [1], NDCG@N [1] and P@N [1] to compare MOL with LePrEF [3].These are the same metrics which have been used by LETOR [24]. The details about the computation of these metrics can be found in the LETOR documentation [24].

4.2.1 Genetic Programming Settings

We have conducted all experiments presented in this work using the lilGP genetic programming distribution [30]. We have adopted the same parameters adopted by Koza [19] and by LePrEF [3], which is also in line with the parameters proposed in a study that uses GP as learn to rank [5].

As we have already discussed, Genetic Programming uses random seeds to create its initial population and the results can be very well affected by the initialization of the first population. To avoid such a scenario of specificity, where the results are too much dependent of the random seeds, we followed in the footsteps of LePrEF [3] to perform 10 different runs of MOL with distinct random seeds, while doing a five fold cross validation with each run, since the data in LETOR is split in five folds already.

The best individual for each fold is selected, based on the UTI values by using the AVG_σ selection method. The training of the individuals for five folds over ten distinct random seeds may become very expensive in a real scenario but it can be argued that the cost is not prohibitive since it occurs ahead of indexing and doesn't impact the user.

Linear Convex Combination Settings

In *Linear Convex Combination* we combine the values of quality of ranking and compression as in equation 3.3. Many values of weights are used to find out the ideal weight that gives us the best comparative values. In our work the values of the weight "*w*" were ascertained manually starting from selecting the most prospective random values like 90%, 80% and so on. Further, finer values of "*w*" were used to test the behavior of the test results around those values of "*w*", which generated good results. This experiment resulted in more than one good results which were further analyzed statistically for their significance and thereafter were presented in the Results section 4.3

Table 4.1 presents the genetic programming configurations used for our experiments using *Linear Convex Combination*. In this scenario we use 1000 individuals for every generation of GP, where the stopping criterion for the GP is 40 generations. In our experiments we process the validation queries with the top 10 individuals selected from the training phase, and select the best one from them.

Weighted Geometric Mean Settings

The table 4.1 presents the genetic programming configurations used for our experiments using *Weighted Geometric Mean*. We have used similar configurations but used used a

Table 4.1: Genetic Programming configurations for *Linear Convex Combination* and *Weighted Geometric Mean*

Parameter	Value
Number of Generations	40
Population size	1000
Tree depth	17
Tournament Size	6
Crossover Rate	0.85
Mutation Rate	0.05
Reproduction Rate	0.10

different fitness function for Weighted Geometric Mean as shown in equation 3.4. We have tested with multiple values of weights assigned to "*a*" and "*b*" to finally arrive at some of the best values which gives statistically significant results. The selection of the values of "*a*" and "*b*" were done manually, similar to the strategy, how we did for *Linear Convex Combination*. The parameters and methodology for the GP process remains the same as shown in the table 4.1 and the Linear Convex Combination method.

Strength Pareto Evolutionary Algorithm Settings

In the implementation of SPEA2 [46] the selection of individuals are not directly done based on AVG_σ selection method. In this pareto based method, MOL implements the SPEA2 [46] algorithm, where the non-dominated individuals are selected from each generation to create the population for the next generation. In the training phase the best individuals are selected and then those are sorted in a top down approach with the best individuals at the top. Then, those individuals are again used to process the validation queries. These algorithms generate a SPEA fitness score for every individual. We now use the AVG_σ selection method to find out the best individual which has generated the best SPEA fitness score.

Table 4.2: GP settings for Elitism with 10% Population

Parameter	Value
Number of Generations	40
Population size	1000
Tree depth	17
Tournament Size	2
Crossover Rate	0.85
Reproduction Rate	0.10
Mutation Rate	0.05

Table 4.3: GP settings for Elitism with 20% Population

Parameter	Value
Number of Generations	40
Population size	1000
Tree depth	17
Tournament Size	2
Crossover Rate	0.75
Reproduction Rate	0.20
Mutation Rate	0.05

Table 4.2 and 4.3 presents the two different GP configurations which were for SPEA2 [46]. We keep using similar parameters like 1000 individuals, stopping criteria of 40 generations and the tree depth, but we change the parameter for the tournament size and those for the genetic operators. SPEA2 [46] implements a binary tournament to generate the population while using crossover and mutation. In the third phase of the breeding operator SPEA2 [46] proposes to select the best individuals, to promote to the next generation, thereby establishing *elitism*.

We tested SPEA2 [46] implementation of MOL with two varieties of *elitism*, where in table 4.2 the top 10% of the population is promoted to the next generation without any change. In another approach the top 20% of the population is promoted to the next generation without any change as shown in table 4.3.

4.2.2 Pruning

In our experiments, we have implemented a basic pruning strategy. We prune all those entries which do not contribute towards the ranking of the document. We pruned all UTIs with zero("0") values. The zero valued UTI values would nevertheless contribute to the ranking as they would not contribute to the computation of frequency of the features used.

This reduces the number of integers stored and facilitates in compression as well. Thus, it is expected that the results obtained during query processing would not be affected by the absence of those entries in the index.

Also, while we are compressing the UTI values with the Elias-γ [9] codes, where on one hand we do not have any codes for "0" in Elias-γ [9] encoding, on the other hand we remove this limitation by pruning or removing the zero valued entries from the UTI, and thereby from the inverted index. This strategy in fact reduces a lot of space and hence improves compression by many folds.

We would like to mention that we performed our experiments also without pruning to find the exact impact of pruning in our implementation. We can expect that the results of compression with pruning would be better than those without pruning, but it would be interesting to study the impact of pruning on compression whatsoever.

4.3 Results

In this section we present the results obtained by the numerous experiments performed by us to verify our implementation. For the sake of completeness we would like to present here, the comparison of the results obtained by LePrEF [3] with other state-of-the-art learn to rank methods. Figure 4.1 shows the values of quality(NDCG [17]) and MAP [1] for LePrEF and other state-of-art methods as proposed in the article [3].

We can clearly notice that here the NDCG result obtained by LePrEF [3] is very similar to that of the other state-of-art methods. LePrEF [3] obtained NDCG and MAP of 0.495 and 0.459, which is slightly below GP (with learning during query processing) with mean

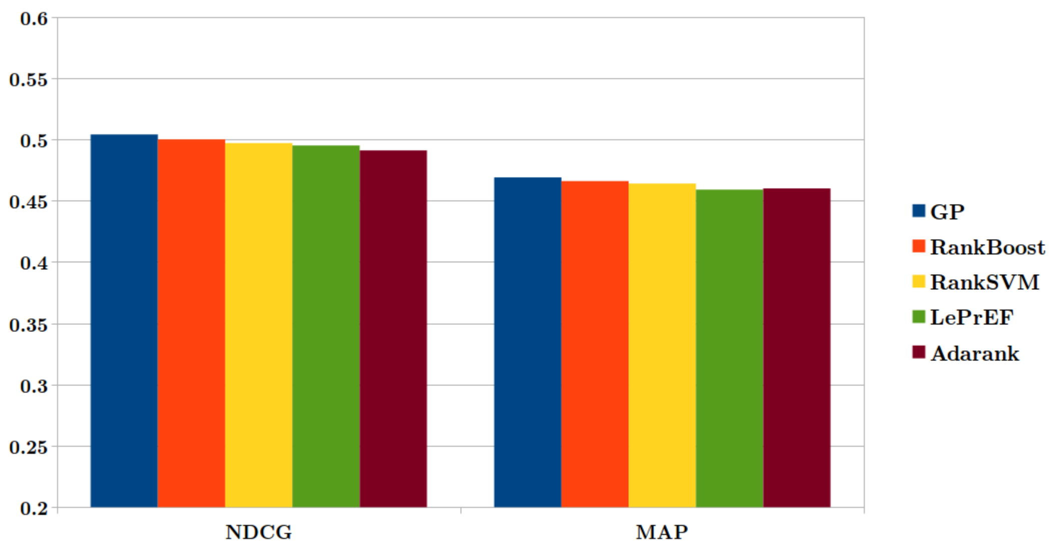

Figure 4.1: Mean NDCG and MAP results obtained by LePrEF(with integer UTI) and other state-of-the-art methods.

