FACULTY OF ENGINEERING AND ARCHITECTURE

Music recommendation using genetic programming

Robbe Vanhaesebroeck
Student number: 01508456

Supervisors: Prof. dr. ir. Luc Martens, Prof. dr. ir. Toon De Pessemier
Counsellor: Stefaan Vercoutere

Master's dissertation submitted in order to obtain the academic degree of
Master of Science in Computer Science Engineering

Academic year 2019-2020

GHENT UNIVERSITY

FACULTY OF ENGINEERING AND ARCHITECTURE

Music recommendation using genetic programming

Robbe Vanhaesebroeck
Student number: 01508456

Supervisors: Prof. dr. ir. Luc Martens, Prof. dr. ir. Toon De Pessemier
Counsellor: Stefaan Vercoutere

Master's dissertation submitted in order to obtain the academic degree of
Master of Science in Computer Science Engineering

Academic year 2019-2020

GHENT UNIVERSITY

Acknowledgement

The period of my master thesis has been without a doubt the weirdest period of my life so far. Not only was this the first time that I had to undertake a project of such proportions, but at the same time, the COVID-19 virus has significantly changed daily life for everybody. Given these special circumstances, it has been an exceptionally hard year that I would not have been able to complete if not for the following people.

First, I would like to thank my supervisor Stefaan Vercoutere who has helped me throughout this year. I could always knock on his door and ask him questions on subjects I did not understand, have discussions about designs and implementations, and he took the time to read parts of my report and give feedback. I would also like to thank my promotors prof. dr. ir. Luc Martens and prof. dr. ir. Toon De Pessemier for proposing this subject, attending the intermediate presentation, and providing valuable feedback.

In my personal life too, there are a few people whom I would like to show my appreciation and without whom I would not have been able to finish this tremendous project. My girlfriend Isa Wille, who has always been there for me, who celebrated my victories with me and helped me get through the rough moments. My friend Pieter De Clercq who supported me through thick and thin, always ready to help me out and listen to me. Doing our thesis at the same time was something that gave me the strength and willpower to give it my best, and sharing jokes and memes always put a smile on my face. My other friend Pieter Dernau, whose regular video chats were a welcome distraction from the project. I would not have had any social contact if it weren't for him. Additionally, he read certain parts and corrected errors.

Finally, I would like to express my love and gratitude for my family. My brother who has brought out the best in me and always encouraged me to be as ambitious as himself. My parents, who have supported me, not only this year but my entire life. They have always let me follow my instincts and have given me the space to grow and learn the things I was passionate about. Special thanks to my mother for reading my entire work and helping with fixing errors. I hope I will make you proud.

Permission for Usage

Music Recommendation Using Genetic Programming

Robbe VANHAESEBROECK (01508456)

Supervisors: Prof. Dr. Ir. Luc Martens, Prof. Dr. Ir. Toon De Pessemier

Counsellor: Stefaan Vercoutere

Master's dissertation submitted in order to obtain the academic degree of
Master of Science in Computer Science Engineering

Academic year 2019–2020

Abstract

Recommender systems help people find their way in large volumes of information in a personalised manner. Their use has become widespread in many domains such as music, movies, e-commerce, news,... Traditional recommender systems focus on accuracy but this may be detrimental to user satisfaction. Evolutionary computing is a suitable candidate to optimise other, more user-centric metrics while not completely neglecting the accuracy requirement. This work presents an algorithm that produces a set of recommendations using a traditional recommender system. Next, the items are reranked according to a function constructed by genetic programming. This way, the system balances accuracy and other metrics. This thesis first reviews the basic concepts of recommender systems and evolutionary computing. Next, the state of the art is discussed. Building on this, the work presents the recommendation approach and conducts some experiments to evaluate its performance. The results show that it is possible to find a reasonable trade-off in terms of most metrics. However, some functions perform well in terms of the metrics, while a deeper analysis shows that they are not fit for use in a real system.

Keywords— Recommender systems, Genetic programming

Music Recommendation Using Genetic Programming

Robbe Vanhaesebroeck

Supervisors: Luc Martens, Toon De Pessemier
Counsellor: Stefaan Vercoutere

***Abstract*—Recommender systems help people find their way in large volumes of information in a personalised manner. Their use has become widespread in many domains such as music, movies, e-commerce, news,... Traditional recommender systems focus on accuracy but this may be detrimental to user satisfaction. Evolutionary computing is a suitable candidate to optimise other, more user-centric metrics while not completely neglecting the accuracy requirement. This work presents an algorithm that produces a set of recommendations using a traditional recommender system. Next, the items are reranked according to a function constructed by genetic programming. This way, the system balances accuracy and other metrics. This thesis first reviews the basic concepts of recommender systems and evolutionary computing. Next, the state of the art is discussed. Building on this, the work presents the recommendation approach and conducts some experiments to evaluate its performance. The results show that it is possible to find a reasonable trade-off in terms of most metrics. However, some functions perform well in terms of the metrics, while a deeper analysis shows that they are not fit for use in a real system.**

***Keywords*—Recommender systems, Genetic programming**

I. Introduction

Today, recommender systems are indispensable to find relevant content on the internet. People interact with them daily when listening to music or watching movies via online streaming services, when checking the news, when buying items in a webshop,... In the past, recommender systems focused solely on providing accurate recommendations. Unfortunately, this could mean that recommendations became obvious. This is especially a problem for multimedia systems, where users are typically interested in finding new and exciting things. In the last few years, research has acknowledged that besides accuracy, other metrics are important to increase user satisfaction. Common metrics are diversity, coverage, novelty and serendipity. Recommender systems should find a balance between being accurate and exploring the catalogue of items to find new, perhaps surprising discoveries. The recommendation problem is thus modelled as a multi-objective optimisation problem, i.e. a problem where multiple, conflicting objectives are jointly optimised without neglecting any one of them.

One way to find solutions to such problems, is evolutionary computing. This dissertation uses a variant of evolutionary computing known as genetic programming to find a scoring function that reranks items in a list of recommendations. The remainder of this paper first introduces some related work in Section II. Next, Section III presents the design of the algorithm. The algorithm has several parameters that influence computation time and quality of the found solutions. Section IV discusses the parameter settings and the found solutions. Finally, Section V concludes this paper.

II. Related Work

A. Recommender Systems

Through the years, researchers have developed many approaches towards recommending items to users in a personalised way. Traditionally, the community makes a distinction between three large categories [1]:

1) *Content-based recommenders*: users rate items where each item is represented by a description. The system recommends items with a similar description. However, the items may be too similar to previously consumed item resulting obvious and uninteresting recommendations.
2) *Collaborative filtering*: the system finds users who have rated similar items similarly. A target user receives recommendations of items she or he did not consume yet, but that other, similar users liked. This technique does not need content descriptions.
3) *Hybrid systems*: The above approaches each have some benefits and drawbacks. Hybrid systems combine elements of both to combine strengths and overcome weaknesses.

The traditional recommender systems were built for explicit feedback, such as a 5-star rating system.

In practice, most of the feedback is implicit, such as clickstreams. In that case, the system cannot distinguish between a user disliking an item or simply not interacting with it. Researchers have developed several techniques to deal with implicit feedback [10]. One popular example is Bayesian Personalised Ranking [13]. The technique formulates a Bayesian model to create a total ordering relationship on the items for each user.

Typically, recommender systems maximise the accuracy of the recommendations. An item should be as relevant as possible. This work expresses accuracy as the NDCG: the normalised discounted cumulative gain. However, the recommendations can become so accurate that they do not provide a lot of added value. The user already knows the recommended items. Especially in the case of multimedia systems, a recommender system should aid a user in discovering new items or tastes. To achieve that, researchers have developed other metrics:

- *Diversity*: how different are the recommendations in a provided list. If they are all very similar, chances are a user will not like any of them.
- *Novelty*: how likely is it that the user did not know an item before it was recommended. Unpopular items have a higher chance of being novel.
- *Serendipity*: how surprising is the recommendation to the user. This metric is hard to achieve because it is very subjective and encompasses several things. A serendipitous recommendation should be novel, and unlike items a user usually prefers. However, the user should like the item before it can be serendipitous.
- *Coverage*: which fraction of items from the total catalogue appears in at least one recommendation list. This is especially interesting for webshops. Amazon, for example, gets most of its profits from the long tail of items [2], [4].

B. Evolutionary Computing

Evolutionary computing is a metaheuristic optimisation technique based on biological evolution [7]. It maintains a population of individuals. An individual encodes a solution to the problem at hand. Each individual has a fitness, which indicates how well it solves the problem. In every iteration, individuals have a chance of being selected for genetic operations, usually proportional to their fitness value. The genetic operations either randomly change the individual (mutation) or combine elements from two or more parents into one or more children (recombination or crossover). By doing this, the algorithm attempts to find the global optimum, or a solution as close as possible to the global optimum.

A somewhat special form of evolutionary computing is genetic programming. This approach evolves functions or computer programs that are typically encoded as syntax trees. The crossover operation exchanges subtrees between individuals while mutation replaces a subtree with a new randomly grown tree.

Evolutionary computing can also be used to solve multi-objective optimisation problems. In such problems, multiple criteria need to be optimised together. Evolutionary computing can find Pareto-optimal solutions. These solutions perform at least as good as other solutions on all metrics but better on at least one of them. In one run, the algorithm can find multiple solutions that make different trade-offs.

C. Multi-objective Recommender Systems

As mentioned in Section II-A, recommender systems have multiple objectives to optimise. Researchers have made attempts to improve different aspects of recommender systems [11]. Several of those use evolutionary computing. Ribeiro *et al.* combined multiple recommender systems that focused on different aspects, where evolutionary computing determined the weights [14]. Guimarães *et al.* developed the GUARD framework [9]. This framework uses genetic programming to find simple neighbourhood-based ranking functions. Another approach first generates a list of recommendations L and then uses an evolutionary computing technique to select k items from that list to optimise multiple metrics simultaneously [5], [8], [12], [15]–[17].

III. Algorithm Design

Figure 1 shows the proposed algorithm. First, a regular recommender system produces a list of recommendations L. This work uses Bayesian Personalised Ranking [13]. In parallel, for each of the items a set of features is produced. These features can come from an external source, such as content-related features but they can also be based purely on collaborative data. The output from these two steps serve as input for the genetic programming step. This step finds a function that scores each item in the list L. Next, it reranks the items according to the new score and keeps the top k items. Finally, the fitness is determined. The fitness consists of 4 recommender system performance metrics: accuracy, coverage, novelty and diversity. Based on the fitness, some of the individuals are selected for reproduction and mutation, forming a new generation. The step runs for a fixed number of generations.

This design has three goals:

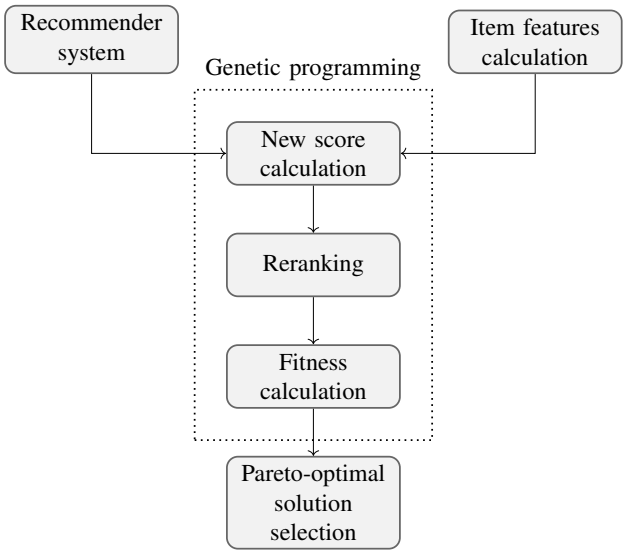

Figure 1: The general flow of the proposed recommender algorithm.

1) It jointly optimises the recommender system metrics.
2) The algorithm can be applied in an environment where a regular recommender system is already running.
3) The time-consuming evolutionary computing step only has to be run sporadically.

Previous research already accomplished the first two goals by first producing a list of recommendations L and subsequently selecting a subset of k items from them [16]. However, with previous designs, the evolutionary computing steps had to be rerun every time a new set of recommendations was needed. The algorithm in Figure 1 uses genetic programming to find a scoring function. This function can be reused every time a new set of recommendations needs to be provided.

IV. Evaluation

The experiments use a subset of 1.4 million user-item-playcount triplets from the Taste Profile subset. This is a subset of the Million Song Dataset [3]. The item features were obtained via AcousticBrainz [6]. First, the recommender system was tuned beforehand. Next, there is one iteration per set of parameters. In each iteration, the recommender system is trained on the training data and produces a list of recommendations per user. Next, these lists serve as input to the genetic programming step together with the item features. The genetic programming step finds several Pareto-optimal scoring functions. To ensure that the scoring functions keep performing well when more user data is available, the traditional recommender system was retrained on train and simulation data and evaluated on the test set. This simulates the real situation where the system needs to provide a new set of recommendations after some time when users have listened to new songs. In that case, the genetic programming step does not need to execute every time, which was one of the design goals. Table I shows the algorithm parameters. For the experiments, the population size and number of generations were varied in a Cartesian grid.

Parameter	Value
# Latent factors	80
Regularisation	0.01
# training epochs	100
Learning rate	0.01
Crossover probability	0.9
Mutation probability	0.05
Tree initialisation minimum height	1
Tree initialisation maximum height	3
Maximum tree height	17
Mutation tree minimum height	0
Mutation tree maximum height	2
Original list size L	50
Final list size k	10
Population size	$\{50, 100, 200, 300\}$
# generations	$\{50, 100\}$

Table I: The algorithm parameters and values.

The experiments show that a larger population size and a higher number of generations find more and better trade-offs. Especially more generations make a big difference. However, the genetic programming step takes proportionally more time. Even large parameter values are no guarantee for finding good trade-offs. Figure 2 shows the found solutions that perform better in terms of coverage, novelty and diversity for population size 100 and 100 generations. Note that the diversity is not explicitly shown in the figure because it remains relatively constant. Be that as it may, all shown solutions have a better diversity than the original recommendation list. Some solutions achieve a significantly higher coverage and novelty than the baseline performance. However, in most cases, this is accompanied by a sharp decrease in NDCG.

The solutions that achieve a reasonable compromise between all metrics are not necessarily good to be used afterwards. The function $\tanh(\cos(\sin(s)))$ where s is the original score achieves a good trade-off in terms of the metrics. However, upon closer inspection, it leaves the list mostly unaltered for some users while for other users it reverses the list completely. In other words, for some users, it focuses purely on accuracy while for the other users it focuses on novelty and coverage. This is not desirable because the goal is to have a compromise on a per-user base. Other functions also rely explicitly on the item features and hence are better to obtain a real trade-

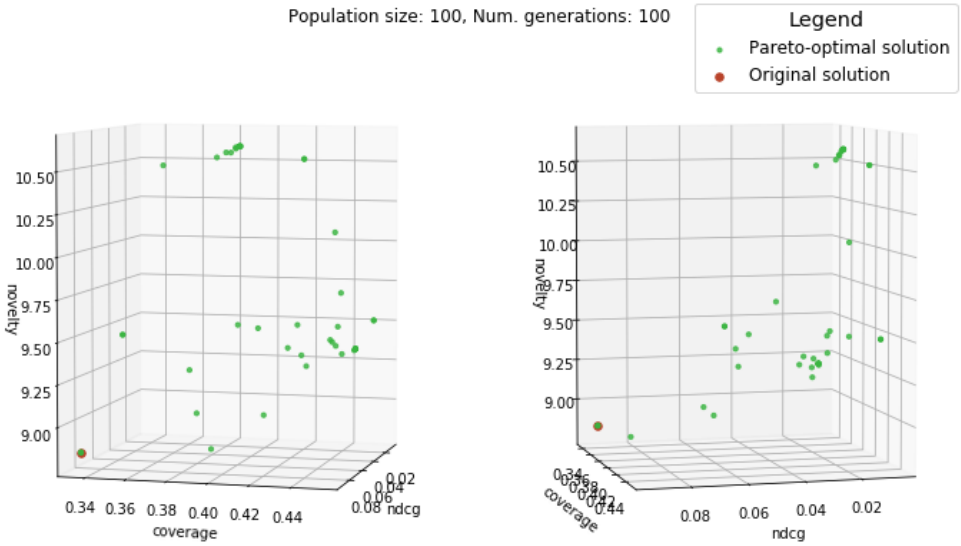

Figure 2: The found solutions that perform better in terms of coverage, novelty and diversity than the original recommendation list.

off. This is the case for $\tanh((\texttt{num\_rat} \cdot s)/-6.74)$. Besides relying on the score, this function also takes into account the number of ratings an item received. The interpretation is that unpopular items will be favoured hence boosting novelty.

V. Conclusion

This work presented the design and evaluation of a recommender system enhanced with genetic programming. The main goal is to balance recommendation accuracy with other performance metrics such as novelty, coverage and diversity. The algorithm can find reasonable trade-offs between the metrics but not all found solutions are practical. Moreover, the presented algorithm can be applied in an environment where a regular recommender system is already running. Finally, the slow genetic programming step only has to be run occasionally, because the resulting reranking functions can be reused.

References

[1] Aggarwal, C. C. *et al.*, *Recommender systems*. Springer, 2016, vol. 1.

[2] Anderson, C., 'The long tail,' *Wired Magazine*, vol. 12, no. 10, pp. 170–177, 2004.

[3] Bertin-Mahieux, T., Ellis, D. P., Whitman, B. and Lamere, P., 'The million song dataset,' in *Proceedings of the 12th International Conference on Music Information Retrieval (ISMIR 2011)*, 2011.

[4] Brynjolfsson, E., Hu, Y. J. and Smith, M. D., 'The longer tail: The changing shape of amazon's sales distribution curve,' *Available at SSRN 1679991*, 2010.

[5] Cui, L., Ou, P., Fu, X., Wen, Z. and Lu, N., 'A novel multi-objective evolutionary algorithm for recommendation systems,' *Journal of Parallel and Distributed Computing*, vol. 103, pp. 53–63, 2017.

[6] (). 'Downloads,' [Online]. Available: https : / / acousticbrainz . org / download (visited on 11/04/2020).

[7] Eiben, A. E., Smith, J. E. *et al.*, *Introduction to evolutionary computing*. Springer, 2003, vol. 53.

[8] Geng, B., Li, L., Jiao, L., Gong, M., Cai, Q. and Wu, Y., 'Nnia-rs: A multi-objective optimization based recommender system,' *Physica A: Statistical Mechanics and its Applications*, vol. 424, pp. 383–397, 2015.

[9] Guimarães, A., Costa, T. F., Lacerda, A., Pappa, G. L. and Ziviani, N., 'Guard: A genetic unified approach for recommendation,' *Journal of Information and Data Management*, vol. 4, no. 3, pp. 295–310, 2013.

[10] Jannach, D., Lerche, L. and Zanker, M., 'Recommending based on implicit feedback,' in *Social Information Access*, Springer, 2018, pp. 510–569.

[11] Kaminskas, M. and Bridge, D., 'Diversity, serendipity, novelty, and coverage: A survey and empirical analysis of beyond-accuracy

objectives in recommender systems,' *ACM Transactions on Interactive Intelligent Systems (TiiS)*, vol. 7, no. 1, pp. 1–42, 2016.

[12] Lin, Q., Wang, X., Hu, B., Ma, L., Chen, F., Li, J. and Coello Coello, C. A., 'Multiobjective personalized recommendation algorithm using extreme point guided evolutionary computation,' *Complexity*, vol. 2018, 2018.

[13] Rendle, S., Freudenthaler, C., Gantner, Z. and Schmidt-Thieme, L., 'Bpr: Bayesian personalized ranking from implicit feedback,' *arXiv preprint arXiv:1205.2618*, 2012.

[14] Ribeiro, M. T., Lacerda, A., Veloso, A. and Ziviani, N., 'Pareto-efficient hybridization for multi-objective recommender systems,' in *Proceedings of the sixth ACM conference on Recommender systems*, 2012, pp. 19–26.

[15] Wang, S., Gong, M., Li, H. and Yang, J., 'Multi-objective optimization for long tail recommendation,' *Knowledge-Based Systems*, vol. 104, pp. 145–155, 2016.

[16] Wang, S., Gong, M., Ma, L., Cai, Q. and Jiao, L., 'Decomposition based multiobjective evolutionary algorithm for collaborative filtering recommender systems,' in *2014 IEEE Congress on Evolutionary Computation (CEC)*, IEEE, 2014, pp. 672–679.

[17] Zuo, Y., Gong, M., Zeng, J., Ma, L. and Jiao, L., 'Personalized recommendation based on evolutionary multi-objective optimization [research frontier],' *IEEE Computational Intelligence Magazine*, vol. 10, no. 1, pp. 52–62, 2015.

Contents

List of Figures

List of Tables

List of Abbreviations

AMAN all missing values as negatives.

AMAU all missing values as unknown.

BPR Bayesian Personalised Ranking.

CF collaborative filtering.

DCG discounted cumulative gain.

EA evolutionary algorithm.

EC evolutionary computing.

GP genetic programming.

MOP multi-objective optimisation problem.

MSE mean squared error.

NDCG normalised discounted cumulative gain.

RMSE root mean squared error.

RS recommender system.

Chapter 1

Introduction

The internet contains an unbelievable amount of information on anything imaginable, and more information is being added every second. News sites publish new articles every day, people post on their social media, webshops sell items online, and multimedia streaming services provide catalogues containing millions of items. Owing to the volume and velocity, people cannot keep up and they may quickly become overwhelmed. This phenomenon is commonly known as the information overload problem. To assist people in finding the product, article, or service they need, RSs have been proven to work very effectively [12].