NDCG $0,504$ and MAP $0,469$, RankBoost [14] with mean NDCG 0.500 and MAP 0.466, and RankSVM [18] with mean NDCG 0.497 and MAP 0.464. On the other hand LePrEF [3] performed better than ADARANK [41] with mean NDCG 0.491 and MAP 0.455. Thus, we can conclude with certainty that the results of LePrEF [3] obtained quality at par with the other state-of-art learn to rank methods.

The state-of-the-art methods mentioned here implement learning during the query processing where they have access to all relevance evidence. Although the process generates search results of high quality, the computational costs are still high. LePrEF [3] here implements a pre-computational step of implementing learning during indexing. Only some and not all relevance evidences are available during indexing and hence LePrEF generates UTI indexes based on lesser information than its counterparts and obtains NDCG [17] results at par with state-of-art methods.

Before comparing the results obtained by our MOL variations, we need to identify exactly what are the best results obtainable in MOL using Linear Convex Combination (MOL-Linear), Weighted Geometric Mean (MOL-Geo) and SPEA2 (MOL-SPEA2) combinations of quality and compression rate.

4.3.1 MOL with Linear Convex Combination

In order to verify what is the best mixture parameter "*w*" for training MOL with linear convex combination in MQ2007, we experimented with a number of different values, and present the mean NDCG, MAP and compression rates for the test sets of its 5 folds in Figure 4.2.

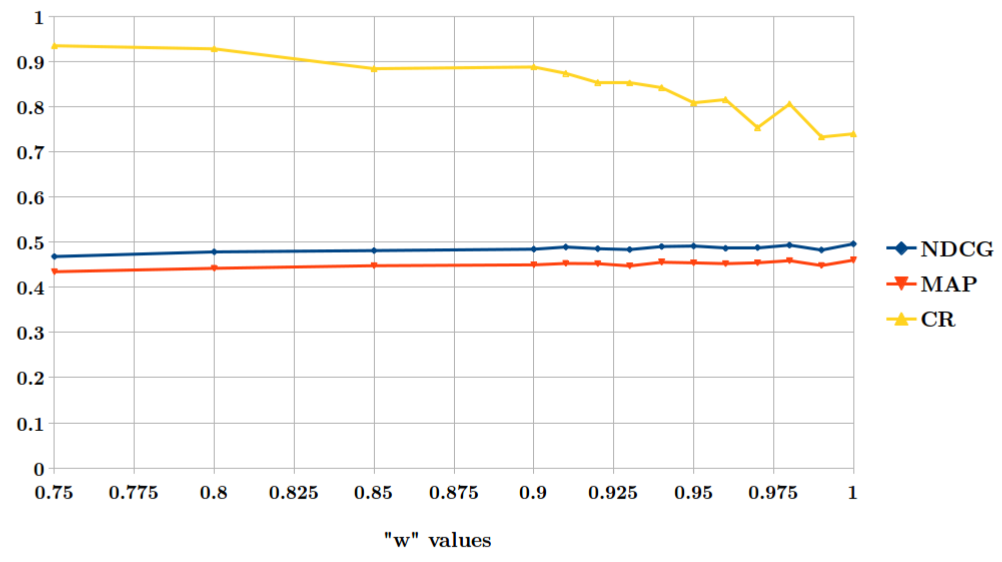

Figure 4.2: NDCG, MAP and CR(compression rate) levels obtained by MOL using Linear Convex Combination for different "w" values. The "w" values correspond to the weights assigned to quality(NDCG) and compression(CR).

We can see here that, there is indeed a trade-off between quality(NDCG [17]) and compression as the weight assigned to quality increases. As we have already mentioned that our principal target is to achieve better compression without sacrificing quality, we would study those methods which achieve NDCG values close to LePrEF [3] and then study their compression values. The best quality values were obtained by using 0.94, 0.95 and 0.98 as values for "*w*". For 0.94, MOL achieved a mean NDCG of 0.49, MAP of 0.455 and about 84% compression in the test set, which is 5.07 bits/entry; for 0.95% MOL achieved a mean NDCG of 0.491, MAP of 0.454 and about 81% compression in the test set, which is 6.14 bits/entry and finally for 0.98, MOL achieved a mean NDCG of 0.493, MAP of 0.459 and about 81% compression in the test set, that is 6.22 bits/entry. We implemented the LePrEF [3](results are shown as MOL with linear convex combination where "*w*" = 1, which behaves exactly as the original LePrEF) by incorporating the compression calculations in it, and found that the quality values 0.495 were slightly better, with MAP of 0.459, but with a compression of around 74%, which is 8.33 bits/entry. We can infer from the compression values of the best results for MOL that we are able to achieve about 39% more compression while using MOL with Linear Convex Combination, without deviating very much from the quality baselines.

We also performed experiments with values of "*w*" below 0.75, that is 75% weight to quality, however we decided not to present them, thereby focusing on the results that we found to be more interesting for the reader. Results with "*w*" below 0.75 kept a gradual fall in quality while the compression kept showing improvements which does not help our cause as the quality values at this level were too low.

4.3.2 MOL with Weighted Geometric Mean

As with the linear combination, while using the weighted geometric mean, we also have to verify which parameters might lead to the best compression/quality tradeoff. In order to verify at which parameters MOL with weighted geometric mean (MOL-Geo) leads to the best quality results while maintaining a good compression level, we fixed the parameter "*b*", which pertains to the compression rate, as 1 and tested for the different values of "*a*", and the results are presented in Figure 4.3.

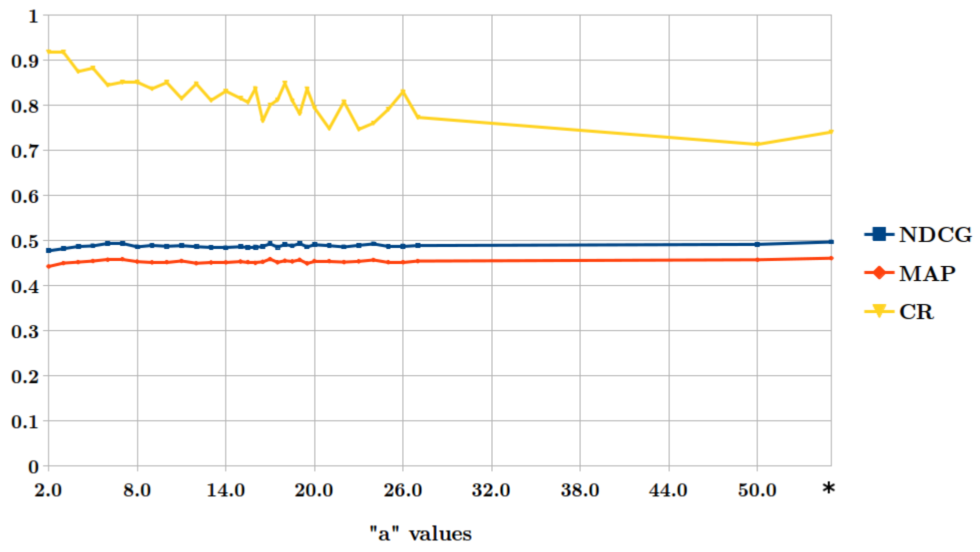

Figure 4.3: NDCG, MAP and compression rate levels obtained by MOL using Weighted Geometric Mean for different "*a*" values, with "*b*" = 1. "*a*" is the weight assigned to quality (NDCG). The * value is the result obtained for "*a*" = 1 and "*b*" = 0 (i.e. all the weight given to quality, which is in fact, the value for original LePrEF)

As it may be seen, in MOL-Geo, there is a larger instability in quality and compression levels as the parameter "*a*" increases, where small changes in values of "*a*" can have positive or negative impact on both compression and quality. The best results with MOL were obtained with "*a*" values of 7, 17 and 19 among others, which resulted in NDCG values in the range of 0.492 – 0.493, MAP values around 0.458, and compression of around 85%, or 4.8 bits/entry. If we study the pattern of the results achieved by MOL-Geo as shown in Figure 4.3, we can easily conclude that the results beyond "*a*" values 50 might not have yielded compression more than our baseline LePrEF [3]. For "*a*" value of 50 we have the mean NDCG of 0.491 whereas, the 71% compression drops below the 74% of LePrEF [3].

Owning to inherent randomness of GP we can always say that there might be one or more solutions which might prove better than the ones presented here. The experiments can always be repeated with multiple different “*a*” values in light of this.

4.3.3 MOL with SPEA2

The results Multi-Objective LePrEF obtained while using SPEA2 [46] to find a pareto front of individuals were not as good as the ones obtained by linear convex combination and weighted geometric mean. MOL-SPEA2 obtained very strong compression rates, but at the cost of smaller quality values. We have used two values for the genetic operator *reproduction*: 0.10 and 0.20 which stands for top 10% and 20% respectively, of the population going to the next generation without any modification, which is popularly known as *Elitism*, thereby making the next generation more viable for better solutions. Further higher values were not preferred as those might have an over-fitting solutions. A graphical comparison of the results of MOL with SPEA2 [46] to that of LePrEF [3] is shown in Figure 4.4.

The best compression rate achieved by MOL-SPEA2 was about 86% (4.5 bits per entry), with mean NDCG of 0.475 and MAP of 0.442, which were way below the values of quality and precision obtained by the original LePrEF as well as MOL with Linear Convex Combination and MOL with Weighted Geometric Mean.

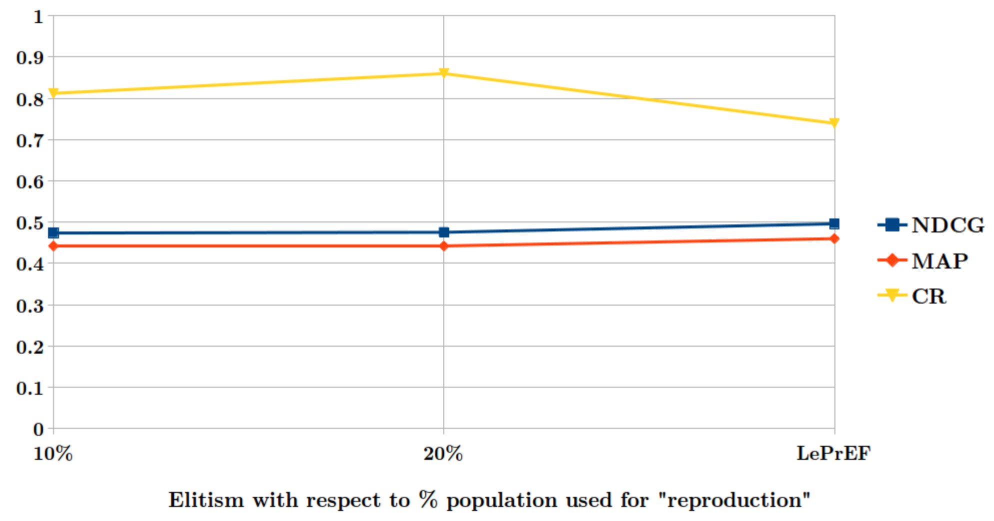

Figure 4.4: NDCG, MAP and compression rate levels obtained by MOL using SPEA2 for two different values, which control *Elitism*

4.3.4 Overall Results

We have performed multiple experiments for MOL using three different approaches: linear convex combination, weighted geometric mean and strength pareto evolutionary algorithm – 2. We performed experiments with various data which were first manually selected at random and then, selected the next parameters based on the results of the past experiment. The results for all these experiments came out with different values for quality and compression, some of which are very close of original LePrEF [3] and some are considerably very less, and can be rejected readily. As we have already discussed earlier that we are using the mean NDCG [17] values of the collection of queries for each method over 10 runs, and not the NDCG values for a single run, we need to be statistically sure before commenting further on any of the final results.

To find out if the differences of the results of MOL were statistically significant than LePrEF [3] we can use T-Test [38], Wilcoxon Test [40] among others. The results of experiments of MOL does not follow normal distribution. Thus, we cannot use T-Test in

Table 4.4: Percent weights assigned to quality(NDCG) of each selected method for comparison

Methods	% Weight
MOL-Linear - 0.95	95.00%
MOL-Linear - 0.98	98.00%
MOL-Geo - 7.0	87.50%
MOL-Geo - 17.0	94.44%
MOL-Geo - 19.0	95.00%
MOL-Geo - 50.0	98.04%

this scenario. We adopted Wilcoxon Test [40] with 95% significance (α=0.05) to test the significance of the differences between the results of MOL from LePrEF [3].

The Wilcoxon Test [40] proves that in both the approaches of MOL-SPEA2, NDCG values suffer from statistically significant loss of quality with respect to LePrEF and thus, MOL-SPEA2 approaches are ruled out from being solutiosn for MOL, whereas the test shortlisted four MOL-Geo methods with "*a*" values of 7, 17, 19 and 50, and two MOL-Linear methods with "*w*" values of 0.95 and 0.98 as the ones which do not show statistically significant loss in quality when compared with the NDCG values of LePrEF [3]. This means, the quality results of the methods, MOL-Geo - 7, 17, 19, 50, MOL-Linear - 0.95, 0.98 and the baseline LePrEF [3] can be considered same. Henceforth we would present our analysis based on the only those results which do not have statistically significant loss in quality when compared to LePrEF [3], thus we remove the results of MOL-SPEA2 from our consideration. Table 4.4 shows the percentage of weight being assigned to quality for each of the compared methods mentioned here.