A recommender system is a type of system that filters out unnecessary and irrelevant information for people in a personalised way to help them in finding the things they need in the colossal collection of data. To achieve this goal, they operate on users' historical consumption patterns to learn which type of items they typically prefer. This data can take several forms such as 5-star ratings, likes and dislikes, purchase history, clickstreams,... Once the system has modelled the user's interests, it attempts to find items that match their tastes. Traditionally, there are three approaches to doing this [4]. Content-based RSs try to find items similar to what a user has interacted with before based on descriptions of the items. In other words, each item is represented as a vector of features and the system searches for items whose features are similar to the features of previously consumed items. Collaborative filtering (CF), on the other hand, does not need content-specific features to recommend items, but identifies other users with similar histories to the target user and then recommends items that those users have consumed, to the target user. Finally, hybrid RSs combine elements from both techniques to achieve better results and overcome the weaknesses of the separate approaches.

When building a RS, the focus is usually on making the recommendations as accurate as possible. If a user receives a list of 10 recommended items, ideally the most relevant item should be on top, followed by the second most relevant item and so on. Research has pointed out that the most accurate recommendations are not necessarily the most appropriate to the users of a

system, and that user satisfaction with the system may decrease by focusing on (only) optimising standard accuracy metrics [41]. Suppose a user likes the songs *Master of Puppets* and *Fade To Black* by *Metallica*. A recommender system may recommend other songs by the same band. In that case, the system has successfully learned that the user likes that particular band and hence provides accurate recommendations. However, the user will probably not be satisfied with the system, because she/he already knew most of the recommended songs.

It is hence clear that a too high focus on accurate recommendations may decrease user satisfaction with the system. This is true to some extent for almost all environments that use RSs, but it becomes especially important for multimedia applications such as music, movies or book recommenders. Users of such a system want to find accurate items they like, but the recommendations should not be unduly obvious because then the RS would not be of much help: the user could have found these items without help. Instead, an important goal of RSs in such situations becomes helping users explore the catalogue of items hopefully finding new and exciting items that the user did not know before but likes nonetheless. In this respect, it can be helpful to recommend items that are unpopular and unknown because this increases the probability that the user was indeed not familiar with the item before. Moreover, RSs typically provide a list of several recommendations. If all the items in that list resemble each other a lot, the probability increases that the user will not like any of them. Therefore, it may be beneficial to make the recommended items somewhat different to each other (within reasonable bounds). Additionally, for webshops, for example, there is an effect known as the *long-tail* phenomenon [6]. This means that a very small minority of the items is very popular while most of the other items have very few interactions. Yet, at Amazon, for example, it has been observed that most of the sales come from the long tail [15]. Hence, explicitly making recommendations for as many different items as possible at the system-wide level may increase sales and profits. Consequently, researchers have developed alternative performance metrics to quantify these behaviours to improve RS performance for those metrics as well. That is not to say these systems should disregard accuracy altogether: all the performance metrics are important and should be jointly optimised.

Since recommender systems no longer only focus on optimising accuracy, the problem becomes a multi-objective optimisation problem (MOP). Most techniques to solve such problems are based on optimising a weighted sum of the different criteria where a human in the loop can set the weights differently to achieve different trade-offs [40]. Be that as it may, one technique named evolutionary computing (EC) can find so-called Pareto optimal solutions to the problem. This means they try to find solutions that perform well on all criteria, without hurting the performance on the other criteria. EC techniques simulate biological evolution. Given a population of individuals that encode a solution to a certain optimisation problem, under the

influence of genetic operations like crossover (sexual reproduction) and mutation, the algorithm will generate a new generation of offspring from individuals of the previous generation. Each individual receives a *fitness* score, which indicates how well the solution encoded by the individual solves the problem at hand. By choosing individuals with a higher fitness to create offspring, over time, the fitness of the population is expected to increase and hence a solution as close as possible to the global optimum can be discovered [21]. Over the decades, many researchers have developed variants of this basic scheme to solve a diverse set of problems. One of these is called GP. This variant attempts to find a function that given a predefined input produces an output to optimise one or more criteria.

The goal of this work is to use GP to build a RS that not only focuses on accuracy but also the other RS performance metrics mentioned above. Besides this main goal, the system should attain some additional objectives. It should be easy to adopt this system in an environment where an existing RS is already running. Additionally, the time-consuming GP step should not be rerun every time the system produces a new set of recommendations. These goals are accomplished by using a two-step process, where first a regular accuracy-focused RS generates a set of recommendations for every user, and then a reranking of those results takes place based on a function that has been evolved using GP. That way, each time the system generates new recommendations, it can reuse the same function.

The remainder of the report looks as follows. First, Chapter 2 will discuss some general concepts related to RSs. Second, Chapter 3 gives an introduction to EC techniques and MOPs. With this theoretical background in mind, Chapter 4 will review recent research and the state-of-the-art regarding RS as MOPs. Chapter 5 lists the main data sources used for experimentation, and explains how the dataset used was built. Then, in Chapter 6 the design of the proposed algorithm is deliberated, following a top-down approach. The experimental results are given and examined in more detail in Chapter 7. Finally, Chapter 8 concludes the report with a summary and some future research directions.

Chapter 2

Recommender Systems

The current chapter discusses some theoretical background related to recommender systems (RSs). First, Section 2.1 outlines the general context of a RS. Section 2.2 summarises the main classes, their properties and some important examples from literature. Then, Section 2.3 spends some time on RSs for implicit feedback data. Finally, Section 2.4 lists some metrics for measuring RS performance.

2.1 Introduction

Recommender systems use historical data on users' interests for certain items to predict possibly relevant items in the present and near future. Typically, this historical data is represented as a matrix R called the *rating* matrix, *utility* matrix or simply *user-item* matrix. Each row represents a single user while each column represents an item. The elements of the matrix r_{ui} indicate the preference of a user u for a particular item i. Often, this preference value is an integer number on a scale of 1 to 5 or 1 to 10 (e.g. star ratings) where the minimum denotes a strong dislike while the maximum denotes a strong preference for the item. A rating scale could also be binary, where a 1 represents a preference and a -1 a dislike. However, there are also unary ratings or implicit feedback data such as purchase history in a webshop, song playing history, video view time, ... In that case, if an entry r_{ui} has a filled-in value, this indicates some interest of the user u for that item i. If an entry in the matrix is not filled in in the matrix, it could be either because the user dislikes that item, but it could also be because the user simply did not interact with the item yet [5].

2.2 Classes of Recommender Systems

There are three big classes of recommender systems: content-based recommenders, collaborative filtering (CF) and hybrid systems. Each of these is briefly discussed in the sections below.

2.2.1 Content-based Recommender Systems

Content-based RSs utilise item descriptions in combination with user rating behaviour to discover which type of items a user likes. For example, suppose Bob likes *Master of Puppets* by *Metallica*. This song belongs to the genre 'metal', and release year 1986. A content-based recommender will then look for songs with similar properties, such as *Angel of Death* by *Slayer* which is also a 'metal' song released in 1986.

Content-based RSs can recommend new items for which sufficient rating data is not available. This is because a user may have rated similar items in the past, and hence the system understands that the active user will presumably like that item as well. However, content-based systems have several disadvantages. First, the provided recommendations are rather obvious. The system will always look for similar content and hence recommended items will typically not be diverse or surprising.

2.2.2 Collaborative Filtering Systems

Collaborative filtering leverages ratings from the community to find relevant items for a target user. Within the CF systems, there are two categories: neighbourhood-based CF and model-based CF. These will be presented in this section, followed by a comparison of both methods relative to each other, and relative to content-based filtering methods.

	Master of Puppets	*The Trooper*	*Highway to Hell*	*Method Man*	*Juicy*
Alice	1	-	-	4	-
Bob	5	-	4	3	-
Carol	2	2	-	5	5
Dave	5	5	4	-	-
Ellen	-	2	-	4	4

Table 2.1: Example of a rating matrix

Neighbourhood-based Collaborative Filtering Within the neighbourhood-based CF methods, there are essentially two dual formulations of the recommendation problem, in the sense that they can mostly use the same formulas and algorithms, adapted to the specific situation:

- *User-user systems*: for a given target user, the system tries to determine the most similar users. The items liked by these users are probably also relevant for the target user if the user did not rate them yet. Take the rating matrix in Table 2.1 as an example. Suppose the system wants to recommend a song to Bob. Bob has rated *Master of Puppets* and

Highway to Hell highly. The system looks for similar users and finds that Dave gave similar ratings as Bob. Since Dave rated *The Trooper* highly, this is probably a song Bob will like as well.

- *Item-item systems*: items that are comparable to items liked by a target user will probably also be interesting to the user. Consider once again the same rating matrix from Table 2.1. Suppose the system wants to recommend items to Alice. Since Alice has rated *Method Man* highly, the system looks for songs with similar ratings. The song *Juicy* has received similar ratings and probably *Juicy* would make a good recommendation for Alice.

In practice, item-item CF systems often achieve better accuracy than user-based systems. This is because, in user-based systems, other users' ratings are used to estimate ratings on unrated items. Users are typically more complex to model than items and they may have overlapping but different interests to the target user. Additionally, user interests may evolve over time and hence result in less accurate recommendations. An item, on the other hand, is static.

Neighbourhood-based models are intuitively simple and hence their recommendations can easily be interpreted. However, their performance degrades when the rating matrix is sparse. If none of Bob's neighbours in a user-based system rated the song *Toxicity* by *System of a Down*, the system will not be able to predict an accurate rating for that song, even though the song may be very relevant to Bob.

Model-based Collaborative Filtering A second class are the model-based CF techniques. This class uses machine learning techniques to predict missing ratings in the user-item matrix. Different machine learning techniques can be generalised to be used for the recommendation task. However, the most commonly known method is the matrix factorisation model. The basic idea is to approximate the rating matrix with dimensions $m \times n$ by a product of a user matrix U with dimensions $m \times k$ and an item matrix V of dimensions $k \times n$. The interpretation is that the rows of the user matrix represent the users' affinity for each of k latent factors while the columns of the item matrix indicate the affinity of the items for each of the latent factors. Algorithms like singular value decomposition (SVD) or Alternating Least Squares (ALS) estimate the user and item matrices based on the observed ratings. Then, to determine the predicted rating of the user u for an item the dot product between U_u and V_i is made.

A concrete example could be the following. Consider the binary rating matrix in Table 2.2. This matrix could be factorised as follows:

	Master of Puppets	*The Trooper*	*Highway to Hell*	*Method Man*	*Juicy*
Alice	-1	-	-	1	-
Bob	1	-	1	1	-
Carol	-1	-1	-	1	1
Dave	1	1	1	-	-1
Ellen	-	-1	-	1	1

Table 2.2: Example of a binary rating matrix

$$\begin{pmatrix} -1 & \cdot & \cdot & 1 & \cdot \\ 1 & \cdot & 1 & 1 & \cdot \\ -1 & -1 & \cdot & 1 & 1 \\ 1 & 1 & 1 & \cdot & \cdot \\ \cdot & -1 & \cdot & 1 & 1 \end{pmatrix} = \begin{pmatrix} -1 & 1 \\ 1 & 1 \\ -1 & 1 \\ 1 & -1 \\ -1 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix}$$

The latent factors can have a direct interpretation. In this example, the first row of the item matrix has a 1 for the rock and metal songs and a 0 for the hip-hop songs, while the second row has a 0 for rock and metal songs and a 1 for the hip-hop songs. The user matrix also shows some patterns, for instance, Alice does not like metal but does like hip-hop. Similar reasoning follows for the other users.

Model-based techniques are less intuitive than neighbourhood-based techniques. While the latent factors can sometimes have an interpretation, this interpretation is not always clear and hence it may be unknown why a particular item is recommended to a certain user. However, model-based techniques deal much better with sparsity issues since they represent the information of the rating matrix in a much more compact way. This also benefits the storage requirements and prediction speed.

Pros and Cons of Collaborative Filtering Compared to content-based filtering, CF systems can typically recommend more diverse items than content-based systems. The system can pick up on seemingly counterintuitive correlations. For example, hip-hop fans may like *Sabotage* by *Beastie Boys* even though this is arguably more a rock song than a hip-hop song. Additionally, CF techniques can easily be used in many different contexts. Since they only use rating information it is unimportant which type of item is being recommended, whether it is music, movies, news,... The same cannot be said for content-based systems since they rely on highly domain-specific features of the items.

However, CF methods, like content-based ones, suffer from the cold-start problem. Just like in content-based methods, CF techniques cannot provide recommendations to new users, since

the system does not have any historical data and hence does not know what they like. However, content-based systems could still recommend new items without sufficient rating information. This is not the case for CF. This problem causes a paradox because the RS will not recommend new items because they do not have ratings, but nobody will rate these items because they are not recommended.

2.2.3 Hybrid Recommender Systems

The final class of recommender algorithms are hybrid systems. As mentioned before, these systems combine elements from both content-based systems and CF to solve some of the typical problems of either class. Hybrid RSs can often reach better performance than standalone CF or content-based systems. They can be constructed either by creating an ensemble of several off-the-shelf algorithms or by designing a custom algorithm that combines data from different sources. Perhaps the most well-known hybrid RS is the winning entry of the Netflix Prize contest: *Bell-Korr's Pragmatic Chaos* [35].

2.3 Implicit Feedback

Most recommendation algorithms were originally developed for explicit feedback systems such as a 5-star rating system, like and dislike buttons,... The problem with explicit feedback is that it can be hard to come by because it requires active user participation. On the contrary, it is easy to collect implicit feedback in almost any information processing system. In a webshop, for example, a user's purchase history, the time spent looking at certain items, which items the user clicked, the items that were put in the shopping cart,... may all be used to derive the user's preferences for certain items. For a music streaming service, the system may keep track of which songs the user listened to, how many times that song was played, whether the song was played entirely or whether the user skipped to the next song after 30 seconds,...

Since implicit feedback is more abundant, and in many environments more practical, researchers have developed some techniques to recommend items based on implicit feedback rather than explicit feedback [33]. The simplest and most intuitive approach is to find a way to convert the implicit feedback into explicit rating data and then use standard recommendation algorithms for the task at hand. This is not a straightforward task since implicit feedback often contains positive-only data with little to no information on what a user actually dislikes. If an item was not consumed this can either be because the user dislikes the item, or simply because the user does not know about the item.

Two ways of constructing a rating matrix from implicit feedback data are either all missing values as negatives (AMAN) or all missing values as unknown (AMAU) [45]. In AMAN, positive

examples receive a label '1' while missing items receive a label '0'. This makes it possible to use existing CF techniques without much adaptation. In AMAU on the other hand, the missing items receive a label '?' and the algorithms only work on the positive examples. This technique requires that the existing algorithms are modified to only deal with positive examples because these may lead to trivial solutions. Both methods are oversimplifying the situation because, in reality, the missing values contain both unknown positive examples as well as negative examples.

A more complicated approach is to embrace the implicit feedback data and develop specialised techniques to produce recommendations from that data. Pan *et al.* developed a method based on low-rank approximation of the rating matrix that assigns confidence weights to the signals to express the probability that a signal is correctly interpreted as positive or negative [45]. They hence have a more nuanced view of the rating matrix where a rating from a user on an item can be positive or negative with a certain probability.

An interesting example of an implicit feedback RS is the Bayesian Personalised Ranking (BPR) model developed by Rendle *et al.* [46]. This model takes a different approach to recommend items. The previous models essentially tried to predict the rating a user u would give to an item i and then ranks the items based on the predicted ratings. The BPR method assumes that the predicted ratings are irrelevant and only the ranking of the items is important: the most relevant item should be at the top of the list. Therefore, the model tries to reconstruct for each user a total ordering of the items. For each user-item-item triplet, if user u prefers item i over item j:

$$i >_u j \qquad u \in U \;\; i, j \in V \tag{2.1}$$

The Bayesian formulation of this problem consists of maximising the posterior probability:

$$p(\Theta| >_u) \propto p(>_u |\Theta) p(\Theta) \tag{2.2}$$

After some further simplifying assumptions, the final form eventually becomes tractable. The authors also developed a variant of gradient descent that converges faster than the classical methods for their objective function. Their paper provides more detailed information [46].

2.4 Measuring Recommender System Performance

The goal of RSs is to help a user find relevant items, and especially in the case of multimedia systems like music or movie streaming, help the user explore items that she/he did not know before. The best way to measure this is through user surveys. However, these can be cumbersome to set up and require active user participation. Therefore, the most commonly used approach is to measure performance on a historical data set. The most obvious way to quantify performance

is to measure the accuracy of the predictions, although other metrics are important for the user experience as well.

2.4.1 Accuracy

To measure the accuracy of the system, a part of the ratings is held out as a test set. The system then makes predictions for the missing ratings and measures the differences between the predicted and actual ratings. The error of the system is commonly expressed in terms of the mean squared error (MSE). Let the test set of ratings be denoted by $\mathcal{T}$, then the mean squared error on this set is defined as:

$$\text{MSE} = \frac{\sum_{(u,i)\in\mathcal{T}}(\widehat{r}_{ui} - r_{ui})^2}{|\mathcal{T}|}. \tag{2.3}$$

Because of the square operation, larger errors decrease performance more than smaller ones. The MSE can quickly grow large. Therefore, an alternative measure is the root mean squared error (RMSE). This metric expresses the error in units of ratings instead of units of squared ratings, resulting in a number that is easier to interpret.

$$\text{RMSE} = \sqrt{\text{MSE}} = \sqrt{\frac{\sum_{(u,i)\in\mathcal{T}}(\widehat{r}_{ui} - r_{ui})^2}{|\mathcal{T}|}} \tag{2.4}$$

These accuracy metrics are often used in practice, for example for the Netflix Prize [9]. However, there are some problems. Ratings on popular items heavily influence both metrics. If a system is highly accurate at predicting ratings for popular items but makes large errors for the less conventional items, the MSE and RMSE might still be better than for a system that makes moderate errors on all items, which is undesirable.

One might also note that most systems provide a list of recommendations and not the actual predicted ratings. To quantify the quality of a list of recommendations, one can compare the ranking provided by the RS to the ideal ranking of items. This can be done using a correlation metric such as the *Spearman ranking coefficient*. A final type of accuracy metrics is utility-based methods, which use both the ratings and the rankings to quantify how useful a recommendation list might be to a user. A user would like the items with the highest ground truth ratings to be at the top of the recommendation list. One way to quantify this, is through the NDCG:

$$\text{NDCG}(L, u) = \frac{\text{DCG}(L, u)}{\text{DCG}(L_{\text{ideal}}, u)}, \tag{2.5}$$

where L is the recommendation list for user u, L_{ideal} is the ideal recommendation list and DCG is the discounted cumulative gain:

$$\text{DCG}(L, u) = \sum_{i=1}^{|L|} \frac{r_{ui}}{d(i)}, \tag{2.6}$$

with $d(i)$ a discount function. A popular choice is $d(i) = \log_2(i + 1)$.

2.4.2 Other Metrics

As already mentioned before, accuracy does not tell the whole story [41]. A music RS that only recommends music that a user has already listened to is very accurate. Unfortunately, the user will not be satisfied with the system because it does not recommend interesting items. Therefore, it is necessary to investigate other metrics as well. These will focus more on other aspects of the user experience. One problem is that these aspects are typically more subjective and hence harder to quantify. Some of these metrics are now discussed in more detail, and if possible a quantitative formula is given.

Diversity

Diversity can be defined as the extent to which recommendations within a single recommendation list differ from each other. A list of songs of the same genre, same bands and same albums has a very low diversity. From a user's perspective, this may be detrimental to the quality of the RS. The recommendations, even though they may be very accurate, do not provide a lot of added value to the user, since the user most likely already knows the type of recommended items. By increasing the diversity of the recommendations, the probability increases that the list contains at least some useful recommendations. Additionally, it encourages users to expand their knowledge or interests.

To measure diversity, one needs to be able to express the difference or similarity between items. The best approach is to use content descriptions since these express the actual properties of an item. Alternatively, if these are not available, one could for example use the latent factors from a matrix factorisation, or even the rating vectors for each item. To calculate the diversity of a recommendation list, one computes the average pairwise distance between the items of the list [34]:

$$\mathrm{div}(L) = \frac{\sum_{i \in L} \sum_{j \in L \setminus \{i\}} \mathrm{dist}(i, j)}{|L|(|L| - 1)}, \tag{2.7}$$

with dist a distance function between two items. The distance function differs widely in literature, from the inverse of Jaccard similarity or Pearson correlation [50], the inverse cosine similarity [24] or even Hamming distance in the case of using rating vectors [14].

Novelty

The term novelty refers to the probability that a user did not know a recommended item before. A popularity-based recommender is unlikely to provide novel recommendations because it focuses solely on items that as many users as possible recognise. Intuitively, one can expect that unpopular items have a much higher chance to be novel to a user. This intuitive notion is translated into the commonly used formula to express the novelty of an item. Namely, the

novelty is often defined in terms of the self-information of an item [34], [50]:

$$\text{nov}(i) = -\log_2 p(i) = -\log_2 \frac{|\{u \in U | r_{ui} \neq \emptyset\}|}{|U|}. \tag{2.8}$$

In other words, the less popular an item is, and the further it is situated in the long tail, the higher its novelty. To express the novelty of a recommendation list, one simply computes the average novelty of all items in the list:

$$\text{nov}(L) = \frac{\sum_{i \in L} -\log_2 p(i)}{|L|} \tag{2.9}$$

Serendipity

Closely related to novelty is serendipity, a concept that can be described as the degree to which a recommendation surprises the user. Serendipity is arguably one of the most difficult metrics to define and to achieve because it is highly subjective. For an item to be serendipitous it should be unexpected, relevant and novel to the user [36]. However, defining when a recommendation is unexpected is not straightforward. One possibility is to compare the recommendations from a RS to a naïve baseline system that generates obvious recommendations, such as a content-based RS [25]. An item that is recommended by the system under test, but not by the baseline system is considered to be unexpected. The usefulness of an item is then to be judged by the user and expressed as $u(i) = 1$ if the item is useful and $u(i) = 0$ otherwise. Suppose the list of unexpected recommendations from the system under test is expressed as L_{unexp} Then, the serendipity of a recommendation list is expressed as follows:

$$\text{srdp}(L) = \frac{\sum_{i \in L_{\text{unexp}}} u(i)}{|L_{\text{unexp}}|} \tag{2.10}$$

It is easy to see that this definition is by no means practical for offline evaluation since it requires active user participation to express which items are useful and which are not.