Figure 4.5 shows the NDCG@N results obtained for N= 5 and N= 10, for all of the compared methods. LePrEF [3] yielded the best result for NDCG@5(0.415) followed closely by MOL-Geo - 7.0(0.411), MOL-Geo - 19.0(0.411) and MOL-Linear - 0.98(0.410). The values NDCG@5 for the other compared methods were not really very

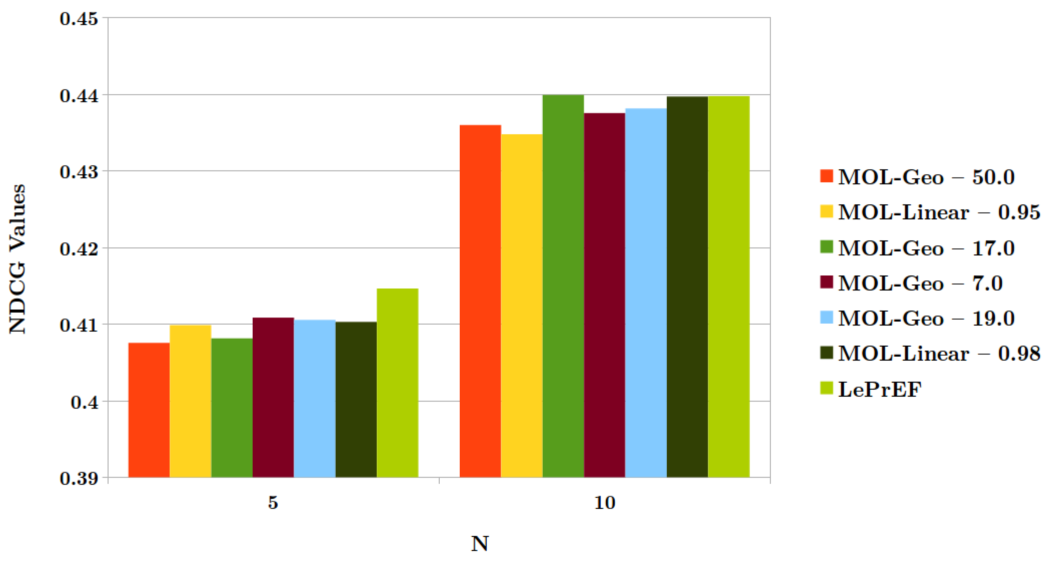

Figure 4.5: NDCG@N results obtained by MOL methods which do not show statistically significant quality loss, and LePrEF

different, MOL-Geo - 50.0 being 0.407, MOL-Geo - 17.0 being 0.408 and MOL-Linear - 0.98 showing NDCG@5 of 0.410. When we look at the values of NDCG@10 the best best values were obtained by MOL-Geo - 17.0 and MOL-Linear - 0.98 which were same as that of LePrEF [3](0.439). The other compared methods, MOL-Linear - 0.95 (0.434), MOL-Geo - 7.0(0.437), MOL-Geo - 19.0(0.438) and MOL-Geo - 50.0(0.435) follow these results closely. We can conclude that in our experiments with MOL we achieve similar levels of quality with added features of compression.

Figure 4.6 shows the P@5 and P@10 values obtained by all the compared methods. LePrEF [3](0.414) showed the best P@5 value whereas MOL-Geo - 17.0(0.382) yielded the best P@10 value. We can see that the other alternatives also obtained very competitive values, for instance MOL-Linear - 0.98 achieved P@5 of 0.412, MOL-Geo - 7.0 and MOL-Geo - 19.0 obtained P@5 values of 0.411 and 0.41 respectively. In case of P@10 also while the baseline LePrEF yielded 0.377, all other compared methods achieved P@10 values greater than the same except for MOL-Linear - 0.95(0.374). Thus,

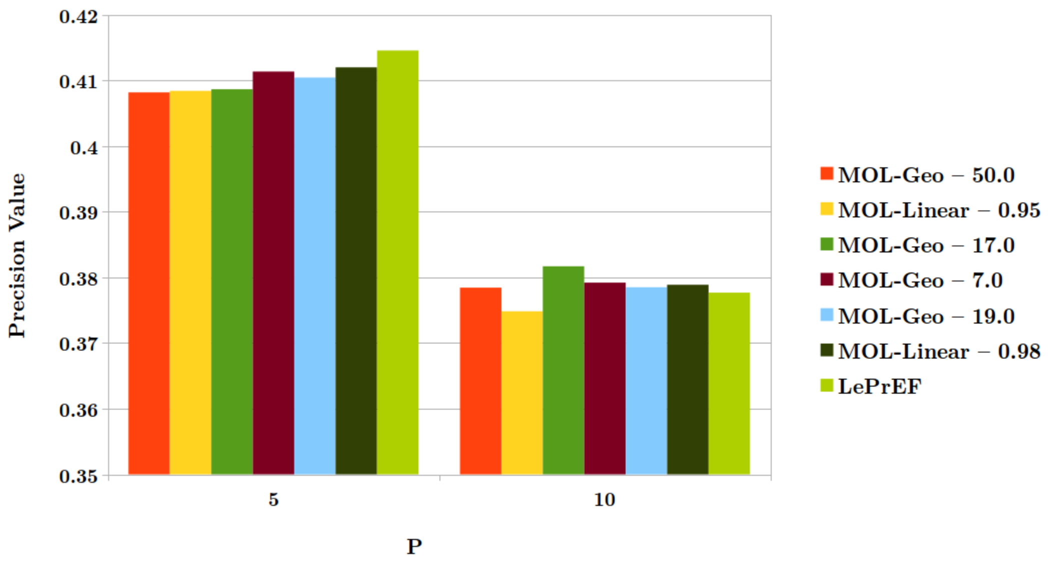

Figure 4.6: MAP@N results obtained by MOL methods which do not show statistically significant quality loss, and LePrEF

it can be seen that the MOL approach has exceeded precision values than the baseline.

While studying *MOL-Linear* and *MOL-Geo* we found that the best approaches of MOL were numerically little lower than the baseline (LePrEF [3]) but were still competitive. Figure 4.7 presents the comparison of quality values of MOL(best approaches) against LePrEF [3]. We can readily infer from the graph that the MOL quality values are very close to LePrEF. LePrEF [3] has NDCG 0.496 and MAP 0.456 whereas, MOL-Linear with 98% weight on quality (MOL-Linear - 0.98) performs exceptionally well with NDCG of 0.493 and MAP of 0.458, followed by MOL-Geo - 19.0 with mean NDCG 0.493 and MAP 0.456, MOL-Geo - 7.0 with mean NDCG 0.492 and MAP 0.458, MOL-Geo - 17.0 with mean NDCG 0.492 and MAP 0.458, MOL-Linear - 0.95 with mean NDCG 0.491 and MAP 0.454 and MOL-Geo - 50.0 with mean NDCG 0.491 and MAP 0.456.

Finally, the most important outcome from our implementation is the expressive gains

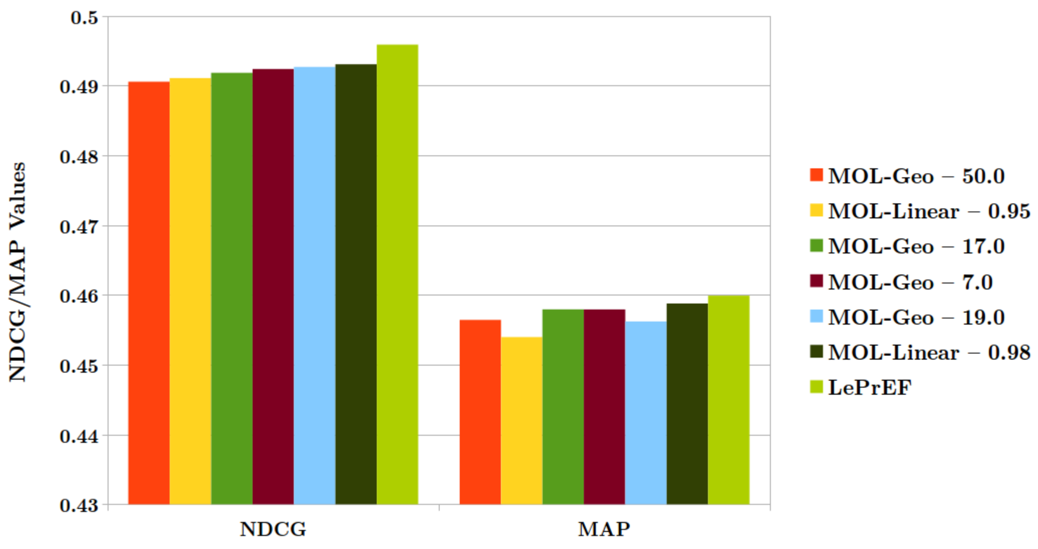

Figure 4.7: Comparison of the results obtained by MOL methods which do not show statistically significant quality loss, and the original LePrEF, measured in terms of NDCG and MAP

in the compression rates achieved in MOL, that comprehensively states that MOL can successfully generate compression friendly indexes/UTIs, which, at the same time, can also generate very high quality results. Figure 4.8 shows the compression achieved in all the compared methods expressed in bits/entry.

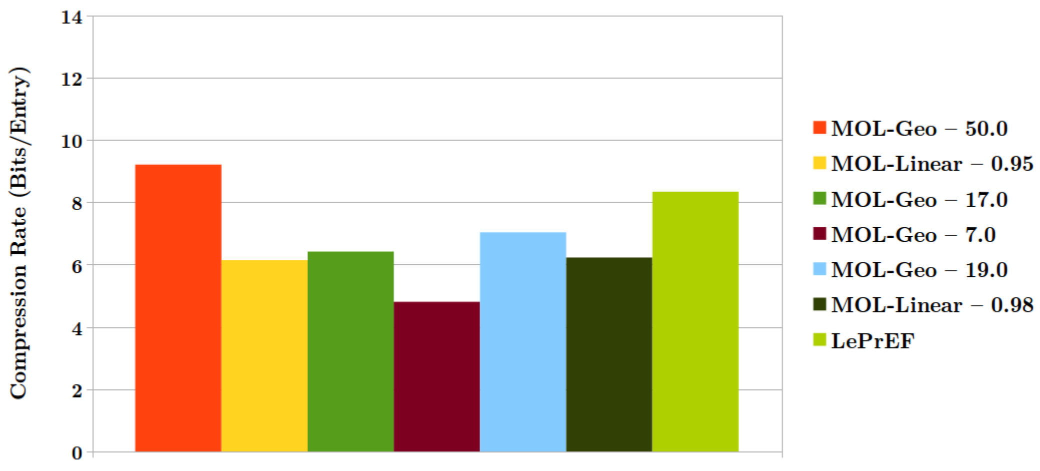

Figure 4.8: Compression Rates obtained by MOL methods which do not show statistically significant quality loss, and the original LePrEF, while using Pruning

In our experiments the best compression is achieved by MOL-Geo - 7.0 (4.8 bits/entry) whereas the worst is obtained by MOL-Geo - 50.0 (9.2 bits/entry). Where the baseline LePrEF [3] is at 8.34 bits/entry after the implementation of similar compression and pruning strategies, the other compared methods, MOL-Geo - 17.0 obtained a compression of 6.41 bits/entry, MOL-Geo - 19.0 yielded 7.03 bits/entry and MOL-Linear - 0.95 yielded 6.14 bits/entry, each one of these methods thereby providing better compression while maintaining nearly similar levels of quality. The best method MOL-Geo - 7.0 achieved about 42% more compression than the baseline. The other methods also obtained good compression levels in comparison to LePrEF [3]: MOL-Linear - 0.95 and 0.98 both yielded about 25% better compression whereas MOL-Geo - 19.0 and MOL-Geo - 17.0 showed about 16% and 23% gains in compression over the baseline. From the presented data we can come to the conclusion that, since the Wilcoxon Test [40] shows that the loss of quality for MOL methods are not statistically significant when compared to LePrEF [3], the best method is MOL-Geo - 7.0, which yields more than 42% compression than LePrEF. The next best compression was shown by MOL-Linear - 0.98, MOL-Linear - 0.95 and MOL-Geo - 17.0.

It is important to mention here that the compression achieved here is a combination of both pruning and compression. LePrEF [3] was not designed with a pruning strategy and hence we implemented pruning in LePrEF [3], and thereafter computed all metrics necessary for the comparison of LePrEF [3] with all of the shortlisted methods. To compare the results of compression obtained in MOL with LePrEF [3] directly, we show the compression obtained by all the methods without considering pruning. Figure 4.9 shows the compression rates achieved by the compared methods using compression only, without pruning.

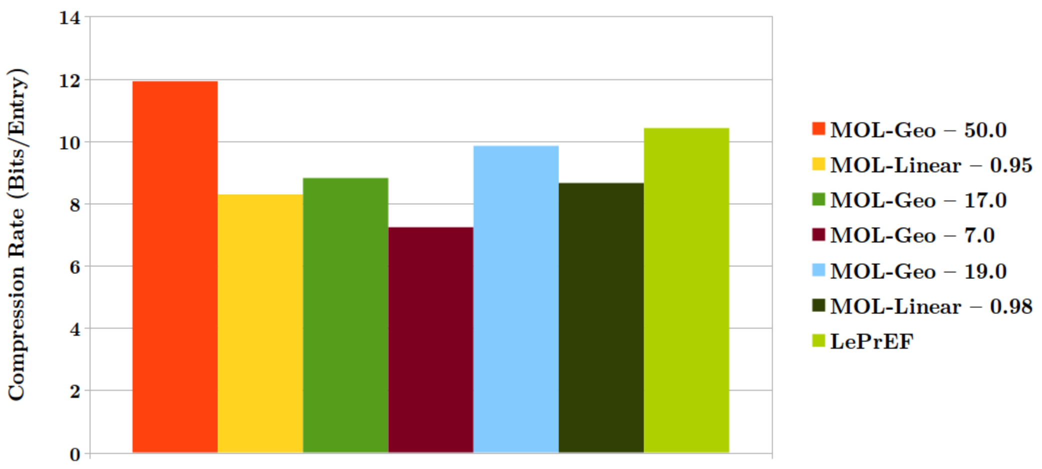

Figure 4.9: Compression Rates obtained by MOL methods which do not show statistically significant quality loss, and the original LePrEF, without Pruning

In the experiments performed by us the best compression is obtained by MOL-Geo - 7.0 with 7.23 bits/entry and the worst is obtained by MOL-Geo - 50.0 with 11.92 bit-s/entry. The baseline LePrEF [3] yielded a compression of 10.41 bits/entry. With the exception of MOL-Geo - 50.0, all compared methods which do not have a statistically significant loss in quality achieved better compression than the baseline. We can clearly see that the best compression rate achieved in MOL-Geo - 7.0 is more than 30% better than the baseline if we do not consider pruning. MOL-Geo - 50.0 obtained about 14% lesser compression, whereas MOL-Linear - 0.98 and MOL-Linear - 0.95 showed better compression by about 17% and 20% respectively. The MOL-Geo - 17.0 and The MOL-Geo - 19.0 also showed compression gains of about 15.5% and 5.5% respectively over the baseline. Thus, MOL methods, as a whole, achieved better compression over LePrEF without the use of pruning also, with the best method still being MOL-Geo - 7.0. Here also the next best methods were the same as of that in the case of overall compression with pruning, MOL-Linear - 0.98, MOL-Linear - 0.95 and MOL-Geo - 17.0.