Coverage

Finally, the coverage of a RS is a system-wide metric that expresses which fraction of items from the catalogue is recommended to at least one user. Increasing the coverage of a system may have positive effects on both user satisfaction and product sales [3], [6]. Additionally, one expects that when the coverage of the system increases, it may also have a positive effect on metrics like diversity, novelty and serendipity although this relation has not been studied in detail [34]. Formally, the coverage is defined as follows:

$$\text{cov} = \frac{|\cup_{u \in U} L_u|}{|I|}, \tag{2.11}$$

with L_u the recommendation list for user u and I the set of all items in the catalogue.

Chapter 3

Evolutionary Computing

The current chapter introduces general concepts from evolutionary computing (EC). Section 3.1 places EC techniques in the context of optimisation algorithms and metaheuristics. Next, Section 3.2 will describe the general schema, and Section 3.3 will zoom in on the main components. Section 3.4 summarises several well-known variants of the basic scheme. Finally, Section 3.5 concludes the chapter with a short explanation of multi-objective optimisation problems, and how to use EC to solve such problems.

3.1 Introduction

EC techniques or evolutionary algorithms (EAs) are algorithms that draw their inspiration from the biological evolution process. They form a subclass of the nature-inspired metaheuristics. Metaheuristics are algorithms that approximately solve so-called *hard optimisation problems*, i.e. problems for which it is not possible to find an optimal solution or a solution within a guaranteed bound of the optimum in a reasonable amount of time [13]. The heuristics are *meta* because they can be applied to a wide range of such problems with little adaptation. Common examples of these metaheuristics include simulated annealing, tabu search, iterated local search, and EC.

Several variations of the basic EA exist, but they all share the same underlying ideas. Given a population of individuals, the individuals will compete for survival. Through sexual reproduction and mutation, the individuals produce a new generation in the hope that generation after generation, the fitness of the population increases [21]. In the following section, the general schema of EAs is discussed, followed by a description of the most important components.

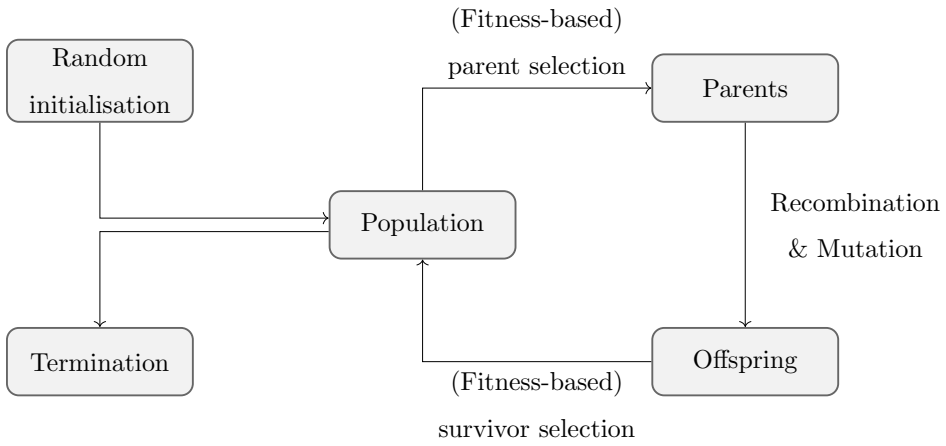

Figure 3.1: The general evolutionary computing scheme [21].

3.2 Schema

The basic schema for EC is given in Figure 3.1. In an EA, the population is key. A population is a group of individuals. One individual encodes a solution to some optimisation problem. The first step initialises the population with a group of random individuals. Second, the algorithm measures the fitness of each individual and chooses some individuals as parents for the next generation. Typically, individuals with the highest fitness are most likely to be selected. The parents create offspring through genetic operations called recombination and mutation. Finally, the survivors are selected from the created offspring to form the next generation. This process repeats for a fixed number of iterations, after which the algorithm terminates, hopefully with a good solution to the optimisation problem.

3.3 Components

3.3.1 Fitness

The fitness function or evaluation function embodies the requirements a solution should strive to meet. The goal of the EA is to maximise the fitness of their individuals and population. If the problem at hand is a minimisation problem, one can reform the original objective function into a fitness function to maximise by putting a minus sign in front of it.

3.3.2 Individuals

Individuals encode a solution to the problem. Different variants of the algorithm use different encodings. For some problems, a solution can be encoded using a binary string. Other problems require a vector of real values or even tree representations. The separate elements of an individual are called *genes*, while the specific value of a gene in an individual is called an *allele*.

3.3.3 Parent Selection

Once the fitness is determined for each individual in the population, the algorithm picks several individuals based on their quality. The individuals will be used in one of the variation operators to create new, better offspring. Different schemes exist to select parents. In some variants, all the individuals are chosen and combined with another random individual to create new offspring. In others, the parent selection is probabilistic, proportional to the fitness of individuals. This means high-quality parents have a higher chance of being selected.

3.3.4 Variation Operators

The variation operators are the driving force to create a diverse population. This is necessary because if all the individuals resemble each other closely, a large part of the search space remains unexplored, decreasing the chance of finding a global optimum. The variation operators are inspired by the genetic processes of recombination and mutation.

Mutation is a unary operator that takes one parent individual, and changes it randomly to create a child. For binary encodings, a common choice is to induce a bit flip at a random position. For real values, one could add a Gaussian noise sample to each of the elements in the vector. Mutation occurs with a given (small) probability, meaning not all selected individuals undergo mutation. Some EC variants dynamically vary the mutation probability. The reasoning is that in the beginning, mutation helps to explore the search space more broadly. Towards the end of the algorithm, the individuals are expected to be close to a good solution, so it makes less sense to significantly change them.

Recombination on the other hand is a binary (or k-ary) operator, also known as crossover. The operator blends parts from two or more parents to create one or more offspring. Like mutation, recombination is stochastic too. Determining which parents and which parts of the parents to combine depends on (pseudo-)random drawings. The main goal of recombination is to merge two or more individuals to hopefully combine their strengths and to create new, stronger offspring. Common recombination operators for vector representations (binary or real-valued) are:

- One-point crossover: selects a random position in the vector and swaps the parts of each parent.
- k-point crossover: selects k point and swaps the different parts.
- Uniform crossover: determines whether to switch the alleles for each gene separately with a given probability.

3.3.5 Survivor Selection

Finally, the algorithm selects survivors from the created offspring based on their quality. It is possible to generate more offspring than the given size of the population. In that case, a subset of the created offspring is selected based on quality. A different approach is to create a smaller number of offspring that replace a part of the preceding population. The replaced part of the population could consist of the weakest individuals, or it could be determined stochastically. Another possibility could be generational where the created offspring completely substitutes the previous population.

3.4 Evolutionary Computing Variants

EC variants differ mostly in the way they represent their individuals. Of course, this also implies that certain variants are more suitable than others to solve certain problems. Arguably the most well known EC technique is the *genetic algorithm* [29]. In its 'canonical' form a genetic algorithm has a binary representation, selects parents proportional to the fitness of individuals. Genetic algorithms implement both recombination and mutation, the latter with a low probability. The new offspring replaces the old offspring in every generation.

A second variant of EAs is called *evolutionary strategies* and it dates back to the early 1960s [11]. These differ from the genetic algorithms by using real-valued vectors to represent individuals. For crossover, they typically use discrete crossover, where certain parts of the vectors are exchanged between two parents. Another frequent alternative is intermediary crossover, where a new allele is formed through a linear combination of the parent alleles. For the mutation operation, they typically add Gaussian random noise to the alleles. An interesting feature of evolutionary strategies is that they dynamically adapt the mutation step size where the mutation step parameter is included in the representation of the individual and can hence also change through recombination and mutation. Finally, survivors are deterministically selected: after the offspring has been created, the fittest individuals from the parents and offspring are selected, to speed up convergence.

Related to evolutionary strategies, *evolutionary programming* was also developed in the 1960s [21]. In the beginning, the researchers principally used finite state machines as individuals, but now they mostly use real-valued vectors just like evolutionary strategies. The main difference is that evolutionary programming does not use recombination but only relies on mutation through Gaussian noise on the alleles. Each parent hence creates one offspring through mutation. The survivors are probabilistically selected through tournament selection. Several randomly selected individuals compete with each other in a tournament, and only the winner survives. That way, less fit individuals still have the chance of surviving causing a better

exploration of the search space. Through the years, evolutionary programming has steadily evolved which led to the development of so-called *meta-evolutionary programming* which introduced self-adaptation of mutation step sizes. This blurred the distinction between evolutionary programming and evolutionary strategies.

The last major variant of EC techniques is *genetic programming* [17], [37]. It deviates significantly from the previous techniques because the individuals are usually not represented as vectors of bits or floating-point numbers. This technique essentially operates on a set of computer programs or mathematical expressions that are usually represented as trees. As crossover operation, GP exchanges subtrees between parents, while for mutation, a random change is done in the trees. Like genetic algorithms, the newly created offspring generation replaces the old generation. While mutation does occur in GP, it is advised to set the mutation rate very low [7], [37].

Many other variants and alterations of these basic schemes exist: learning classifier systems, differential evolution, particle swarm optimisation, estimation of distribution, co-evolutionary algorithms, cultural evolution, scatter search, ant colony optimisation, artificial immune systems... [13], [21]. The specifics of each algorithm are different but they almost always follow a similar scheme to the one explained in Section 3.2.

3.5 Multi-objective Optimisation

In many optimisation problems, there is an objective function that needs to be optimised, subject to some restrictions. However, in some problems there can be multiple, sometimes conflicting requirements that should each be optimised. In that case, it becomes difficult to define when a solution is optimal. A solution may optimise one of the objective functions, but achieve poor results for the other requirements. One possibility to avoid this that is often used in literature is to define an optimal point in the sense of *Pareto-optimality*.

Definition 3.1 (Dominance) *Given a function* $\mathbf{F}(\mathbf{x}) = (F_1(\mathbf{x}), F_2(\mathbf{x}), \ldots, F_n(\mathbf{x}))$ *and two points* $\mathbf{x}, \mathbf{x}^* \in \mathbf{X}$, $\mathbf{x}^*$ *dominates* $\mathbf{x}$ *if and only if* $\mathbf{F}(\mathbf{x}) \leq \mathbf{F}(\mathbf{x}^*)$ *and* $\exists i \in 1, \ldots, n : F_i(\mathbf{x}) < F_i(\mathbf{x}^*)$. *In other words,* $\mathbf{x}^*$ *is at least as good as* $\mathbf{x}$ *on all criteria, and better for at least one.*

Definition 3.2 (Pareto-optimality) *Given a set of functions to be optimised* $\mathbf{F}(\mathbf{x}) = (F_1(\mathbf{x}), F_2(\mathbf{x}), \ldots, F_n(\mathbf{x}))$, *a point* $\mathbf{x}^* \in \mathbf{X}$ *is Pareto-optimal if and only if there exists no other point* $\mathbf{x} \in \mathbf{X}$ *that dominates* $\mathbf{x}^*$.

In other words, a Pareto-optimal solution is not dominated by any other solution; no solution performs better for one of the objectives and at least as good for all other ones. For a given set of

functions, there are usually multiple Pareto-optimal points that form the so-called *Pareto-front* or *Pareto-set*. This is shown in Figure 3.2. No solution performs better on both objectives than the solutions on the red line.

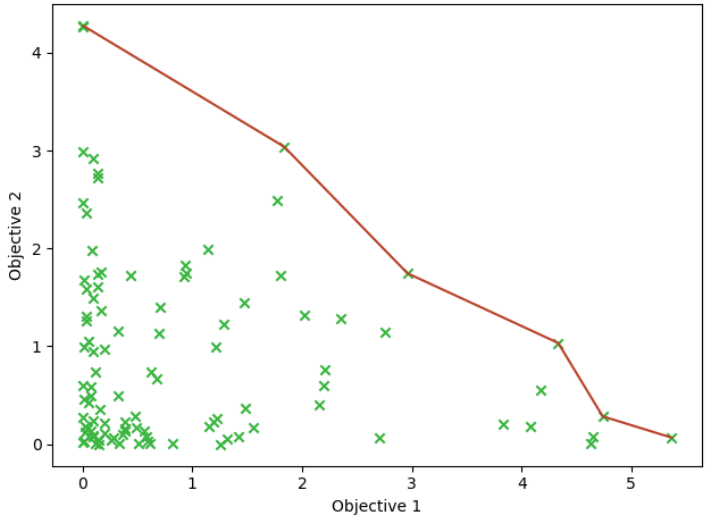

Figure 3.2: An illustration of some Pareto-optimal solutions compared to other solutions.

One frequently used method to solve a MOP is through scalarisation [40]. The different objective functions are joined into a single objective function that is then solved using standard methods. Sometimes this is just a weighted sum of all the objective functions, but more complex designs exist that incorporate parameters to model the preferences of the decision-makers. Another method could be to work lexicographically. This strategy orders the results according to the most important optimisation function first, followed by another and so on. By varying the parameters, different Pareto-optimal points can be found.

EAs can also be used to solve MOPs. The advantage of this technique is that the objective functions do not need to be weighted, and no parameters need to be set or varied to find different Pareto-optimal solutions. A single run of the algorithm can find multiple solutions on or near the Pareto-front. A system can then present the different Pareto-optimal solutions, and the customer can choose the best trade-off. One disadvantage is that EAs are not guaranteed to find actual Pareto-optimal solutions since they are probabilistic and heuristic. Nevertheless, they have shown promising results in finding multiple good solutions for the problem at hand.

Chapter 4

Related Work

The previous chapters treated the basic concepts of recommender systems (RSs) and evolutionary computing (EC). RS performance is not one-dimensional but encompasses many aspects that can be subjective and conflicting. This chapter will review recent research that has attempted to optimise the various facets of RS performance at the same time. In particular, the focus will be on methods that have used EC techniques in some form. The remainder of this chapter looks as follows. First, Section 4.1, will cast RSs as multi-objective optimisation problems (MOPs) where the metrics from Section 2.4 are the objectives to be optimised. Second, Section 4.2 briefly enumerates several approaches to solve the problem at hand. The chapter concludes with a more detailed view of research using evolutionary algorithms (EAs) to increase RS performance in Section 4.3.

4.1 Recommender Systems as a Multi-objective Optimisation Problem

As mentioned in Section 2.4.2, several criteria characterise the performance of an RS. Accuracy, diversity, novelty, serendipity and coverage all contribute to the overall user experience of the system. RSs can hence be seen as MOPs. The goal is to maximise all the metrics mentioned above. However, improving one metric must not be at the expense of another. For example, a RS may maximise accuracy as much as possible, resulting in extremely low coverage or novelty. This is not advisable for a good user experience [41]. In practice, different trade-offs are possible. For a new user, it is probably good to provide accurate recommendations since not a lot of information about their preferences is available. On the other hand, for more seasoned users, it may be better to increase the novelty or diversity in the recommendations. In doing so, the user can discover new and exciting items. Therefore, the goal is to find solutions on (or near) the Pareto-front such that the best compromise between the metrics can be selected.

4.2 Solving the Multi-objective Optimisation Problem

Several approaches have been developed to increase one or more of the 'alternative' RS performance metrics [34]. Some techniques start from the recommendations produced by a regular recommender that optimises accuracy. They then reorder the results to optimise one or more alternative metrics without losing too much accuracy. Inspired by Smyth and McClave, several researchers have used a simple *greedy* algorithm to rerank the recommendations to boost diversity [48]. The idea is to start from the original list of recommendations (which should all be somewhat relevant) and in every round, the item that is most dissimilar to the already included items is chosen. This technique has been successfully applied by Ziegler *et al.* to increase topic diversity [55]. Barraza-Urbina *et al.* introduced a parameter to better control the trade-off between diversity and accuracy [8]. Adomavicius and Kwon developed another reranking approach to find a balance between coverage and accuracy [2]. Their design introduces a parameter that switches between ranking functions that optimise one of the metrics. Items with predicted ratings above some threshold will be ranked according to a standard accuracy-based ranking function. The other items are ranked according to some alternative ranking function that optimises different criteria. By increasing the threshold towards higher predicted items, the accuracy increases but coverage decreases.

Another class of techniques model the problem at hand as a graph problem and then use graph algorithms to collectively optimise several performance metrics. Onuma *et al.* developed a system that assigns a score to items such that items that have connections to separate clusters of users receive a higher score [44]. If an item connects to several user clusters, it has a higher chance of being serendipitous. Nakatsuji *et al.* constructed a user similarity graph and performed a random walk algorithm to find connected but not too similar users. These then serve as a source for serendipitous recommendations [43]. Zhou *et al.* developed a *heat spreading* algorithm [53]. They model the users and items in a network and initialise the algorithm by assigning a 'resource' to each item. Then they redistribute this across the network using a process analogous to heat diffusion. The 'resource' is evenly spread from items to the neighbouring users, and then back from the users to the items. By doing so, the algorithm favours items with few links, thus boosting novelty. By introducing a hybridisation parameter, they can balance their approach and ProbS, another graph-based algorithm that redistributes the resources in a different way that leads to a high accuracy [54]. The advantage of their scheme is that the hybrid method has the same computational complexity as both separate algorithms.

Finally, several researchers have leveraged EC techniques to increase several alternative metrics to accuracy. This will be the focus of the next section.

4.3 Evolutionary Algorithms for Recommender Systems

Because of their metaheuristic nature, EAs can be used in many different ways. In the RS field, they have been used to tackle several of the classic problems such as the cold start problem, sparsity,... Horvàth and de Carvalho published a review on some recent papers [31]. They performed a SWOT analysis identifying multi-objective optimisation as one of the main strengths of using EC in RSs. EC has several use cases within the RS framework. However, the focus of this work is on improving the performance of RSs concerning the different metrics discussed in Section 2.4. Therefore, the rest of this section examines the multi-objective optimisation techniques using EC.

Perhaps one of the most straightforward approaches to optimising multiple criteria is combining different algorithms. Ribeiro *et al.* proposed to use a set of algorithms $A = (A_1, A_2, \ldots, A_l)$ that predict ratings for a user-item pairs [47]. Each algorithm optimises a different objective. The final rating is a weighted sum of the individual ratings, where the weights are determined using an EA. The advantage of this strategy is that it does not depend on which algorithms are aggregated, nor on the data domain. Though searching for Pareto optimal weights is expensive, the authors mention that they can be calculated in an offline manner. Moreover, the weights do not have to be recomputed often. Their results show that a good trade-off can be made between the different metrics, dependent on what a user needs.

The GUARD framework focused on generating computationally simple, memory-based ranking functions that optimise accuracy, diversity and novelty [28]. The authors of the paper use genetic programming to evolve a function that uses simple aggregations of the data such as the average rating of a user or item, bias terms estimated from a simple model,... They cache the terminals beforehand, which takes considerable time but can be done offline. The authors tested their framework on MovieLens data (100K and 1M) but only outperformed the baselines on the smaller data set. They hypothesised that the parameters of their method were not properly tuned for the bigger data set.

Wang *et al.* developed a technique where first a regular item-based CF system is used to predict a list of recommendations L followed by an EA that selects k items from this list where different trade-offs can be made between accuracy and diversity [52]. They use a multi-objective EA based on decomposition (MOEA/D) where the individuals are lists of items selected from the original list L, and recombination and mutation are one-point crossover and one-point mutation respectively.

A slightly different path taken by the same authors is named MORS [51]. They wanted to obtain a balance between recommending items that are accurate, and items that are in the long tail. In a first step, they compute a list L like before, and for each item, they determine

the unpopularity. The unpopularity is inversely proportional to the mean rating and standard deviation of the rating, meaning that items that receive low ratings are unpopular. After these initial computations, they apply MOEA/D to find a balance between the different objectives. The authors also swapped the one-point crossover operation for two-point crossover.

Similarly, Geng *et al.* attempted to improve the diversity and novelty of recommendations while maintaining the accuracy using the Nondominated Neighbour Immune Algorithm (NNIA) [26].

Related to the previous works, Zuo *et al.* proposed to use a multi-objective EA (NSGA-II) to simultaneously provide accurate, diverse and novel recommendations [56]. They use ProbS as their accuracy estimator [54].

Cui *et al.* developed an improved variant inspired by the previously discussed designs [18]. Their goal is to maintain accuracy while also providing topic diversification. The researchers reason that the previous methods can be improved by slightly altering the genetic operations. More specifically, they established a crossover operator based on typical user behaviour. This operator takes multiple parents and calculates the frequency that each item appears in the parent. The higher this frequency, the higher the probability that the user will like this item. Hence, it receives a higher probability of being included in the child solution. Experiments on the MovieLens 100K dataset shows that their approach can achieve better results, especially in terms of diversity and novelty.