We performed further analysis of the results of MOL and LePrEF [3] by studying the data distribution of UTI values generated by the aforementioned methods. Figures 4.10 and 4.11 presents the UTI data distribution for both.

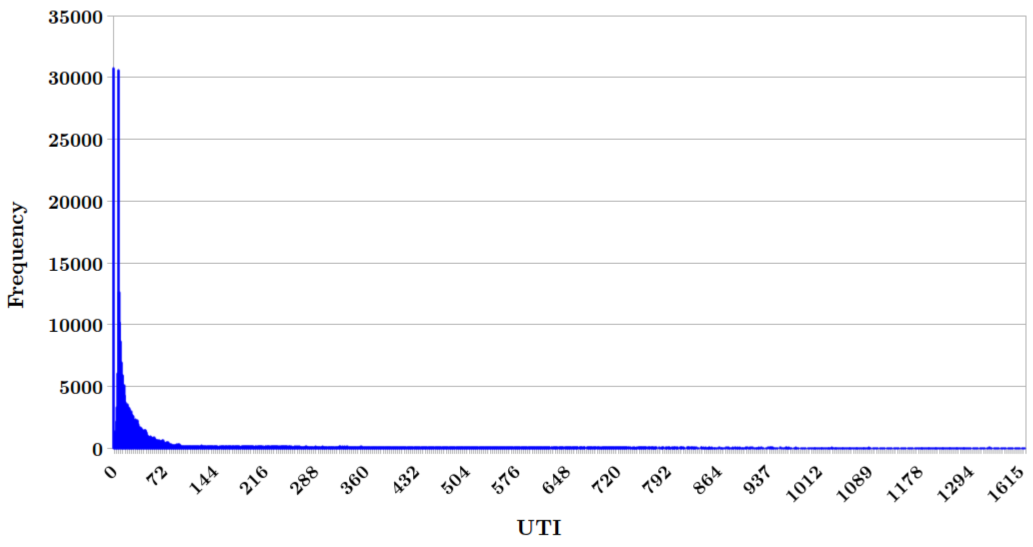

Figure 4.10: UTI data distribution for LePrEF

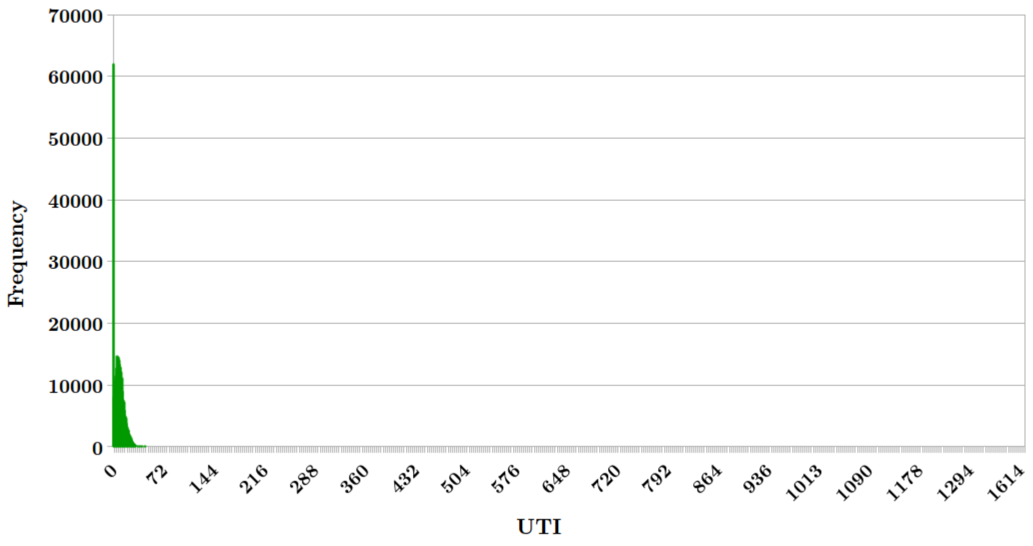

Figure 4.11: UTI data distribution for MOL(MOL-Geo - 7)

For better understanding we are using the UTI values of the best MOL method, MOL-Geo - 7. We can see that the UTI values in LePrEF [3] have a wider distribution of values than that of MOL. MOL data distribution has UTI integer values of only upto

49 whereas LePrEF generates UTI integer values of upto 3461. We also notice that the number of 0 UTI values in MOL is way more than the number in LePrEF, which in turn helps in concise index and higher compression. In MOL, genetic programming learns to generate only those UTI values, which leads to good quality search results as well as good compression, and hence does not generate very high integer values, which are not very compression friendly. On the contrary, in LePrEF, where only quality is used as an objective function and not compression, it generates UTI values in order to create a good quality search results only and does not care even if those values are high and are not compression friendly.

Impacts of Pruning on Compression

As seen in previous subsections, the implementation of the MOL variations can lead to significant increase in compression rate. We have mentioned the scenario where MOL uses compression and no pruning but still leading to better compression rates than LePrEF [3]. In another scenario where we use both pruning and compression together to compress the inverted indexes we also achieved expressive gains in compression over the baseline. However, it is not clear whether this behavior is due to the fact that the impact values generated by MOL are smaller (and, thus, compressing better), or if it is due to the pruning of larger number of entries. In order to better understand the role of pruning in this compression gain, in Figure 4.12 we present the overall compression rate achieved by the methods, as well as how much of this compression was achieved by pruning entries and by compression the data after pruning. We should note here that the compression expressed here is in percentage of space saving yielded during our experiments.