Finally, Lin *et al.* proposed some further improvements to these techniques [39]. To improve convergence speed, they use an extreme point guided method. This means that solutions are computed that maximise one of the objectives (accuracy, diversity or novelty) based on the prior knowledge of the RS. These extreme points are then used for the initialisation of the population such that it converges more quickly to the Pareto-front and also to diversify the population. For their crossover operation, they select two parent solutions that each have a recommendation list for all the users in the cluster. They then find similar users in the different parent solutions and perform an adapted uniform crossover operation to ensure that no invalid recommendation lists are generated. Their algorithm, outperforms earlier techniques, while also reducing convergence time.

Chapter 5

Data

This chapter describes the data which was used for the RS and how the data from different sources were combined to utilise it for experimentation. Section 5.1 discusses the Million Song Dataset and its related collections of data. Some statistics and remarkable properties will be given. Unfortunately, there were some issues along the way that needed to be addressed. These will be mentioned in Section 5.2. To solve some of the problems, additional data was used, which will be described in Section 5.3. This section will also examine the properties of these data sources and explain how the Million Song Dataset and the new data were linked. Section 5.4 investigates the audio features in more detail. Finally, Section 5.5 summarises the features that will be used for the algorithm in Chapter 6.

5.1 Million Song Dataset

The Million Song Dataset is a set of one million contemporary[1] songs [10]. The Echo Nest provided the core dataset, which consists of the metadata and audio analysis of each song. The metadata consists of general information about the song such as a song ID for identification, a title, the release year, a track ID to refer to the music track that was used for the audio analysis, the album and some extra information of the album,... In case the release year is unknown, it is set to 0. However, this can skew the distribution of the dataset. Therefore, the release year is imputed in the following way. If the release year is missing, but the artist has other songs with known release year, the release year is set to the average release year of that artist. In case there are no release years for the artist, it is set to the overall average release year.

The data also include additional information on the artist: the artist's Echo Nest ID, name,

[1]The dataset contains no new songs since it was initially released, meaning that no songs released past 2011 are included.

location, tags, IDs from other music services such as playme[2], 7digital[3] and MusicBrainz. The Echo Nest also included the artist familiarity and artist 'hotttnesss' [38]. The familiarity indicates how well-known an artist is. The 'hotttnesss' on the other hand expresses to what extent an artist is gaining popularity. At the time of the initial release artists with the highest familiarity were Akon, Paramore, and Britney Spears. Kanye West, Daft Punk and Black Eyed Peas received the highest hotttnesss.

The audio features contain a wide range of acoustic information such as the timings of the beats and bars, the key, the mode, the average loudness of the analysed track, the sections of a song (e.g. verse, chorus), the segments, some segment-specific features such as loudness, pitch, timbre,...

Besides the core data, the Million Song Dataset also contains a collection of related datasets.

- The Second Hand Songs dataset comprises some cover songs.
- The musiXmatch dataset provides lyrics of the songs.
- The MAGD and tagtraum dataset provide genre labels.
- The last.fm dataset includes other song-level tags.

Finally, there are two datasets containing user listening data. The first one is the Taste Profile subset which comprises triplets of users, songs and the number of times a user listened to a particular song that is also present in the Million Song Dataset. The other dataset is a mapping of Echo Nest song IDs to thisismyjam IDs. Thisismyjam was a social website that allowed users to share their favourite song of the moment, and other people could like this. The service no longer exists but they made a dump of its data available for download. The creators of the Million Song Dataset matched the Echo Nest IDs to the thisismyjam IDs such that this user data can also be used, e.g. in a RS. Eventually, The Taste Profile subset seemed like the best alternative because it integrates better with the original dataset: no additional mapping phase is needed. The Taste Profile subset will be discussed next.

5.1.1 Taste Profile

The Taste Profile subset contains user listening data from 1 019 318 anonymised users on a subset of the songs present in the Million Song Dataset. In total 384 546 unique songs are present in the total dataset good for 48 373 586 entries in total. The dataset thus has an extremely large size, and that is not the only challenge. The feedback data is implicit: it is not because a user listened to a song once or twice, or even ten times that she/he rates it highly or lowly. Additionally,

[2] `http://www.playme.com/ww/web/radio/`

[3] `https://www.7digital.com/`

most items have very little different listeners. Figure 5.1 shows the number of distinct users that have listened to a song. Notice that the y-axis is logarithmic. Most songs have very few different listeners, while some have a high number of unique listeners. It turns out that 50% of the songs have 13 or fewer distinct listeners.

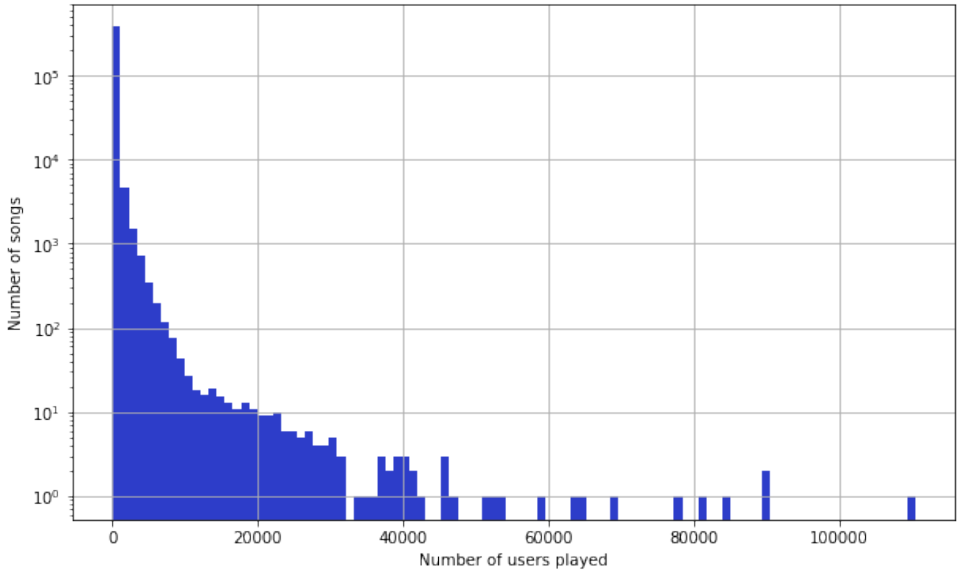

Figure 5.1: The number of unique listeners per song. Notice the logarithmic y-axis.

On the other hand, one constraint of the Taste Profile subset is that each user has listened to at least 10 different songs. Yet, 50% of the users have listened to 27 songs or less, which is not a lot to accurately model a user's preferences. Figure 5.2 shows the precise distribution. Notice the logarithmic y-axis again.

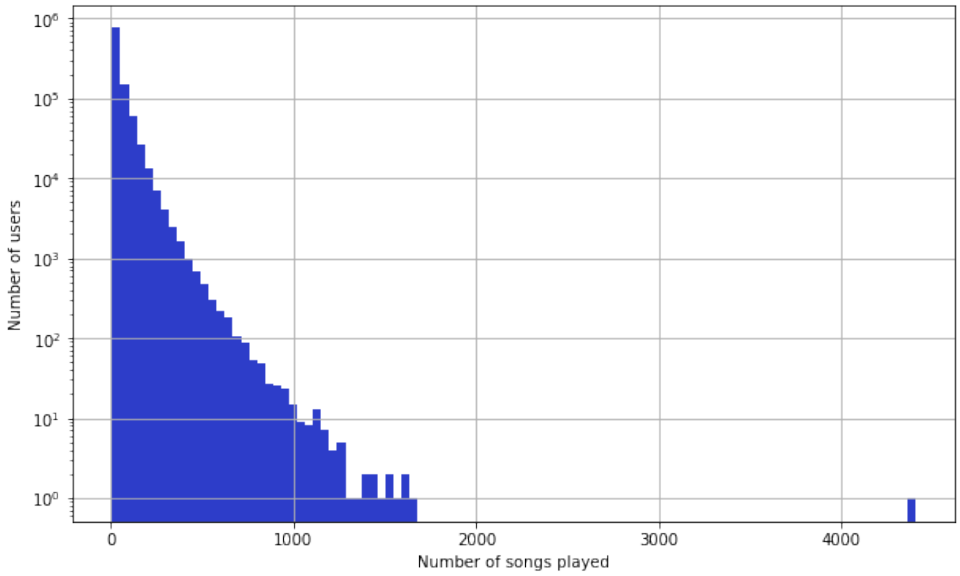

Figure 5.2: The number of unique songs played per user. The y-axis is logarithmic.

These statistics and distributions already indicate that the data is extremely sparse. In the total Taste Profile subset, only 0.012% of the user-item pairs are known. The sparsity of the data makes recommendation a difficult task: as mentioned in Chapter 2 enough examples need to be available for a model to reliably learn a user's preferences. Fortunately, the Million Song Dataset provides audio features for the songs. A hybrid RS could use those in combination with the collaborative data to improve the quality of the recommendations.

5.2 Problems

During the exploration of the dataset, some problems arose. While a part of the metadata is available for all the songs, it is only possible to download the audio features and additional metadata for 1% of the songs as a preview to test out algorithms. For the other items, a public dataset snapshot is accessible on Amazon AWS. Unfortunately, the overlap between the subset of songs and the Taste Profile Dataset was too small for practical purposes. Additionally, at the time I did not have an Amazon EC2 instance available and in the meantime, I went looking for a temporary solution. When eventually, an EC2 instance was obtained, this solution proved to be good enough for experimentation.

5.3 Auxiliary Data

The solution for the problem with the original audio features involved getting freely accessible data from other sources. Two additional sources were needed to obtain alternative audio features. As mentioned in Section 5.1, the original metadata contains IDs to link to other platforms to fetch extra data. Unfortunately, this did not help a lot in trying to find other features: the 7digital ID can be used to get an audio fragment for the songs, and the other IDs can only provide more information on the artists. In the past, the Echo Nest had an API known as *Project Rosetta*. People could send song IDs from several music services to this API and obtain the corresponding song ID from a different music service. When Spotify acquired The Echo Nest in 2014, they deprecated the API, and today it is no longer accessible. Luckily, the people at AcousticBrainz scraped the API for all the Million Song Dataset songs resulting in a dump of JSON files to convert Echo Nest song IDs into IDs of other services [42]. Using this data dump, the Echo Nest song IDs can be mapped to MusicBrainz IDs.

Subsequently, the MusicBrainz IDs can be used to retrieve audio features for the songs. The AcousticBrainz database contains audio features for its songs, and for this database too, a dump of JSON files is available [20]. Two types of features are possible: low-level and high-level. The former consists of all kinds of technical acoustic descriptors for the songs related to rhythm,

tone, loudness, spectral properties,... The latter results from processing the low-level features with several classifiers and algorithms to obtain values that are much easier for humans to interpret such as moods, danceability, genres, whether the song is instrumental, ... This project opted for the high-level features for two reasons. First, the size of the high-level features is much smaller because the small number of meaningful features compress the information significantly. Second, the high-level features take away the burden of feature engineering: if only the low-level features were available, several meaningful feature transformations would have to be designed to be usable in a final recommender model. However, using high-level features could also have some drawbacks. It takes away a part of the flexibility of having raw acoustic information. Moreover, the classifiers are not perfect. They can make mistakes. For example, classifying a song as a rock song while in reality, it is a jazz song. However, from the confusion matrices that are provided with the models, these seem to be a minority and the benefits outweigh the drawbacks. A more in-depth examination of the features present in the high-level set will follow in the Section 5.4.

5.3.1 Data Loss

Sadly, the matching of Echo Nest song IDs to MusicBrainz IDs and then finding the song IDs is not perfect. During each of the previously described "linking" steps, a considerable number of mappings are not available. From the 384 546 songs originally present in the Taste Profile Subset, only for 222 779 unique songs, a matching MusicBrainz ID was found. Moreover, finding a MusicBrainz ID does not necessarily mean that there are audio features for it. From the songs remaining after the first matching step, audio features are only found for 70 214 songs. The number of users that have listened to one of these songs also drastically decreases to 395 811. This reduces the number of entries in the Taste Profile subset from 48 373 586 to 14 099 853 user-item-play count triplets. However, this is still more than sufficient since the final experiments only use a subset of the available data.

5.4 Audio Classifiers

The high-level data contains a range of semantically meaningful music features that are computed by several default models in AcousticBrainz [1]. In total there are no less than 18 different classifiers that attribute different properties to each of the songs. The properties are usually easy to interpret for humans like genres, moods, whether the song is instrumental,... There are both binary and multi-class classifiers. The remainder of this section enumerates the different classifiers and classes used to construct a feature vector.

5.4.1 Genres

AcousticBrainz contains multiple genre classifiers. Some of these are general and assign a label like rock, metal, hip-hop, reggae,... However, some classifiers identify subgenres within a music genre. For example, one classifier determines the subgenre for electronic music, such as ambient, drum and bass, house, techno and trance. Another classifier identifies the subgenres of ballroom music based on dance styles. These "specific" classifiers are run for all songs, even when the base genre is not electronic or ballroom.

5.4.2 Mood

A song can put people in a certain mood. It can sound happy or sad, aggressive or relaxed, or it can put people in a party mood. AcousticBrainz utilises several, mostly binary classifiers to determine these moods. There is one multi-class classifier that divides songs into one of five classes [32]. Researchers found that the mood of a song is hard to determine because there is no general standard to describe it. Therefore they recommended a cluster-based approach. Appendix A gives precise descriptions of the classes.

5.4.3 Audio

There are some classifiers that identify the sound of a song. The classifiers all use the underlying low-level audio features, but these models distill the information into something interpretable for humans. The danceability of a song determines whether a song is suitable to dance on or not. There is also a classifier to identify if a song (or the vocals) are sung by a male or female voice. Additionally, A song can contain mostly vocals or be instrumental. it can also sound dark or bright (the timbre). Finally, the tonality can also be determined. In most cases, a song has a tonal centre around which it is built. Some songs or compositions do not have this, in which case they are atonal.

5.5 Item Features

The final set of features the algorithm will work with consists of two parts. The feature vector includes the audio features from the previous section. For binary classifiers, only one class remains in the final feature vector. For example, if a classifier determines whether a song is "sad" or not, only the "sad" class probability is added to the feature vector. The "not sad" class is not included since the cross-correlation with the "sad" class is -1 anyway. For multi-class classifiers, all class probabilities are added to the feature vector.

Besides the audio features, some of the metadata can also be informative. The metadata is

provided by the Million Song Dataset and includes the duration of the song, the release year, the artist familiarity and the artist hotttnesss.

Chapter 6

Design of the Algorithm

This chapter gives a broad overview of the recommender algorithm enhanced with genetic programming (GP). First, Section 6.1 explains the motivation and goals of the algorithm. Then, Section 6.2 sketches the proposed approach at a high level. Sections 6.3 to 6.5 zoom in on certain parts to provide a more detailed explanation where necessary.

6.1 Motivation & Design Goals

Traditional RSs, such as the ones summarised in Chapter 2 tend to concentrate on maximising the accuracy of recommendations, which is not advisable to achieve the best possible user experience [41]. Section 4.3 reviewed several techniques that leverage the power of EC to solve RSs as a MOP.

Specifically, the algorithm described in this chapter draws inspiration from the general scheme put forward by Wang *et al.* where a regular RS first generates a list of recommendations L from which eventually k items are selected [52]. The main benefit from their scheme is the fact that the first step in the algorithm consists of just a regular RS, which implies that their strategy is easy to adopt in an existing environment to improve the user experience. The principal drawback of their method, and other related methods outlined in Section 4.3 is the fact that the EA has to be rerun every time the RS has to generate a new set of recommendations. Since one of the main disadvantages of EC techniques is their time complexity, this is detrimental to the performance of the entire system. These considerations lead to the following design goals:

- The algorithm design should be such that it can easily be adopted in an existing environment where a regular RS is already running.
- The algorithm should attempt to optimise the different RS metrics at the same time, making sure not to prioritise one metric over the others.

- The algorithm should not rerun the EC steps every time to provide new recommendations.

The system proposed by Wang *et al.* achieves the first and second goals, but not the third. However, by replacing evolutionary programming with GP, where the individuals represent functions or computer programs, it is possible to realise all three design goals. This is because the GP step is used to find a scoring function that transforms the original ranking of items into a different ranking that achieves a better performance on all metrics. Once the algorithm has produced a suitable scoring function, it can be reused on a new set of recommendations. That way, the GP step only needs to run sporadically, much less frequently than the regular RS.

6.2 High-level Overview

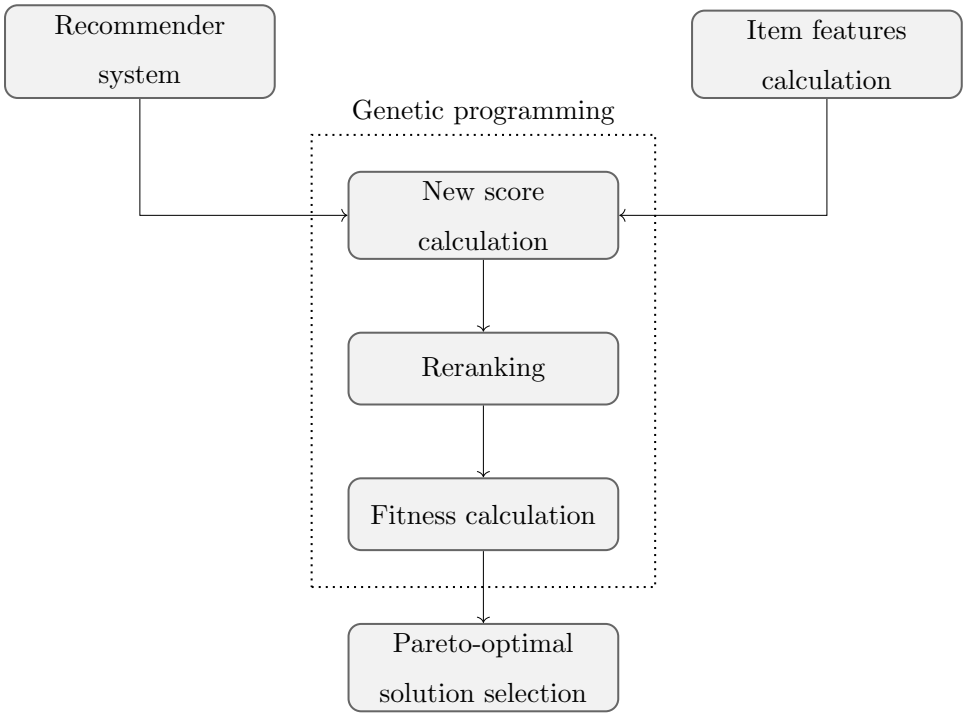

Figure 6.1: The general flow of the proposed recommender algorithm.

Figure 6.1 shows a birds-eye view of the proposed algorithm. In the first step, a RS is trained and used to generate several recommendations for each user in the system. This RS focuses exclusively on accuracy and thus, the recommendations should be mostly relevant for the users. Additionally, the system calculates some features for each of the items in the catalogue. These can be actual item features such as the ones discussed in Section 5.4. They could also be transformed versions of the original features such as principal components. Another possibility could be to derive the features purely from the collaborative data if no domain-specific information is present (e.g. bias terms, self-information etc.). Note that it is possible to calculate the item features beforehand since it does not depend on any results of the RS or any other systems.

Following these initial steps, the GP step starts. First, a function takes the original recommender score and item features as input to calculate a new score for each item in the list. Next, the items are reranked based on the new score. Finally, this new ranking is evaluated using the typical RS performance metrics to determine whether the scoring function provides a good balance between these metrics. This entire process is repeated for all functions within the generation, after which a new generation is produced using the genetic operators discussed in Chapter 3.

After the GP step has processed the last generation of individuals, the algorithm terminates. During the process, the GP step keeps track of all non-dominated solutions it encountered. After termination, a user can ultimately choose between the obtained Pareto-optimal solutions. That way, different users can find a balance between the different criteria according to which ones they value most.

6.3 Recommender system

The first step of the algorithm comprises a regular accuracy-focused RS. It takes the user listening data as input and comes up with a list of items for each user. In this stage, the recommendations should be as relevant as possible. In principle, any algorithm can be used: neighbourhood-based CF, matrix factorisation, deep learning models. One may also utilise a hybrid recommender system that makes use of domain-specific information. This work adopted the BPR model[1] from Section 2.3 as the RS. It fits best with the dataset because the researchers explicitly designed it for implicit feedback data. Moreover, it performs well and takes little training time.

6.4 Item Features

As mentioned in Section 6.2, the item features can be (transformations of) domain-specific features, but they can also be derived from purely collaborative data. Even though domain-specific features are available, it also seems reasonable to take into account collaborative information. For example, using a simple bias model, bias terms can be calculated to capture whether people tend to listen to certain songs more than others. Additionally, the number of unique users that consumed an item, or the self-information which is a function of that number, could also prove useful when scoring items. For instance, a scoring function could assign a higher score to items with a higher self-information to increase the novelty of the recommendations.

This is valuable information that can help the GP step to discover better solutions. Therefore,

[1]The implementation used in this work comes from the Implicit library (`https://github.com/benfred/implicit`) via an API bridge in the LKPY project [23]

this project uses both the pre-existing features as well as the features based on collaborative data.

6.5 Genetic Programming

6.5.1 Process

The GP process involves many design choices related to representation, initialisation, operators and selection. Before diving into how the recommendations tie in with the GP part, the most important design decisions related to GP will briefly be listed.

The representation that is used here is a traditional (syntax) tree. An inner node represents an operation like the addition or multiplication, while the leaves may contain constants or input variables. The tree can be "compiled" into an executable expression that takes the variables as input and produces a score as output. The included operators and constants are shown in Table 6.1. The safe division is like a regular division, the only difference being that when a division by zero occurs, it returns 0. The tree may also contain a uniformly generated constant between -10 and 10. The i in the symbol c_i indicates that if multiple constants occur within one tree, these constants may have different values. An example expression could be:

$$s' = c_0 \times s + \sin\left(c_1 + (x + y)\right), \tag{6.1}$$

where x and y are input variables to the function. s is a somewhat special input variable and represents the score of the item in the original recommendation list. The tree representation of this expression is given in Figure 6.2.

Name	**Symbol**
Addition	$+$
Subtraction	$-$
Multiplication	$\times$
Safe division	$/$
Cosine	cos
Sine	sin
Hyperbolic tangent	tanh
Uniform random constant $\in [-10, 10]$	c_i

Table 6.1: The allowed operators and constants in the tree representation.

To initialise the first generation, the classic ramped half-and-half schema is used [37]. Essentially, when generating random trees in the beginning, a tree is allowed to have a depth between

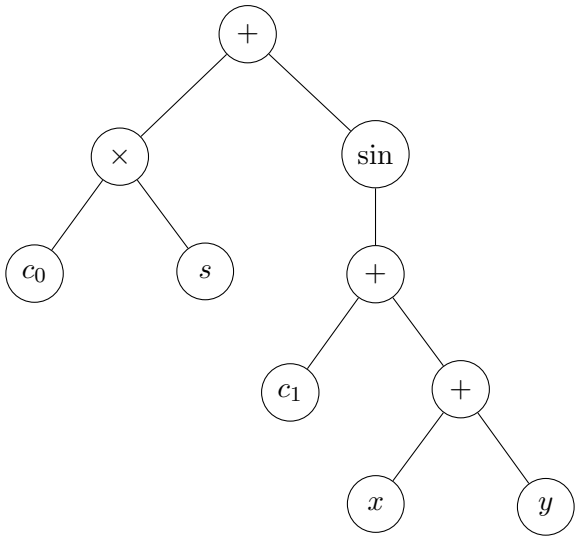

Figure 6.2: An example tree representation for Equation (6.1).

a minimum and a maximum. The ramped half-and-half schema divides the initial population such that an equal part of the trees has a particular depth. Moreover, it splits the tree generation into two strategies. Half of the trees within a class is grown fully, meaning that all leaves have the same depth, while the other half is allowed to have leaves at different depths. By using this initialisation scheme, the hope is that the initial population will be as diverse as possible to explore a larger part of the search space.

After the initialisation, the individuals are evaluated. This will be discussed more in detail shortly. Since there are multiple objectives to be optimised, the selection operator needs to take that into account. This work uses the NSGA-II algorithm from the DEAP library because it has been shown to find non-dominated solutions as close as possible to the Pareto-front [19]. Moreover, the algorithm explicitly maintains diversity in the parents during the selection procedure. In doing so, the individuals of the next generation explore a larger part of the search space so that they can discover more distinct Pareto-optimal points.

Using the multi-objective selection operator the parents for the next generation are chosen and they create offspring using genetic operators. The crossover operation is a simple one-point crossover where two parents exchange subtrees to create new individuals. The mutation operator selects a random node in the parent and replaces the subtree at this node with a random fully grown tree (of limited depth). There is one common issue that often occurs in GP known as *bloat*. This is the phenomenon that the average size of the individuals keeps getting longer unless appropriate action is taken. Over the years, many researchers have made attempts to control bloat in the trees. One of the most intuitive ways, already proposed by Koza is to just limit the height of the trees [37]. In other words, if a crossover or mutation operation produces a child that exceeds the height limit, this child will not be valid. In that case, a copy of the

parent will take its place in the next generation. This work opts for this simple but effective approach. The proposed genetic operators produce a new set of individuals from the parents and the cycle can start over. The process is repeated for a fixed number of generations. The precise parameter values will be listed in Chapter 7. All functionality related GP was implemented using the *Distributed Evolutionary Algorithms in Python* package also known as DEAP[2]. The library contains tools for designing and implementing many variations of the general EA.

6.5.2 Scoring Function

The goal of the GP step is to find a function that, given as input a list of recommendations and features of the items, computes a score for each item in the list. This is exactly what happens in Equation (6.1): a function of some input variables (including the original score) produces a new score s' and the items are reranked based on s' After reranking, the new recommendation lists are evaluated for the RS metrics in the hope that the function achieves a good balance between all of them. These metrics are the fitness values in the evolutionary process.

In practice, the implementation does not compute the score separately for each item. Instead, a single data frame contains the recommendation lists of all users. The program merges the item features to the appropriate items in the lists. That way, the scoring function can operate on Pandas[3] columns using the broadcasting mechanism for speed. This results in a much more efficient implementation than iterating over all users and items separately. The RS performance metrics also utilise the broadcasting mechanism to improve computation speed.

6.5.3 Pareto-optimal Solution Selection

The GP process runs for a fixed number of generations. During the process, the program keeps track of non-dominated solutions in a separate data structure. Throughout the generations, if the algorithm discovers a new non-dominated solution, it is added to the data structure. If the new solution dominates any of the existing non-dominated solutions, they are removed. As discussed in Section 3.5, EC techniques can find multiple points close to the Pareto-optimal solutions in a single run. Therefore, when the algorithm terminates, there are several scoring functions to choose from.

There are several possibilities to select a solution. For example, it is possible to select the solution with the highest coverage subject to the condition that the accuracy should be at least 60% of the original. Another approach could be to let the users interactively decide which criteria are more important for them or use an algorithm to determine it for them. That means that different users could have different reranking functions tailored to their needs.

[2] `https://github.com/deap/deap`

[3] https://pandas.pydata.org/

Chapter 7

Experiments

This chapter discusses several implementation details related to the algorithm presented in Chapter 6. Additionally, some experiments were carried out to investigate whether the proposed approach of the previous chapter achieves better performance in terms of the alternative RS metrics. The rest of the chapter has the following structure. First, Section 7.1 describes the dataset on which the experiments were conducted. Second, Section 7.2 details the choices for the parameters of the algorithm. Section 7.3 clarifies the quantitative performance criteria used for the experiments. Next, Section 7.4 presents the setup of the experiments. Then, Section 7.5 enumerates the results from the different experiments with the necessary visual material. The chapter concludes with a deeper discussion and interpretation of the results the algorithm found in Section 7.6.

7.1 Data

Chapter 5 already went into great detail to describe how the data was obtained and combined from various sources. To recapitulate, after the data from all sources were combined, the dataset size abated to about 14 million user-song entries instead of 48 million. Due to the large sparsity, this subset still contains almost 400 000 users and more than 70 000 songs. Training and using the system on a dataset of this size would require substantial time.

To keep the time manageable for experimentation, the final dataset is a subset containing 1.4 million entries. The number of entries in this dataset gives a distorted picture because it still comprises about 48 000 users and 34 000 songs. This is vast compared to the more popular Movielens 1M dataset[1] of an analogous size which contains ratings from only 6 000 users on 4 000 movies.

[1] `https://grouplens.org/datasets/movielens/1m/`

7.2 Algorithm Parameters

The algorithm outlined in Section 6.2 is configurable through several parameters. Some of these parameters pertain to a single step of the algorithm such as the RS or the GP parts. Other parameters regulate the interaction between these steps.

As the system is structured to be applicable in an environment where a regular RS is already in place, the parameters of the base RS need to be properly tuned. In a realistic scenario, these would typically be tuned to maximise the accuracy of the system. The BPR algorithm incorporates many parameters which typically provide a trade-off between better accuracy and shorter training times. For most of these, the defaults proved to be a good starting point. The only parameter that was changed is the number of latent factors of the underlying matrix factorisation model. This was set to 80 instead of 100. Table 7.1 shows the RS parameters and their corresponding final values.

Parameter	**Value**
# Latent factors	80
Regularisation	0.01
# training epochs	100
Learning rate	0.01

Table 7.1: The RS parameters and values.

For the GP step, there are also numerous parameters to be set. Since this step takes a long time to run, it was not possible to perform a full grid search over all parameter values. In literature, choosing the values typically relies on conventions or ad hoc choices [22]. Additionally, there is some limited form of experimentation with different parameters and a limited range of values. However, this brings about problems of its own because the parameters affect each other. It is not possible to optimise the parameters one by one: they must be collectively optimised. Finally, there is no guarantee that the optimal parameter values will be among the tested combinations. There is a compromise between the granularity of the search grid and speed.

Since the focus of this work is not on solving the tuning problem of EA, the selected parameter values are selected following the conventions. This work experimented with two GP parameters, namely the population size and the number of generations. All of the other parameter choices were made in an ad hoc fashion, corresponding to the conventions. They are shown in Table 7.2.

Finally, there are two system-wide parameters. These are the size of the original recommendation list L generated by the regular RS and the size of the output list k. The former will

Parameter	Value
Crossover probability	0.9
Mutation probability	0.05
Tree initialisation minimum height	1
Tree initialisation maximum height	3
Maximum tree height	17
Mutation tree minimum height	0
Mutation tree maximum height	2

Table 7.2: The parameter choices for the GP step.

significantly influence the final performance. The larger the original list, the more exploration opportunities for the GP step. On the other hand, a larger list L also implies a larger computation time for the scoring function and a larger memory footprint. The output list size k depends more on the environment. Sometimes, the user would need a large list of recommendations to choose from, while in other situations a few options suffice. Table 7.3 displays the system-wide parameter values.

Parameter	Value
Original list size L	50
Final list size k	10

Table 7.3: The system-wide parameter values.

7.3 Performance Criteria

Before showing and discussing the results of the experiments, it is necessary to know which quantitative criteria the algorithm should optimise. Section 6.2 mentioned that the typical RS performance metrics should be optimised. These comprise the accuracy, novelty, coverage, diversity and serendipity.

As discussed in Section 2.4 the last one does not have a practical quantitative formula, because it would require explicit user feedback on whether a recommendation is surprising and relevant. The remaining metrics do have quantitative formulae which can be utilised to quantify the performance before, during and after the GP step.

For the accuracy, this work prefers the normalised discounted cumulative gain (NDCG) over the root mean squared error (RMSE) to reflect the fact that the ranking of the items is more important than the predicted rating. Moreover, the feedback is implicit so the 'ratings' do not

express the degree of preference, making the RMSE even less relevant. The novelty is expressed by the average self-information in the recommendation list as shown in Equation (2.9). The coverage is simply calculated as the fraction of items that appear in at least one recommendation list such as in Equation (2.11).

Finally Equation (2.7) provides a formula to calculate the diversity of the recommended items as the intra-list distance between the items. However, there is a slight problem in that this formula can take quite a while to compute for a large number of users. This is especially the case because pandas' groupby-apply mechanism cannot utilise the broadcasting mechanism. Considering the diversity needs to be computed for every individual during the GP step, an alternative approach to quantify this metric is proposed. Every item in a user's recommendation list has several features. Intuitively, the diversity of the recommendation list will be larger if the standard deviation on each of these features increases. Therefore, a few important features are chosen and in every step, their average standard deviation in the recommendation list is calculated as a measure for the diversity. Let $i_j = (i_{j,1}, i_{j,2}, \ldots, i_{j,k})$ be an item represented by its features. The alternative formula to calculate diversity is then:

$$\text{div}(L) = \frac{1}{k} \sum_{n=1}^{k} \sqrt{\frac{1}{L} \sum_{j=1}^{L} (i_{j,n} - \mu_n)^2}, \tag{7.1}$$

Here, μ_n is the average value of the n-th feature. The formula computes the mean standard deviation on the item features. This is a lot faster to compute and therefore, this formula is used instead of Equation (2.7). The diversity is not calculated with all item features, but only with those that have an intuitive meaning for humans. The experiments calculate diversity based on the Dortmund genre classifier and the binary danceability, happy, sad, timbre and tonal classifiers.

7.4 Experimental Setup

The experimental setup is as follows. The RS is tuned beforehand using threefold cross-validation and its parameters remain fixed throughout the rest of the process. Next, the data is split into three parts: one training part, one part to simulate future interactions, and one part for testing. The split is made per user. The first two parts represent 90% of the data while the third part represents 10%. Within this 90%, the training part represents 90% while the future interactions part is good for 10%. There is one iteration per set of parameters. In each iteration, the regular RS is trained on the training data and produces a set of recommendations. Then, this set of recommendations and the item features serve as input to the GP step. This step runs for a fixed number of generations to find a set of Pareto-optimal scoring functions. To ensure that the

functions attain the goal of optimising alternative RS metrics when more data is available, the regular RS is retrained. However, this time the training data also includes the part of the data to simulate future interactions. The scoring functions found during the GP step are then used to rerank the new recommendation lists, and the performance is recorded and compared to the original performance. That way, the experiments check what happens after new data becomes available. This relates to a realistic scenario where after some time, users listened to some new songs and need a new recommendation list based on their updated profiles. The functions should still obtain a reasonably high score for the different metrics.

7.5 Results

The current section presents the results of the experiments. First, Section 7.5.1 discusses the baseline performance. The experimental environment and tested parameter settings are given in Section 7.5.2. Section 7.5.3 considers the influence of more data on the performance of the algorithm. Next, Section 7.5.4 discusses the potential problem of bloat during the GP step. Finally, Section 7.5.5 examines the solutions found in this step and their results on the performance metrics.

7.5.1 Baseline

Before diving into the results, the baseline performance needs to be determined. The baseline system is the original RS without the GP step applied after it. Note that the hyperparameters of this RS were tuned using a separate strategy, as discussed in Section 7.4. The original performance is shown in Table 7.4. This table does *not* indicate generalisation performance of the RS. Instead it shows what the performance of the RS is without the GP step. The name-column contains the names these RSs will get throughout this chapter. RSTrain refers to the RS trained with just the train set and RSFull was trained with both the train and validation sets.

Performance-wise, the NDCG and coverage increase significantly when more data is available. This is because the system has more knowledge to model the users' preferences accurately. The novelty and diversity decrease slightly. Note that the baseline to compare against is the performance for RSFull.

7.5.2 Experimental Settings

The experiments varied the population size and the number of generations of the GP step. The larger both of these are, the longer the GP step will take because there are more individuals to evaluate. Table 7.5 shows the different settings that were tested during experimentation with

Dataset	Name	NDCG	Coverage	Novelty	Diversity
Train	RSTrain	0.073833	0.273876	8.965489	0.807235
Train + Simulate	RSFull	0.092631	0.335119	8.838002	0.802768

Table 7.4: The original RS performance.

their total execution time. The experiments used an Amazon EC2 t3.large instance with 2 AMD EPYC 7571 vCPUs and 8GiB of memory.

Population size	# generations	Time
50	50	6h 24m
50	100	12h 30m
100	50	12h 8m
100	100	23h 36m
200	50	23h 20m
200	100	46h 11m
300	50	34h 40m
300	100	67h 55m

Table 7.5: The GP settings that were varied during the experiments with their time to execute.

7.5.3 Influence of More Data

As stated in Section 7.4, the experiments measure the influence of more user data to simulate the scenario where users listen to new songs and thus need new recommendations after some time. This influence is studied because one of the goals of the design is to run the GP step only sporadically. To measure the influence, the results of the found functions on recommendations from both RSTrain and RSFull were compared. Table 7.6 shows the average performance differences on all metrics during the training stage vs. when evaluated on the test set. Like for the baseline, the NDCG and coverage of the system increase when evaluated on the test set while the novelty and diversity decrease slightly. However, the differences are somewhat less outspoken. This implies that the found functions remain stable when more user data is available. When the functions are used to rerank unseen lists of recommendations, they keep performing well.

7.5.4 Bloat

Besides overfitting, GP can also suffer from bloat. During evolution, the code keeps track of several statistics of the population. One of these is the height of each individual. Figure 7.1

Avg. difference RSFull vs. RSTrain					
Pop. size	**# gens.**	**NDCG**	**Coverage**	**Novelty**	**Diversity**
50	50	0.008249	0.039432	-0.111176	-0.004463
50	100	0.010288	0.048405	-0.103987	-0.004774
100	50	0.009216	0.045136	-0.108187	-0.004985
100	100	0.009690	0.048690	-0.104535	-0.004520
200	50	0.010369	0.046150	-0.113284	-0.004233
200	100	0.011365	0.051425	-0.106182	-0.005099
300	50	0.011322	0.051318	-0.107435	-0.005246
300	100	0.011086	0.051318	-0.101779	-0.004936

Table 7.6: The average performance differences on reranked RSFull and RSTrain recommendations.

shows the minimum, average and maximum height of the individuals per generation. For each set of GP parameters, there is a clear upward trend in the average and maximum heights. However, the increase in average height is rather small and thus bloat is not a problem in this case.

7.5.5 Pareto-optimal Solutions

All of the Pareto-optimal solutions perform at least as good on one of the four criteria from Section 7.3. During the final solution selection step of the algorithm, the system can serve several purposes. Perhaps, the owner of a webshop cares most about NDCG and coverage while novelty and diversity are secondary goals. An end-user of a music streaming service, on the other hand, would care more about diversity and novelty than about coverage and NDCG. Some users may want the best of all worlds. These scenarios require looking at a different number of dimensions. Therefore, the present discussion will start by investigating the found solutions in two dimensions. Next, it will add a dimension to optimise simultaneously. Finally, all four metrics will be looked at together.

Two Dimensions

The GP step finds many solutions on or near the Pareto-front. These solutions are then used to rerank the recommendations from RSFull. Because the system attempts to optimise four metrics, the performance cannot be shown simultaneously for all of them. Figures 7.2 to 7.7 present all pairs of these metrics as two-dimensional scatterplots.

There are a few noticeable trends in the figures. When considering just NDCG and coverage

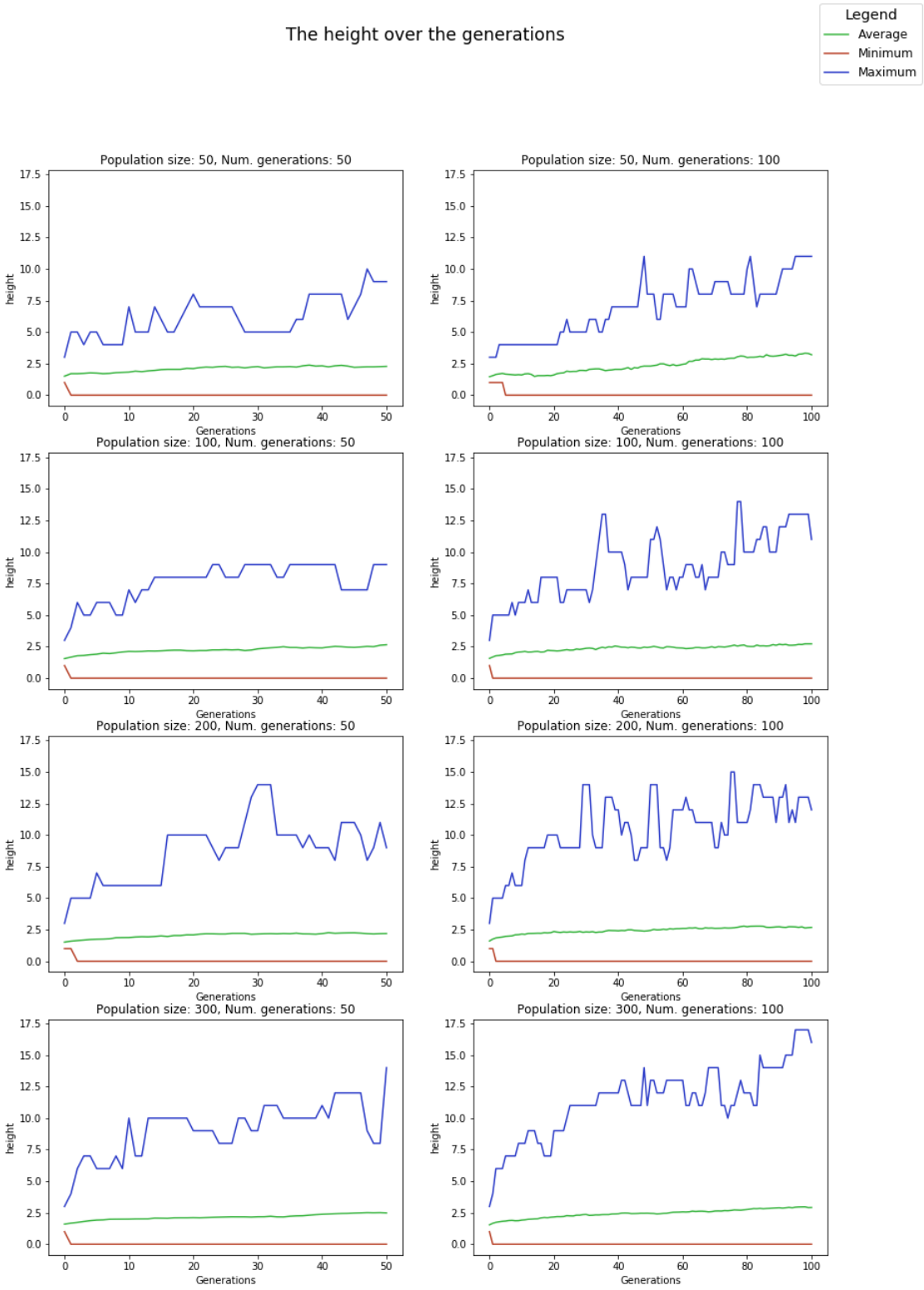

Figure 7.1: The variation of height throughout the generations.

in Figure 7.2, it is possible to select a solution that achieves a significantly higher coverage with an acceptable accuracy loss. Particularly for more generations and more individuals per generation, this is the case. Sadly, many solutions also perform worse on both of the metrics. Note that this does not mean that they do not belong to the Pareto-front. Their novelty and/or diversity can still be better than other solutions.

Figure 7.3 shows an evident negative correlation between the NDCG and novelty. This is to be expected because a higher novelty of an item means that fewer people have listened to it. Therefore the chance that a user listened to that song, and thus that it would show up in the test set, is smaller. This ultimately results in a smaller accuracy. Again, for higher values of the parameters, the algorithm finds slightly better trade-offs.