As can be seen, indeed the MOL variations lead to an increase in entry pruning in

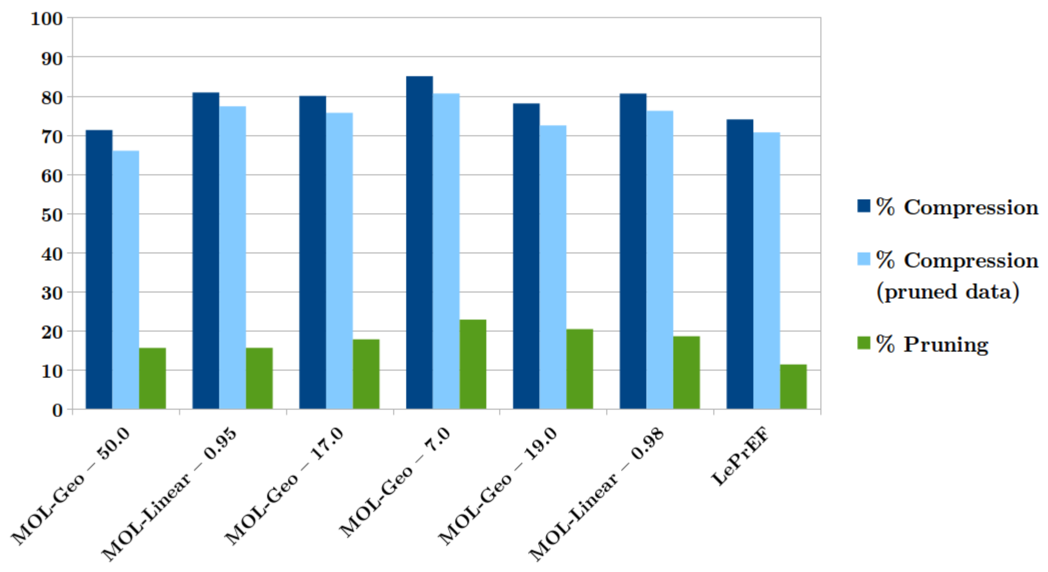

Figure 4.12: Comparison of overall compression with respect to compression and pruning separately

comparison to the original LePrEF [3] in the experiments performed by us. Introduction of pruning increased the overall compression in all compared methods which do not have statistically significant loss in quality against LePrEF [3] with the only exception being MOL-Geo - 50.0 where the overall compression fell below the original compression of LePrEF. MOL-Geo - 7.0 achieved a greater compression by aggressively pruning a lot of entries. Further careful study reveals that if we consider the compression of the residual data after pruning, the trend of compression rates remains almost the same for all compared methods, except an expected little dip in the compression rates. This drop in the compression rates can be credited to the fact that pruning removes the unyielding entries from the index, thereby reducing the size and the compression rate.

Chapter 5

Conclusions

In this thesis we implement an idea of introducing multi-objective optimization techniques in LePrEF to propose Multi-Objective LePrEF (MOL) which inherits all advantages of LePrEF and generates high quality compression friendly indexes. The principal advantage of this method is that with MOL we achieve high quality search results similar to those of LePrEF, and also achieve an expressive gain in index compression as an alternative objective. LePrEF only considers a single objective, quality of search results (NDCG), whereas MOL works with two objectives (quality and compression) in parallel. A compressed index is always very important to search systems, because firstly, they take less space and thus, reduce storage costs, secondly, they make data transfer faster with lesser disc reads and writes and thirdly, helps in increased and efficient caching during query processing.

MOL implemented three different techniques to study the effects of multi-objective optimization on LePrEF. We used linear convex combination(Linear), weighted geometric mean(Geo) and strength pareto evolutionary algorithm (2^{nd} version)(SPEA2) as our means to carry our experiments to find out the best possible method. Our studies have used LETOR dataset and shown that though MOL-SPEA2 attains a great compression, the NDCG values were far below the baseline and hence was not acceptable. On the contrary the best approaches of MOL-Linear and MOL-Geo showed great NDCG values, the best one being MOL-Geo - 7.0 with mean NDCG of 0.493 and MAP 0.458 alongside LePrEF's NDCG of 0.496 and MAP 0.459. As it was also established by the Wilcoxon Tests that MOL-Geo - 7.0 results do not have any statistically significant loss in quality and that showed expressive gains in the compression rates, 4.8 bits/entry to 8.33 bits/entry shown by LePrEF, which is about 42% more compression, we can conclude that our proposal of MOL is able to generate a model which successfully implements multiple objectives at the same time and generates indexes which yield high quality search results and are also very much compression friendly.

Our work on one hand answers several questions regarding multi-objective optimization in LePrEF, and on the other opens up various new research directions. We have used basic pruning techniques to remove entries from UTI which do not contribute to ranking. The use of pruning has helped generating a better compressed index without any visible loss in the quality of the search results. This motivates us to look further in this direction with the idea of using more sophisticated pruning techniques to study how the quality of results behave.

We have used *Elias*-γ coding to compress the indexes. In future further research may be put into the same direction by using more sophisticated compression schemes which can compress the indexes more, thereby reducing the storage and other advantages of compressed indexes. Also, it would be very interesting to see how does the highly compressed indexes perform during query processing. Thus, usage of compression schemes which can decompress very fast would be the next alternative to use in MOL.

In the future MOL can be also used with faster query processing techniques to find out the efficiency performance of MOL with respect to the baselines. It would be interesting to find out the details of such experiments and study more about these.

This thesis provided us a new look at the existing LePrEF to use multi-objective optimization techniques to generate indexes that compress better. We see that the modern search systems obviously can be improved a lot and our new research direction can be further exploited in this direction to achieve more.

Bibliography

[1] Baeza-Yates, R., and Ribeiro-Neto, B. *Modern Information Retrieval*, 2 ed. Addison-Wesley Professional, Boston, MA, USA, 2011.

[2] Cambazoglu, B. B., Zaragoza, H., Chapelle, O., Chen, J., Liao, C., Zheng, Z., and Degenhardt, J. Early exit optimizations for additive machine learned ranking systems. In *Proceedings of the third ACM international conference on Web search and data mining* (2010), ACM, pp. 411–420.

[3] Costa Carvalho, A. L., Rossi, C., Moura, E. S., Silva, A. S., and Fernandes, D. Lepref: Learn to precompute evidence fusion for efficient query evaluation. *Journal of the American Society for Information Science and Technology 63*, 7 (2012), 1383–1397.

[4] Dang, V., Bendersky, M., and Croft, W. B. Two-stage learning to rank for information retrieval. In *Advances in Information Retrieval*. Springer, 2013, pp. 423–434.

[5] de Almeida, H. M., Gonçalves, M. A., Cristo, M., and Calado, P. A combined component approach for finding collection-adapted ranking functions based on genetic programming. In *Proceedings of the 30th annual international ACM SIGIR* (2007), ACM, pp. 399–406.

[6] De Moura, E. S., dos Santos, C. F., Fernandes, D. R., Silva, A. S., Calado, P., and Nascimento, M. A. Improving web search efficiency via a locality based static pruning method. In *Proceedings of the 14th international conference on World Wide Web* (2005), ACM, pp. 235–244.

[7] De Moura, E. S., Santos, C. F. d., Silva, A. S. d., Calado, P., Nascimento, M. A., et al. Locality-based pruning methods for web search. *ACM Transactions on Information Systems (TOIS) 26*, 2 (2008), 9.

[8] Deb, K., Pratap, A., Agarwal, S., and Meyarivan, T. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation 6*, 2 (2002), 182–197.

[9] Elias, P. Universal codeword sets and representations of the integers. *IEEE Transactions on Information Theory 21*, 2 (1975), 194–203.

[10] Fan, W., Fox, E. A., Pathak, P., and Wu, H. The effects of fitness functions on genetic programming-based ranking discovery for web search. *Journal of the American Society for Information Science and Technology 55*, 7 (2004), 628–636.

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

[12] Fan, W., Gordon, M. D., Pathak, P., Xi, W., Fox, E., et al. Ranking function optimization for effective web search by genetic programming: An empirical study. In *System Sciences, 2004. Proceedings of the 37th Annual Hawaii International Conference on* (2004), IEEE, pp. 8–pp.