Figures 7.4, 7.6 and 7.7 indicate that the diversity remains relatively constant as a function of the other metrics. This could be a consequence of the chosen features to calculate the diversity. However, for some parameter settings, the GP step finds solutions where the diversity decreases drastically. Additionally, this seems to happen only for a relatively small range of values of the other metrics. Even upon closer inspection, it is not immediately clear why this happens. One hypothesis is that the underlying reranking functions rely too much on only a few features. Section 7.6 will investigate this in more detail.

It is also possible to find a good compromise between novelty and coverage as Figure 7.5 shows. The largest part of the found solutions achieves a slightly higher novelty than the original solution but a much smaller coverage. Be that as it may, some solutions reach better scores for both metrics. Again, higher parameter values seem to find better trade-offs.

Three Dimensions

The two-dimensional figures give some information about how two metrics correlate but they cannot give a total picture. A point with a high NDCG and novelty could score very low for coverage. Given the fact that the diversity is relatively constant Figures 7.8 to 7.15 provide a three-dimensional view of the NDCG, novelty and coverage of the Pareto-optimal solutions for the different parameters in Table 7.5. The figures all have some extreme points. With a few exceptions, the original solution is the one with the highest NDCG value. On the opposite end, there are a few solutions that achieve a high novelty and reasonable coverage, but at the cost of the accuracy. Likewise, it is also possible to choose a solution that optimises coverage and achieves acceptable novelty. Unfortunately there too, this results in a loss of accuracy. When trying to find solutions that do not disregard accuracy completely, some problems arise. The figures show that without losing too much accuracy, there are quite a lot of solutions that achieve a high novelty. However, this comes at the cost of coverage. Especially for the smaller

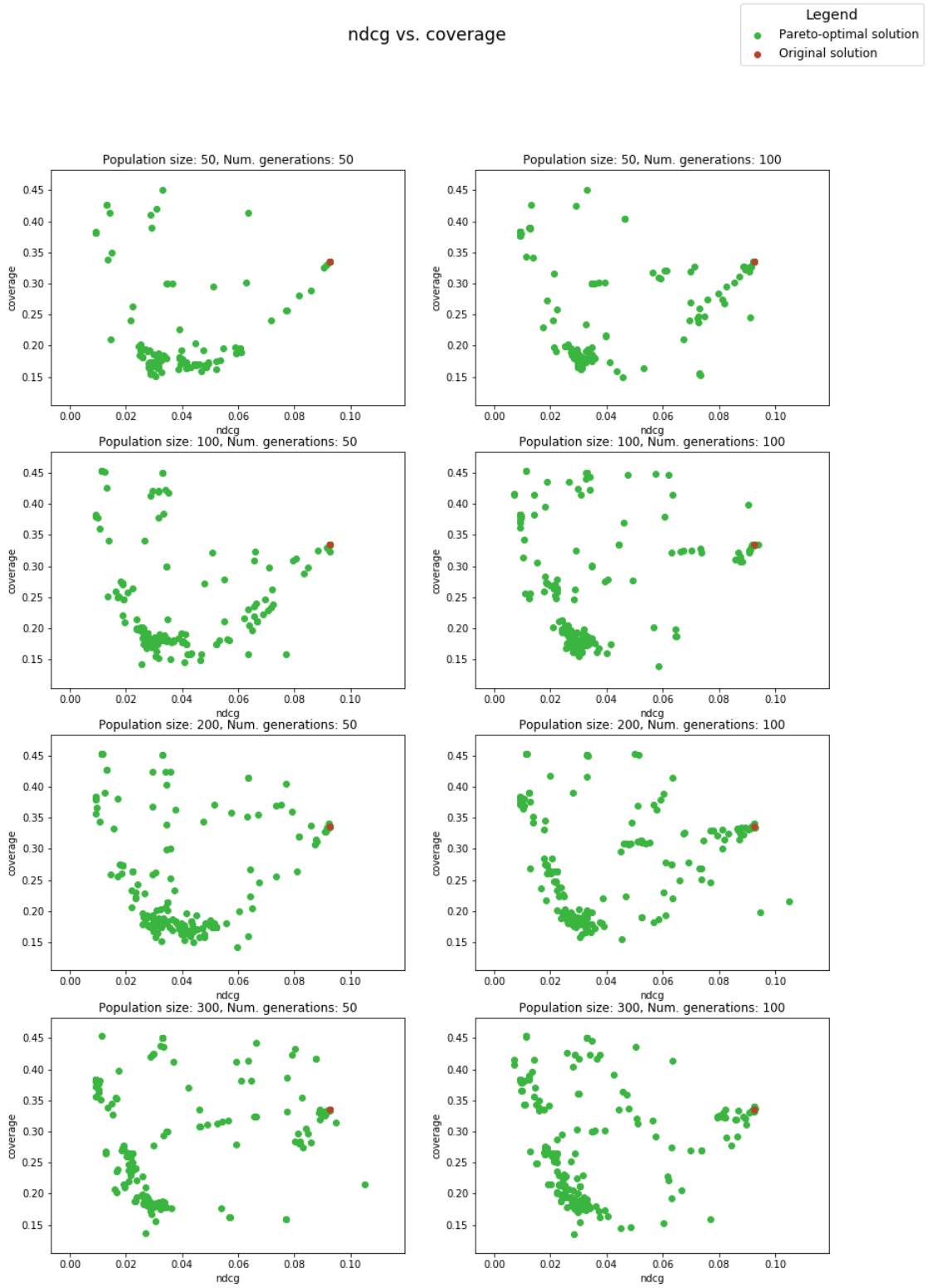

Figure 7.2: NDCG vs. coverage.

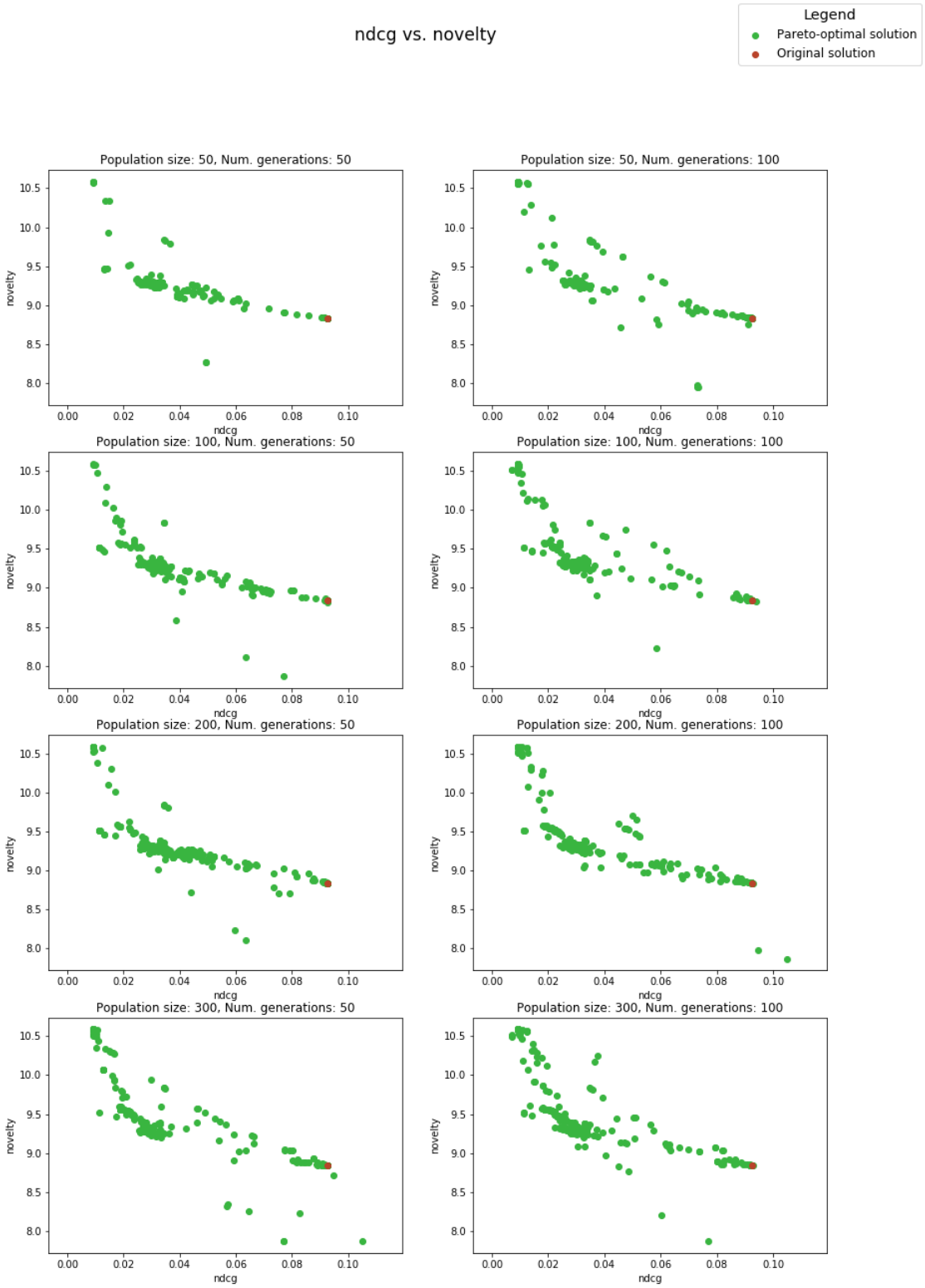

Figure 7.3: NDCG vs. novelty.

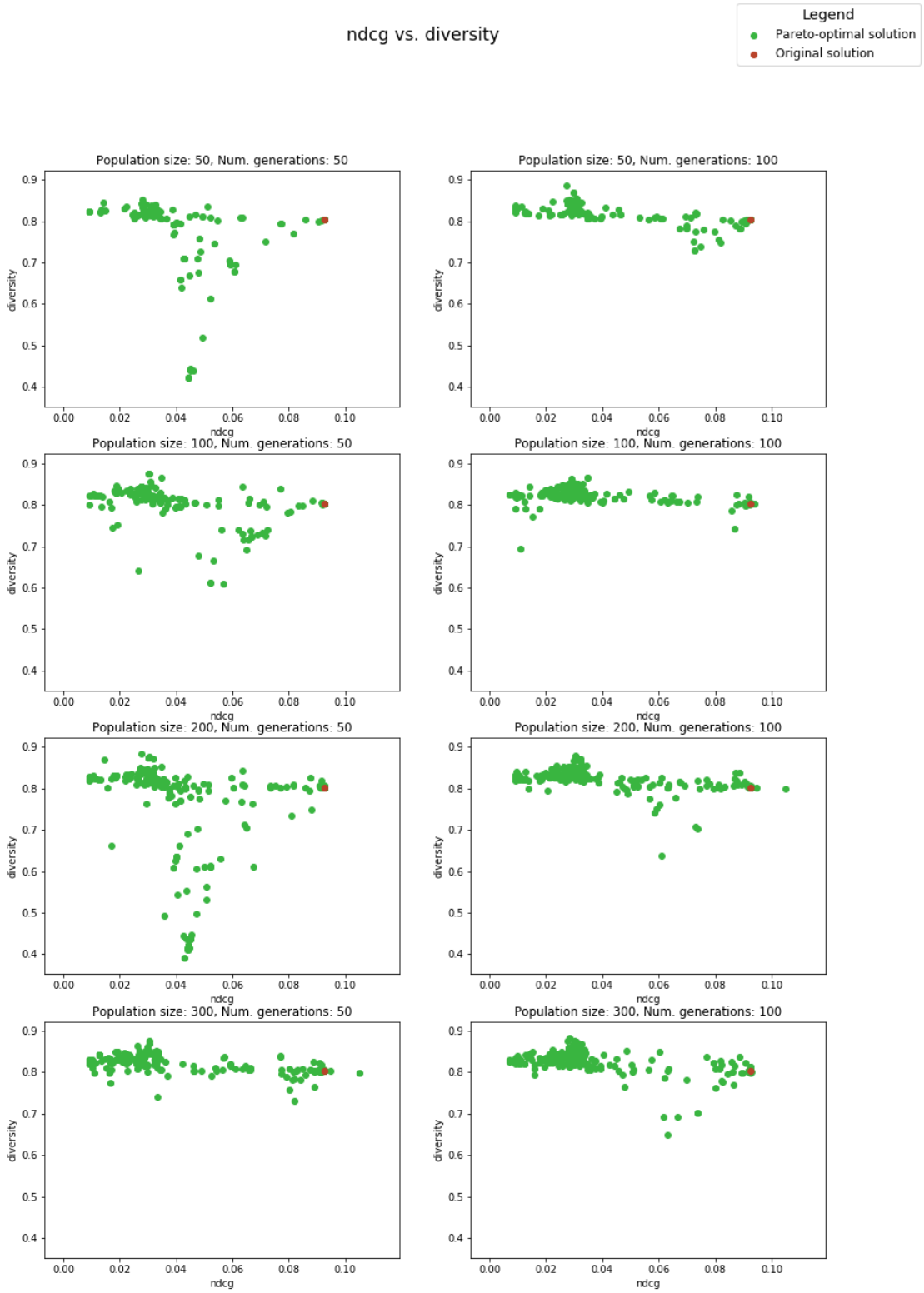

Figure 7.4: NDCG vs. diversity.

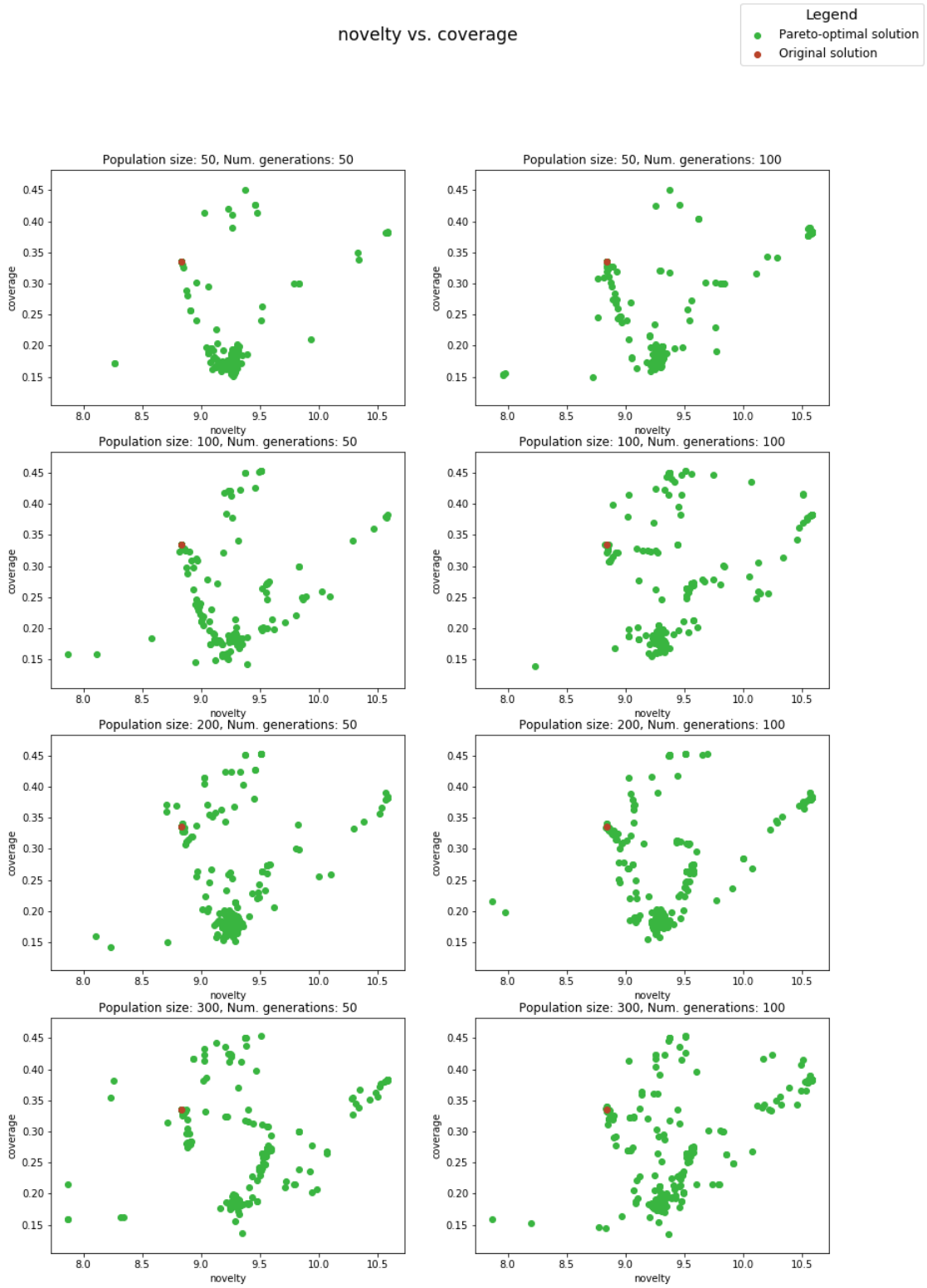

Figure 7.5: Novelty vs. coverage.

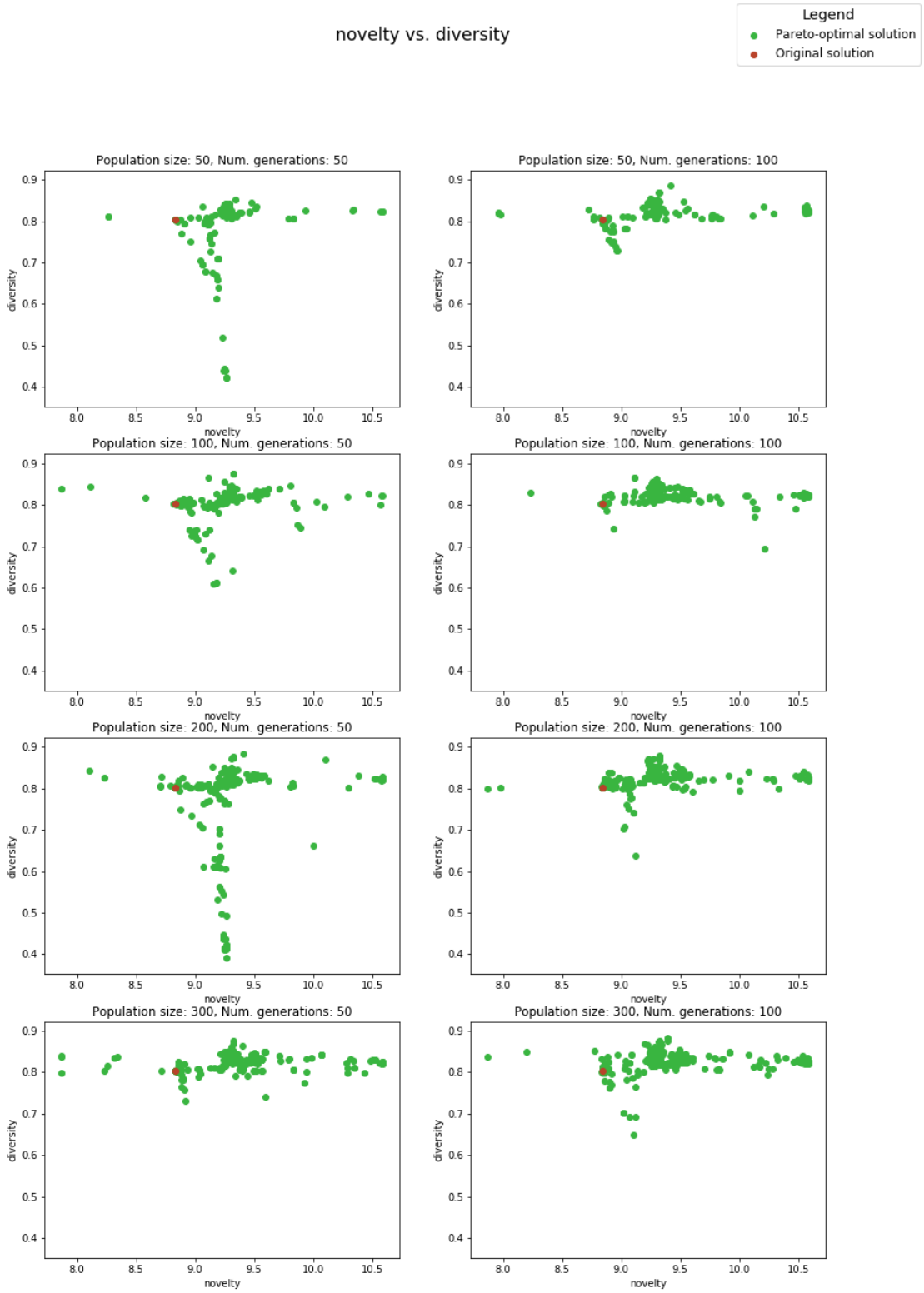

Figure 7.6: Novelty vs. diversity.

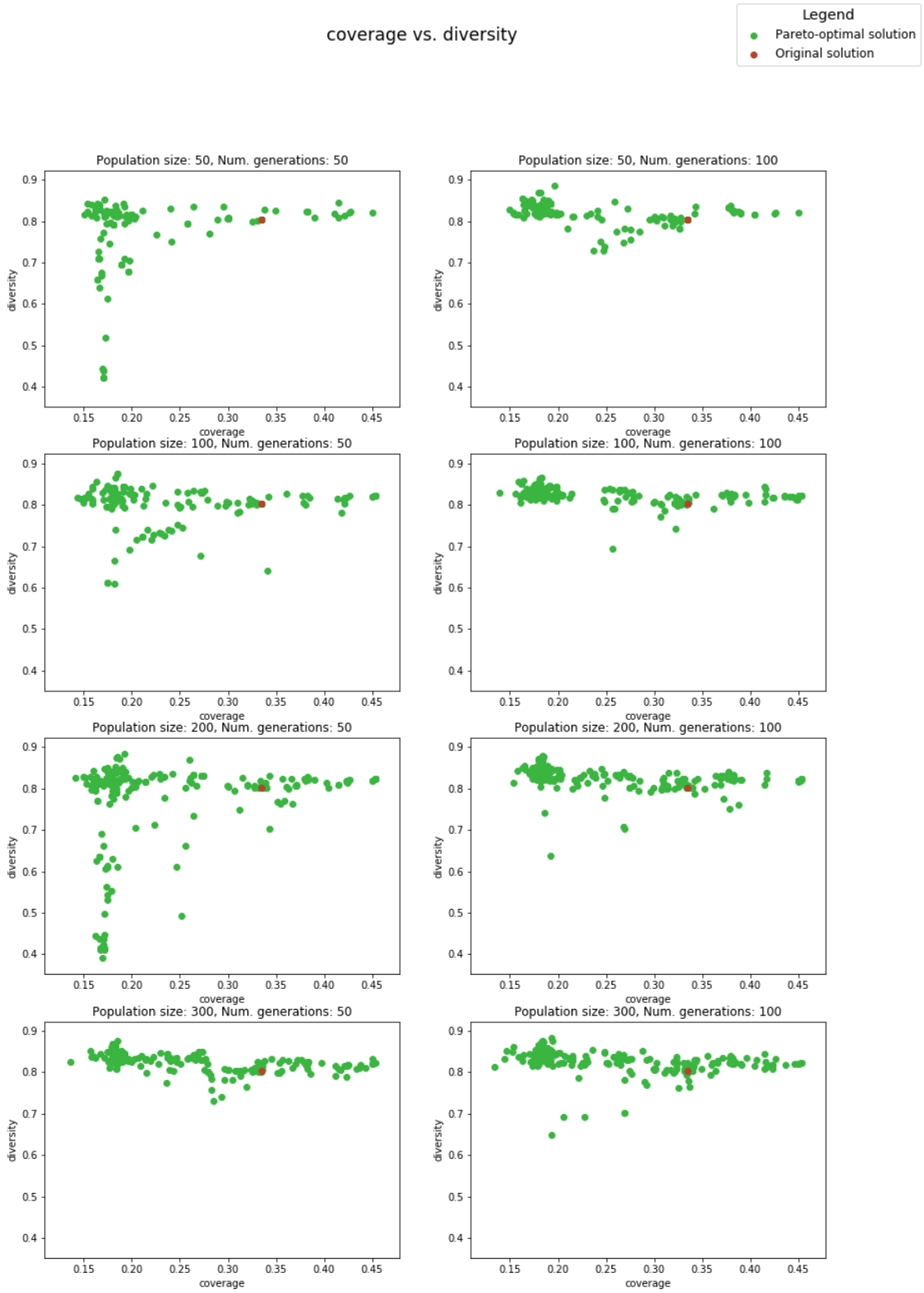

Figure 7.7: Coverage vs. diversity.

parameter values of Figures 7.8 to 7.10, there are no solutions that achieve a reasonable trade-off between these three metrics. For the larger parameter values in Figures 7.11 to 7.15 the situation improves slightly.

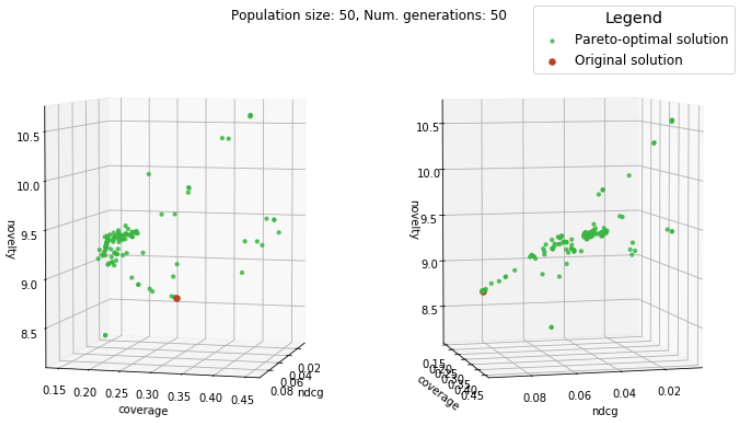

Figure 7.8: The NDCG, coverage and novelty for population size 50 and 50 generations.

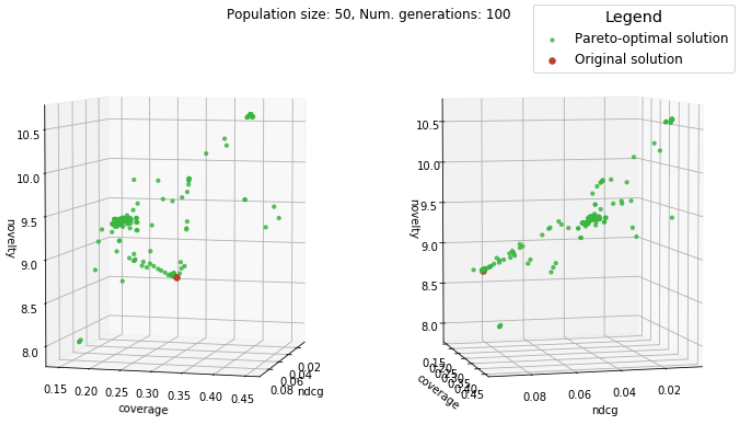

Figure 7.9: The NDCG, coverage and novelty for population size 50 and 100 generations.

Four Dimensions

So far, the graphs discussed trade-offs between at most three out of four metrics. Adding an extra dimension implies that it is no longer possible to visualise the solutions in easily interpretable graphs. Given that the main interest is to improve novelty, coverage and diversity while not discarding NDCG, Figures 7.16 to 7.23 show the solutions that perform at least as good as the baseline on those three criteria. The figures demonstrate that for smaller values of the parameters, the algorithm has trouble finding good solutions for all metrics. It does not find many solutions and the solutions that it finds often have a low NDCG. As the parameter values

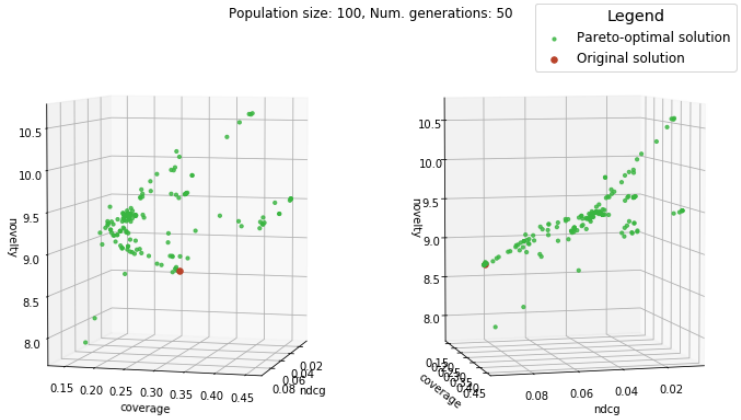

Figure 7.10: The NDCG, coverage and novelty for population size 100 and 50 generations.

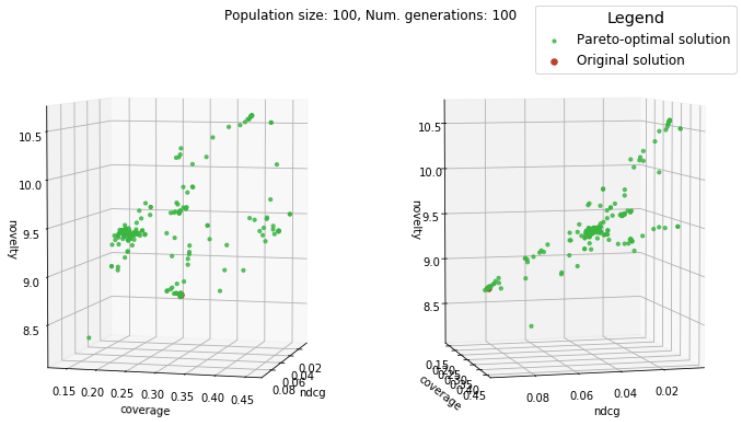

Figure 7.11: The NDCG, coverage and novelty for population size 100 and 100 generations.

Figure 7.12: The NDCG, coverage and novelty for population size 200 and 50 generations.

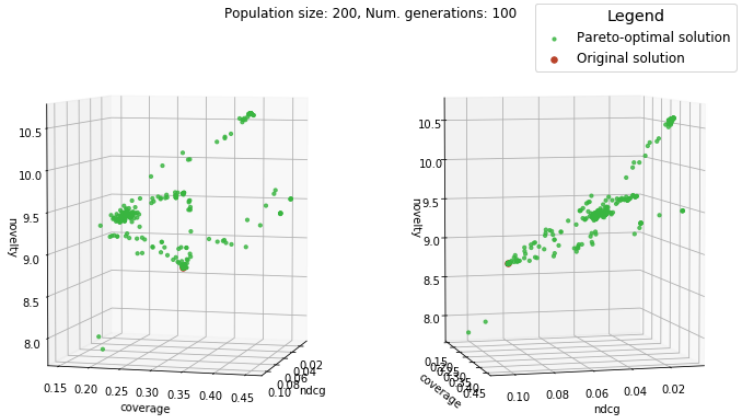

Figure 7.13: The NDCG, coverage and novelty for population size 200 and 100 generations.

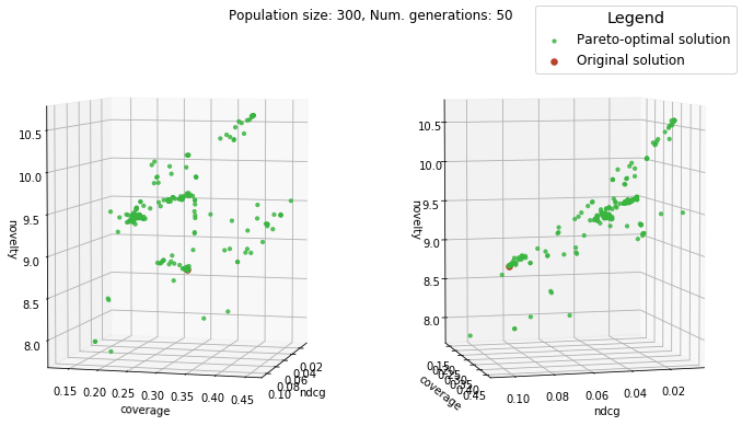

Figure 7.14: The NDCG, coverage and novelty for population size 300 and 50 generations.

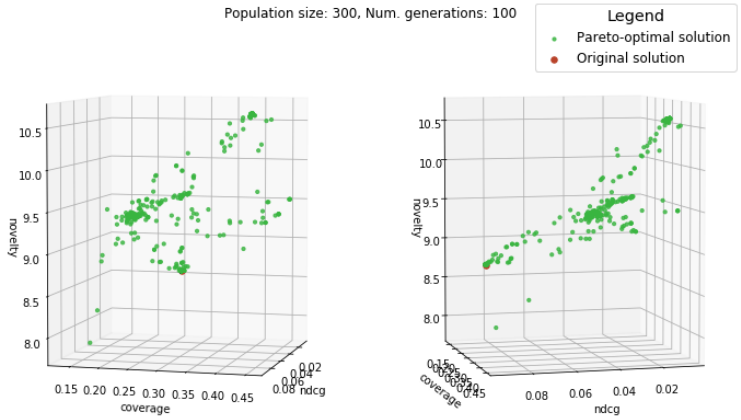

Figure 7.15: The NDCG, coverage and novelty for population size 300 and 100 generations.

increase, so do the number of solutions. Moreover, the quality of the found solutions increases as well because the NDCG does not deteriorate as strongly. The number of generations has a larger influence on the number of found solutions. For example, for 100 individuals and 50 generations, the algorithm only found 18 solutions that performed better on all metrics (except NDCG). Letting the same number of individuals evolve over 100 generations results in 44 solutions. Comparing Figure 7.18 and Figure 7.19 also shows that the latter settings find better trade-offs.

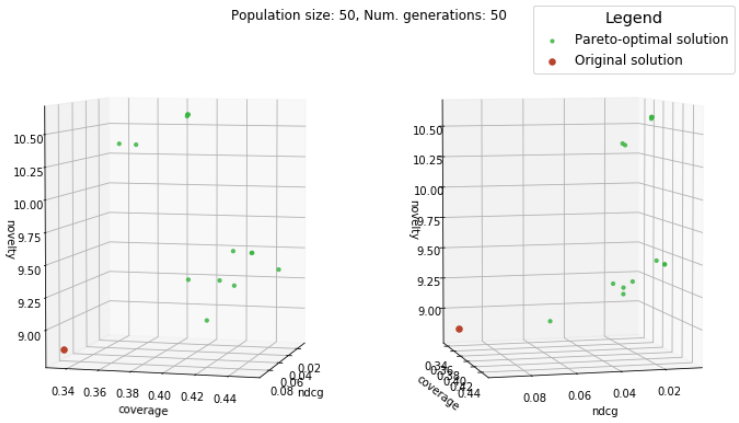

Figure 7.16: Solutions that perform at least as good in terms of coverage, novelty and diversity for population size 50 and 50 generations.

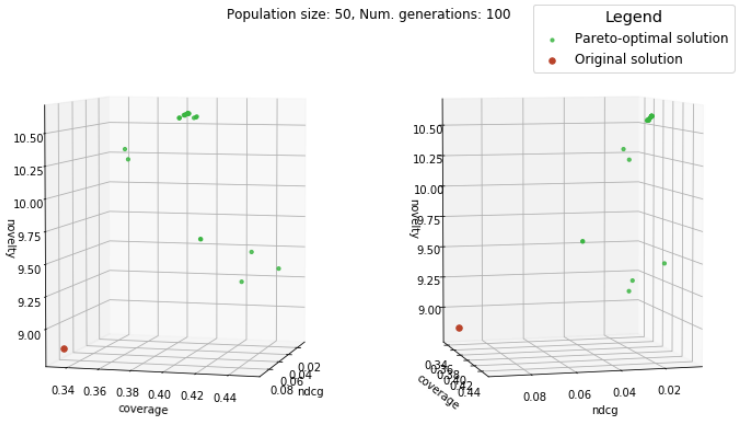

Figure 7.17: Solutions that perform at least as good in terms of coverage, novelty and diversity for population size 50 and 100 generations.

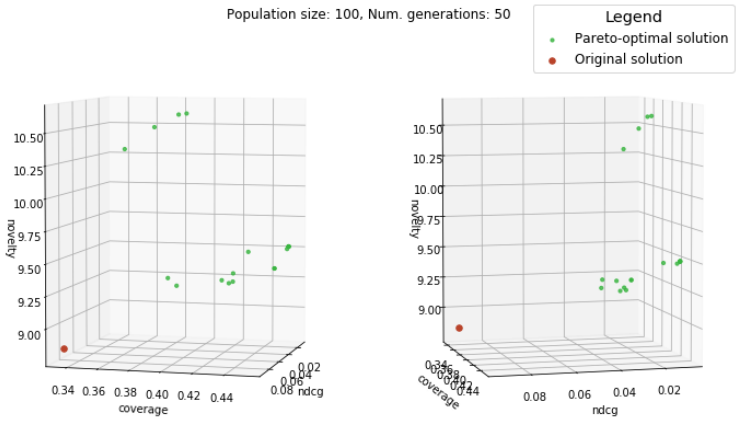

Figure 7.18: Solutions that perform at least as good in terms of coverage, novelty and diversity for population size 100 and 50 generations.

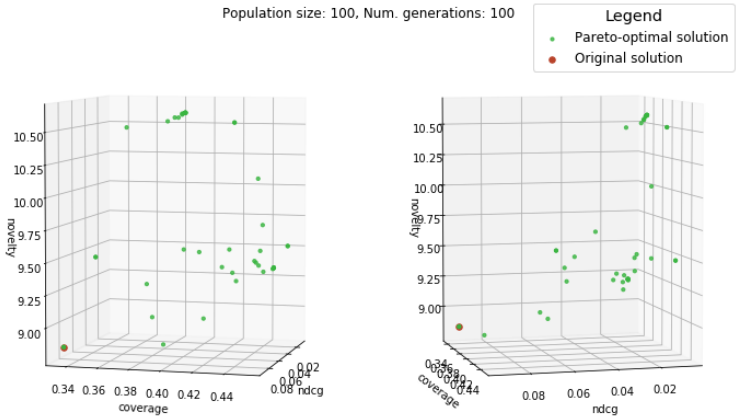

Figure 7.19: Solutions that perform at least as good in terms of coverage, novelty and diversity for population size 100 and 100 generations.

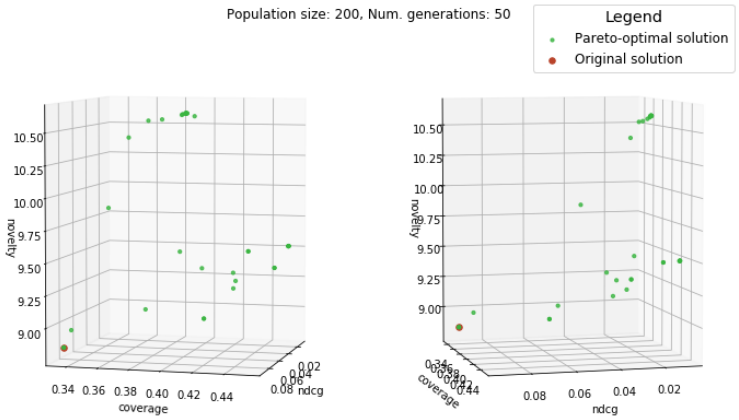

Figure 7.20: Solutions that perform at least as good in terms of coverage, novelty and diversity for population size 200 and 50 generations.

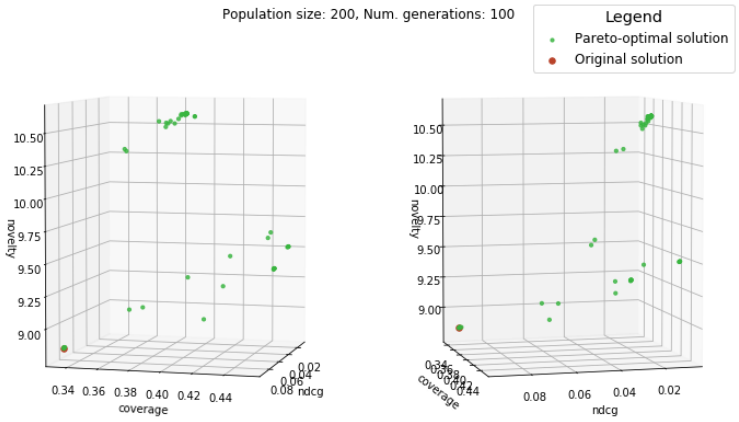

Figure 7.21: Solutions that perform at least as good in terms of coverage, novelty and diversity for population size 200 and 100 generations.

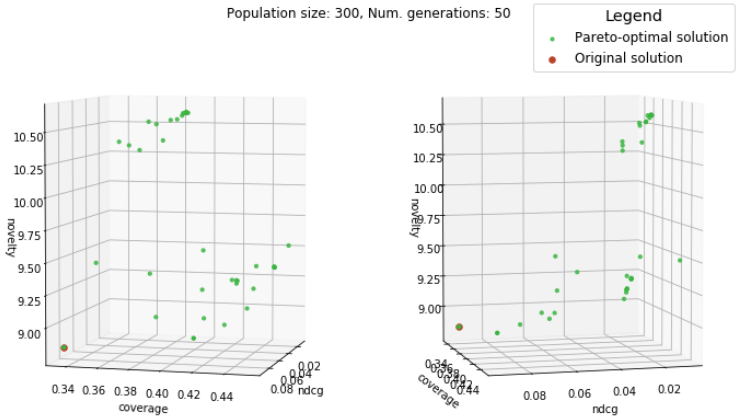

Figure 7.22: Solutions that perform at least as good in terms of coverage, novelty and diversity for population size 300 and 50 generations.

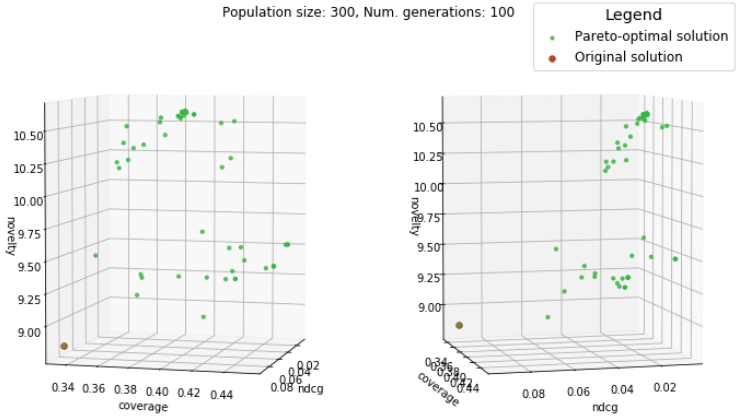

Figure 7.23: Solutions that perform at least as good in terms of coverage, novelty and diversity for population size 300 and 100 generations.

7.6 Discussion

The previous section presented many graphs as a means to convey the general trend in the results. While the figures provide a lot of information, they are not numerically precise. Therefore, the current section considers the underlying numbers used to make the figures, to precisely state how well the algorithm performs.