[13] Fonseca, C. M., and Fleming, P. J. An overview of evolutionary algorithms in multiobjective optimization. *Evolutionary computation 3*, 1 (1995), 1–16.

[14] Freund, Y., Iyer, R., Schapire, R. E., and Singer, Y. An efficient boosting algorithm for combining preferences. *The Journal of machine learning research 4* (2003), 933–969.

[15] Golomb, S. Run-length encodings. *IEEE Trans. Inf. Theor. 12*, 3 (July 1966), 399–401.

[16] Iba, H., Hasegawa, Y., and Paul, T. K. *Applied genetic programming and machine learning*. cRc Press, 2009.

[17] JÄRVELIN, K., AND KEKÄLÄINEN, J. Cumulated gain-based evaluation of ir techniques. *ACM Trans. Inf. Syst. 20*, 4 (Oct. 2002), 422–446.

[18] JOACHIMS, T. Optimizing search engines using clickthrough data. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining* (2002), ACM, pp. 133–142.

[19] KOZA, J. R. *Genetic programming: on the programming of computers by means of natural selection*, vol. 1. MIT press, 1992.

[20] KOZA, J. R., JONES, L. W., KEANE, M. A., STREETER, M. J., AND AL-SAKRAN, S. Towards industrial strength automated design of analog electrical circuits by means of genetic programming. *Genetic Programming Theory and Practice II* (2004), 121–138.

[21] LANGDON, W., BARRETT, S., AND BUXTON, B. Predicting biochemical interactions-human p450 2d6 enzyme inhibition. In *The Congress on Evolutionary Computation* (2003), vol. 2, IEEE, pp. 807–814.

[22] LANGDON, W., AND POLI, R. Better trained ants for genetic programming. Tech. rep., The University of Birmingham, UK, 1998.

[23] LEMIRE, D., AND BOYTSOV, L. Decoding billions of integers per second through vectorization. *Software: Practice and Experience 45*, 1 (2015), 1–29.

[24] LIU, T.-Y., XU, J., QIN, T., XIONG, W., AND LI, H. Letor: Benchmark dataset for research on learning to rank for information retrieval. In *Proceedings of SIGIR 2007 workshop on learning to rank for information retrieval* (2007), pp. 3–10.

[25] MANNING, C. D., RAGHAVAN, P., SCHÜTZE, H., ET AL. *Introduction to information retrieval*, vol. 1. Cambridge University Press, Cambridge, UK, 2008.

[26] MISHRA, K., AND HARIT, S. A fast algorithm for finding the non dominated set in multiobjective optimization. *International Journal of Computer Applications 1*, 25 (2010), 46–54.

[27] OREILLY, U. M., AND HEMBERG, M. Integrating generative growth and evolutionary computation for form exploration. *Genetic Programming and Evolvable Machines 8*, 2 (2007), 163–186.

[28] OSYCZKA, A. Multicriteria optimization for engineering design. *Design optimization 1* (1985), 193–227.

[29] OTTAVIANO, G., AND VENTURINI, R. Partitioned elias-fano indexes. In *Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval* (2014), ACM, pp. 273–282.

[30] PUNCH, B., AND ZONGKER, D. Lilgp 1.01 user's manual. Tech. rep., Technical report, Michigan State University, Michigan, USA, 1996.

[31] RICE, R. Some practical universal noiseless coding techniques. *Tech. Rep. JPL-79-22* (1979).

[32] ROBERTSON, S. E., AND WALKER, S. Some simple effective approximations to the 2-poisson model for probabilistic weighted retrieval. In *Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval* (1994), Springer-Verlag New York, Inc., pp. 232–241.

[33] SARAIVA, P. C., SILVA DE MOURA, E., ZIVIANI, N., MEIRA, W., FONSECA, R., AND RIBERIO-NETO, B. Rank-preserving two-level caching for scalable search engines. In *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval* (2001), ACM, pp. 51–58.

[34] SILVA, T. P. C., DE MOURA, E. S., CAVALCANTI, J. M. B., DA SILVA, A. S., DE CARVALHO, M. G., AND GONÇALVES, M. A. An evolutionary approach for combining different sources of evidence in search engines. *Information Systems 34*, 2 (2009), 276–289.

[35] SILVERMAN, B. W. *Density estimation for statistics and data analysis*, vol. 26. CRC press, 1986.

[36] SRINIVAS, N., AND DEB, K. Muiltiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary computation 2*, 3 (1994), 221–248.

[37] STEPANOV, A. A., GANGOLLI, A. R., ROSE, D. E., ERNST, R. J., AND OBEROI, P. S. Simd-based decoding of posting lists. In *Proceedings of the 20th ACM international conference on Information and knowledge management* (2011), ACM, pp. 317–326.

[38] STUDENT. The probable error of a mean. *Biometrika* (1908), 1–25.

[39] TROTMAN, A. Learning to rank. *Information Retrieval 8*, 3 (2005), 359–381.

[40] WILCOXON, F. Individual comparisons by ranking methods. *Biometrics bulletin 1*, 6 (1945), 80–83.

[41] XU, J., AND LI, H. Adarank: a boosting algorithm for information retrieval. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval* (2007), ACM, pp. 391–398.

[42] YAN, H., DING, S., AND SUEL, T. Inverted index compression and query processing with optimized document ordering. In *Proceedings of the 18th international conference on World wide web* (2009), ACM, pp. 401–410.

[43] YEH, J.-Y., LIN, J.-Y., KE, H.-R., AND YANG, W.-P. Learning to rank for information retrieval using genetic programming. In *Proceedings of SIGIR 2007 Workshop on Learning to Rank for Information Retrieval (LR4IR 2007)* (2007).

[44] ZHANG, J., LONG, X., AND SUEL, T. Performance of compressed inverted list caching in search engines. In *Proceedings of the 17th international conference on World Wide Web* (2008), ACM, pp. 387–396.

[45] ZHANG, M., AND BHOWAN, U. Pixel statistics and program size in genetic programming for object detection. Tech. rep., Victoria University of Wellington, 2004.

[46] ZITZLER, E., LAUMANNS, M., THIELE, L., ZITZLER, E., ZITZLER, E., THIELE, L., AND THIELE, L. Spea2: Improving the strength pareto evolutionary algorithm. Tech. rep., Swiss Federal Institute of Technology, (ETH) Zurich, 2001.

[47] ZITZLER, E., AND THIELE, L. An evolutionary algorithm for multiobjective optimization: the strength pareto approach. Tech. rep., Swiss Federal Institute of Technology Zürich (ETH), 1998.

[48] ZITZLER, E., AND THIELE, L. Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach. *IEEE Transactions on Evolutionary Computation 3*, 4 (Nov 1999), 257–271.

[49] ZIVIANI, N., DE MOURA, E. S., NAVARRO, G., AND BAEZA-YATES, R. Compression: A key for next-generation text retrieval systems. *Computer 33*, 11 (2000), 37–44.

[50] ZUKOWSKI, M., HEMAN, S., NES, N., AND BONCZ, P. Super-scalar ram-cpu cache compression. In *Data Engineering, 2006. ICDE'06. Proceedings of the 22nd International Conference on* (2006), IEEE, pp. 59–59.