As mentioned before, there are little solutions that perform better in terms of coverage, novelty and diversity and maintain a reasonable NDCG. Table 7.7 lists a few scoring functions that achieve a sensible compromise between the performance metrics. Table 7.8 shows the corresponding performance for these functions. For a more in-depth comparison with the baseline, Appendix B shows the relative differences of the metrics between the baseline and the scoring functions. The formulas include several content-related features.

- `tz_xxx`:feature from the Tzanetakis classifier
- `dm_xxx`: feature from the Dortmund classifier
- `ros_xxx`: feature from the Rosamerica classifier
- `ism_xxx`: ballroom subgenre predicted by the Ismir04 Rhythm classifier
- `mirex_x`: cluster from the Mirex moods classifier
- `hotttnesss`: artist hotttnesss from The Echo Nest
- `familiarity`: artist familiarity from The Echo Nest

Chapter 5 and Appendix A provide more information on the different classifiers and features. Besides the content-related features, some of the found functions also include features from the collaborative data. `num_rat` is the number of ratings an item received, while `self_inf` signifies the self-information of an item.

7.6.1 Functions

Why do the scoring functions in Table 7.7 achieve better performance, especially in terms of coverage and novelty? To determine this, the behaviour of the functions and the features are studied in detail. The original score s originates from the BPR model. In principle, the score does not have bounds and depends on the input rating matrix. For this dataset, the score ranges from 0.75 to 6.21. Figure 7.24 shows the functions that depend only on the s variable in Table 7.7. For the functions in the 1st, 8th and 11th rows, items that originally received a high score around 3 or 6 will continue to score highly. However, items that score between 4 and 5 will be penalised. The function of the fifth row is a translated version. Unfortunately, the

	Pop. size	# gens	Scoring function
1	50	50	$\cos(\sin(s))$
2			$\sin(s/\texttt{tz\_reg})$
3	50	100	$\cos(\tanh((\texttt{dm\_blues} + \texttt{num\_rat}) \cdot \texttt{tz\_jazz}))$
4			$\cos(\tanh(\texttt{num\_rat} \cdot \texttt{tz\_jazz}))$
5	100	50	$\cos(\cos(s))$
6			$\cos((\sin(s)/\texttt{ros\_rock})$
7	100	100	$\texttt{dm\_funksoulrnb} + s$
8			$\tanh(\cos(\sin(s)))$
9			$\tanh((\texttt{num\_rat} \cdot s)/-6.74)$
10			$\tanh((\texttt{num\_rat} \cdot s)/-6.74 + \texttt{familiarity})$
11	200	50	$\sin(\cos(\sin(s)))$
12			$\cos(\cos(\texttt{ros\_jaz} + (\tanh(\texttt{dm\_raphiphop}) + s) \cdot s))$
13	200	100	$(\texttt{dm\_rock} + \texttt{hotttnesss} + \texttt{self\_inf} + \texttt{mirex\_1})/\tanh(\sin(s))$
14			$(\texttt{dm\_rock} + \sin(\cos(2.36)) + \texttt{self\_inf} + \texttt{mirex\_1})/\tanh(\sin(s))$
15	300	50	$\sin(s + 9.17)$
16			$\cos(\cos(\texttt{dm\_pop} \cdot \texttt{ism\_rumba}) + s)$
17	300	100	$\cos(\texttt{self\_inf}/s)$
18			$\cos(\texttt{ism\_chachacha})/\sin((\texttt{dm\_funksoulrnb} + \cos(s))$

Table 7.7: A few interesting scoring functions.

interpretation is a little more complicated. Typically, a user's score range is limited to only a part of the total score range. For example, a user could have scores between 3.3 and 4.3. In that case, the final ordering would reverse all the items from the original list. On the other hand, if a user's scores lie between 1.6 and 3, the original ordering would not change. The problem is that for some users, the RS will provide accurate recommendations, while for others it will focus only on novelty and coverage. This is undesirable because it would be better to provide a mix of both per user. In other words, while the metrics seem acceptable at a high level, the underlying recommendations will not be satisfactory for the users. An analogous interpretation follows for the 15th row.

The previous functions depend only on the original score of an item. Functions that also rely on other features are more interesting and have a better chance of providing a real, per-user trade-off. The functions in the 3rd and 4th rows do not depend on the original score at all. Instead, items with a low number of ratings, and a low probability of being a jazz song (or blues song in the 3rd row) receive a higher score. This is because the $\cos(\tanh(x))$ is decreasing for $x > 0$. In other words, it favours less popular songs that are likely not jazz or blues. The functions on row 9 and 10 also favour low scores and a low number of ratings, resulting in a better novelty, coverage and diversity.

The functions in row 13 and 14 of Table 7.7 also depend on several item features. These

	Pop. size	# gens	NDCG	Coverage	Novelty	Diversity
1	50	50	0.063341	0.414413	9.028852	0.807926
2			0.030898	0.420940	9.232488	0.813773
3	50	100	0.046388	0.404578	9.622844	0.815006
4			0.046325	0.404285	9.624598	0.815634
5	100	50	0.034142	0.423399	9.330914	0.818138
6			0.033461	0.385230	9.213196	0.818165
7	100	100	0.092718	0.335294	8.839651	0.805955
8			0.063341	0.414413	9.028852	0.807926
9			0.061880	0.447693	9.477741	0.812037
10			0.057543	0.449069	9.561734	0.812956
11	200	50	0.063341	0.414413	9.028852	0.807926
12			0.035892	0.424131	9.208680	0.814095
13	200	100	0.051480	0.451908	9.658818	0.819415
14			0.050134	0.453284	9.700299	0.820671
15	300	50	0.087555	0.416637	8.938546	0.805547
16			0.080077	0.433497	9.030072	0.807909
17	300	100	0.050488	0.436571	9.456571	0.817617
18			0.045776	0.363863	9.137676	0.802937

Table 7.8: The performance for the functions from Table 7.7.

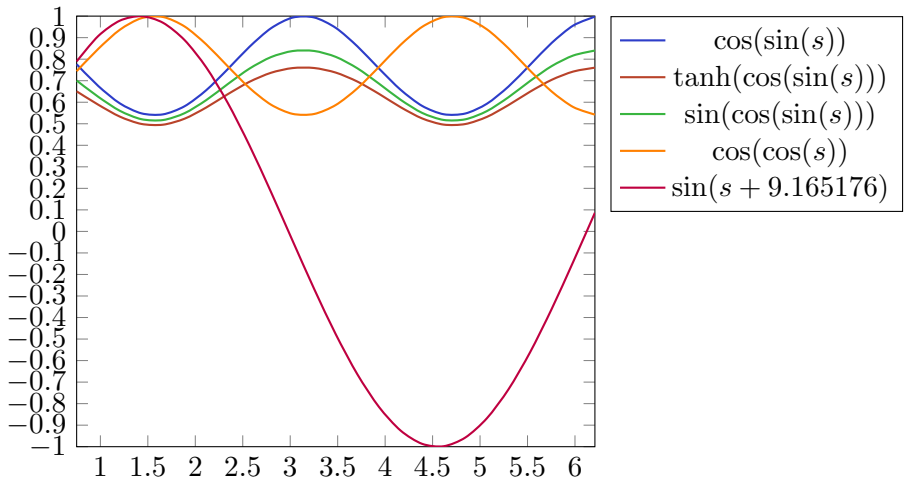

Figure 7.24: The reranking functions that only depend on the score.

functions favour a large self-information, reflected in the high novelty. They also attribute higher scores to rock songs that sound passionate, rousing and confident (first Mirex cluster). The denominator contains $\tanh(\sin(s))$. Depending on the range of scores, this may favour higher or lower scores. However, due to the hyperbolic tangent function, the influence of this factor will be relatively limited. The functions only differ in the fact that one also takes into account the artist hotttnesss. This results in a slightly higher NDCG but a slightly lower coverage, novelty and diversity.

7.6.2 Evaluation Method

On closer inspection, the evaluation method presented in Section 7.4 was not optimal. The evaluation method does not properly check for overfitting of the GP solutions. Ideally, the traditional RS would first be tuned. Then, it should compute recommendation lists for all users that serve as a new dataset for the GP step. This new dataset should then be split into train, validation and test sets. Finally, those sets should be used to train, tune and evaluate the GP step. That way, the found solutions can be checked on totally new lists of recommendations to make sure that they work well in general circumstances. For the evaluation method in Section 7.4, the recommendation lists will contain overlapping recommendations. More precisely on average 54.23% of the recommendations overlap for the RS trained on just the train set compared to the one trained on the train and simulation set.

Chapter 8

Conclusion

This work investigated the use of GP to enhance the performance of an existing RS. The first part reviewed the basic theory of RSs and EC. This provided a starting point for the literature summary. Over the last two decades, researchers have invested a lot of time examining RSs. Traditionally, RSs concentrate on accuracy as the main criterion to optimise. However, several studies pointed out that this can lead to reduced user satisfaction. Therefore, researchers have shifted their attention to other aspects of RSs such as coverage, novelty, diversity and serendipity. These metrics often impose conflicting requirements to the recommendations and thus, a good algorithm should attempt to find a balance between them.

While researchers have developed several techniques to achieve this goal, EC has gained traction lately. This optimisation method draws inspiration from biological evolution to find solutions to a problem that are as close to the global optimum as possible. One of the main strengths of EC is that it can optimise multiple, conflicting objectives at the same time. Hence, the technique is suitable to build an enhanced RS that focuses on multiple metrics simultaneously.

The algorithm presented in this work uses a particular variant of EC, namely GP. This variant evolves functions or computer programs. The GP step of the algorithm takes a list of recommendations produced by a regular RS as input and produces a score for each of the items. The score depends on the original score from the RS but also on content-related features. Next, the algorithm reranks the items based on the new score and returns that list to the end-user. Since GP maintains a set of different solutions, it can find several Pareto-optimal solutions, i.e. solutions that perform at least as good on all metrics and better for at least one metric compared to other solutions.

The content to recommend in this work was music. More specifically, the Taste Profile subset, related to the Million Song Dataset, was used for collaborative user data. The Million Song Dataset provides an extensive set of audio features for the songs in the Taste Profile subset.

Unfortunately, due to problems with the logistics of this dataset, an alternative dataset from AcousticBrainz was used instead.

With the algorithm and data in place, the next step is to verify if the algorithm attains the goal of optimising the RS performance in terms of all metrics. The algorithm has many parameters related to the original RS, the GP step and the algorithm as a whole. Since algorithm execution can take a long time, only a few parameters were varied while the others were chosen in an ad-hoc manner. Especially the population size and the number of generations in the GP step were important. These have a big influence on the quality of the found solutions, and on the execution time. The final parameter settings have to find a balance between these two. The experiments found that in almost all cases it is possible to find a good trade-off between pairs of metrics. However, finding a scoring function that achieves a good compromise between three or more metrics is much harder. For smaller population sizes and number of generations, the system is unable to find a good balance. For higher parameter values, the system finds a balance, but at the cost of increased computation time of the GP step. Fortunately, this part of the algorithm only has to be run very infrequently in comparison with the underlying RS. Once the scoring functions are known, they do not have to be recomputed often.

8.1 Future Work

The system presented in this work can serve as a basis for further research in several directions:

1. The evaluation method should be corrected. Section 7.6.2 discussed some issues with the evaluation method that should be addressed.

2. The influence of other GP parameters should be tested. As mentioned before, this work made ad-hoc decisions for some algorithm parameters. Potentially, different settings for those parameters may achieve better results. These settings could be numeric parameters, but also the choice of reproduction operators and selection mechanism.

3. The current implementation will not scale to multiple machines and very large datasets. Nevertheless, the algorithm design shows a lot of parallelism options. Several RSs have been implemented for clusters [1]. The GP step itself also provides some inherent parallelism. The EC library used to implement this project provides distributed computing capabilities, but unfortunately, some bugs in the library require quite a few workarounds to make it work.

4. This work focused on music recommendation in particular. However, with relatively little changes, it could be adapted for other datasets as well. Many of the algorithms discussed

[1] `https://spark.apache.org/docs/latest/mllib-collaborative-filtering.html`

in the related work chapter used MovieLens datasets. The advantage of this is that a comparison with those algorithms then becomes possible.

5. The algorithm can work with many types of features. This work used a combination of content-specific features from an external data source as well as features derived from the collaborative user data. One thing to research is what happens when only collaborative features are used. In that case, could the same scoring function be used for different datasets?

Appendices

Appendix A

Full Data Description

Chapter 5 discussed the different data sources used for this project. A part of the data originates from AcousticBrainz, and consists of class probabilities for 18 different classifiers. The classifiers attribute genres, moods, and audio-related features to each song. This appendix presents the different classifiers in more detail. AcousticBrainz uses SVM models from the Essentia project[1], which provides tools for music analysis.

A.1 Genres

There are multiple genre classifiers. Some of these are general while others make a subdivision within a specific genre.

A.1.1 Dortmund

The Dortmund genre classifier places songs into one of 9 general genre classes: alternative, blues, electronic, folkcountry, funksoulrnb, jazz, pop, raphiphop and rock [30]. The classifier reaches relatively high accuracies for folkcountry, jazz, raphiphop and rock. The other classes typically do not achieve such a high accuracy, possibly because the training material was not balanced. The aforementioned classes are overrepresented.

The data used for experimentation is classified for the most part as electronic, shown in Table A.1. This could indicate that a large fraction of the songs is misclassified by this classifier. In retrospect, this classifier's classes should probably not have been used to measure the diversity of the songs. Most of them will have a high value for the electronic class and a low value for the other classes.

[1] `https://essentia.upf.edu/`

Elec.	Folkcountry	Rock	Alt.	Blues	Jazz	Raphiphop	Pop
87.11%	4.97%	3.10%	2.07%	1.56%	0.86%	0.31%	0.0014%

Table A.1: Class percentages of songs in the data for the Dortmund classifier.

A.1.2 Rosamerica

This is another general genre classifier [27]. The classes are different from the Dortmund one: classical, dance, hip-hop, jazz, pop, rhythm'n'blues, rock and speech. It achieves a high accuracy for most classes, except for the pop and rhythm'n'blues classes, which are confused relatively often.

When looking at the class distribution in , the results are more evenly distributed than for the Dortmund classifier.

Rock	R&B	Pop	Hip-hop	Classical	Jazz	Dance	Speech
31.25%	19.54%	16,30%	12.33%	7.42%	7.38%	5.49%	0.29%

Table A.2: Class percentages of songs in the data for the Rosamerica classifier.

A.1.3 Tzanetakis

The last general genre classifier was developed by Tzanetakis and Cook [49]. Their system makes a distinction between the following genres: blues, classical, country, disco, hip-hop, jazz, metal, pop, reggae and rock. The researchers believe that they reach close to human performance levels. The class distribution in Table A.3 shows that here too, there is a significant class imbalance, pointing at potential misclassifications.

Jazz	Rock	Hip-hop	Pop	Blues	Classical	Reggae	Country	Metal	Disco
92.60%	4.13%	1.49%	0.83%	0.48%	0.28%	0.086%	0.048%	0.035%	0.030%

Table A.3: Class percentages of songs in the data for the Tzanetakis classifier.

A.1.4 Electronic Music

This classifier subdivides electronic music into several subgenres: ambient, drum 'n bass, house, techno and trance. Note that the classifier is also used on songs even though they are not necessarily electronic. Therefore, the output will not always be semantically meaningful. Table A.4 shows the class distribution for the current classifier. Most of the songs are classified as ambient.

Ambient	Trance	House	Techno	Drum 'n bass
62.61%	28.37%	7.10%	1.02%	0.91%

Table A.4: Class percentages of songs in the data for the Electronic classifier.

A.1.5 Ballroom

Another subgenre classifier is the 'Ismir04 rhythm' classifier based on the ballroom dataset [16]. The dataset contains ballroom music from 8 possible subgenres: Cha Cha, Jive, Quickstep, Rumba, Samba, Tango, Viennese Waltz and Slow Waltz. However, some of these subgenres contain further subdivisions: Rumba is further split into American, international and miscellaneous rumba. Again, this classifier is used for every item, even when the song is not a ballroom song. The class distribution is shown in Table A.5.

Cha Cha	Vien. Waltz	Tango	Am. Rumba	Waltz	Samba	Jive	Int. Rumba	Quickstep	Misc. Rumba
30.71%	24.81%	24.76%	9.61%	3.30%	2.72%	2.22%	1.50%	0.27%	0.10%

Table A.5: Class percentages of songs in the data for the ballroom classifier.

A.2 Moods

Besides genres, there are also classifiers to identify moods in songs. AcousticBrainz uses the following default models to identify the moods. Note that some of these moods are not necessarily feelings:

- Acoustic or not acoustic
- Electronic or not electronic
- Agressive or not aggressive
- Relaxed or not relaxed
- Happy or not happy
- Sad or not sad
- Party or not party
- Mirex

Table A.6 shows the class distributions of the aformentioned classifiers. All mood classifiers are binary, except the last one. Hu and Downie found that classifying the mood of a song is

challenging because there is no standardised way to describe it [32]. Therefore, they worked out a cluster-based approach with the following clusters:

1. passionate, rousing, confident, boisterous, rowdy
2. rollicking, cheerful, fun, sweet, amiable/good natured
3. literate, poignant, wistful, bittersweet, autumnal, brooding
4. humorous, silly, campy, quirky, whimsical, witty, wry
5. aggressive, fiery, tense/anxious, intense, volatile, visceral

The class distribution of this classifier is given in Table A.7.

Mood	**Positive**	**Negative**
Acoustic	24.31%	75.69%
Aggressive	19.78%	80.22%
Electronic	59.41%	40.59%
Happy	27.83%	72.17%
Party	27.21%	72.79%
Relaxed	68.35%	31.65%
Sad	24.31%	75.69%

Table A.6: Class percentages of songs in the data for the binary mood classifiers.

Cluster 1	**Cluster 2**	**Cluster 3**	**Cluster 4**	**Cluster 5**
1.28%	11.97%	20.72%	3.69%	62.34%

Table A.7: Class percentages of songs in the data for the Mirex classifier.

A.3 Other Classifiers

Finally, there are a few other classifiers that tag songs in various ways:

- Danceable or not Danceable
- Male or female
- Dark or bright (timbre)
- Tonal or atonal

- Voice or instrumental

Table A.8 shows these distributions. Some classes do not correspond to human intuition. For example, more than half of the songs is classified as instrumental. This is probably a misclassification. Moreover, less than half of all songs is classified as tonal, while in reality the majority of the songs has a tonal centre.

Classifier	**Positive**	**Negative**
Danceability	35.70%	64.30%
Is male	37.52%	62.48%
Bright timbre	41.12%	58.88%
Tonal	37.75%	62.25%
Instrumental	52.65%	47.35%

Table A.8: Class percentages of songs in the data for the remaining classifiers.

Appendix B

Full Results Table

	Pop. size	# gens	NDCG		Coverage		Novelty		Diversity	
			Absolute	Relative	Absolute	Relative	Absolute	Relative	Absolute	Relative
1	50	50	0.063341	-31.62%	0.414413	+23.66%	9.028852	+2.16%	0.807926	+0.64%
2			0.030898	-66.64 %	0.420940	+25.61%	9.232488	+4.46%	0.813773	+1.37%
3	50	100	0.046388	-49.92%	0.404578	+20.73%	9.622844	+8.88%	0.815006	+1.52%
4			0.046325	-49.99%	0.404285	+20.64%	9.624598	+8.90%	0.815634	+1.60%
5	100	50	0.034142	-63.14%	0.423399	+26.34%	9.330914	+5.58%	0.818138	+1.91%
6			0.033461	-63.88%	0.385230	+14.95%	9.213196	+4.25%	0.818165	+1.98%
7	100	100	0.092718	+0.09%	0.335294	+0.05%	8.839651	+0.02%	0.805955	+0.40%
8			0.063341	-31.62%	0.414413	+23.66%	9.028852	+2.16%	0.807926	+0.64%
9			0.061880	-33.20%	0.447693	+33.59%	9.477741	+7.24%	0.812037	+1.15%
10			0.057543	-38.00%	0.449069	+34.00%	9.561734	+8.19%	0.812956	+1.27%
11	200	50	0.063341	-31.62%	0.414413	+23.66%	9.028852	+2.16%	0.807926	+0.64%
12			0.035892	-61.25%	0.424131	+26.56%	9.208680	+4.19%	0.814095	+1.41%
13	200	100	0.051480	-44.42%	0.451908	+34.85%	9.658818	+9.29%	0.819415	+2.07%
14			0.050134	-45.88%	0.453284	+35.26%	9.700299	+9.76%	0.820671	+2.23%
15	300	50	0.087555	-5.48%	0.416637	+24.33%	8.938546	+1.14%	0.805547	+0.34%
16			0.080077	-13.55%	0.433497	+29.36%	9.030072	+2.17%	0.807909	+0.64%
17	300	100	0.050488	-45.50%	0.436571	+30.27%	9.456571	+7.00%	0.817617	+1.85%
18			0.045776	-50.58%	0.363863	+8.58%	9.137676	+3.39%	0.802937	+0.02%

Table B.1: The performance for the functions from Table 7.7 relative to the baseline performance.

Bibliography

[1] (). 'Accuracies and confusion matrices for default models,' [Online]. Available: `https://acousticbrainz.org/datasets/accuracy` (visited on 12/04/2020).

[2] Adomavicius, G. and Kwon, Y., 'Improving aggregate recommendation diversity using ranking-based techniques,' *IEEE Transactions on Knowledge and Data Engineering*, vol. 24, no. 5, pp. 896–911, 2011.

[3] Adomavicius, G. and Kwon, Y., 'Optimization-based approaches for maximizing aggregate recommendation diversity,' *INFORMS Journal on Computing*, vol. 26, no. 2, pp. 351–369, 2014.

[4] Adomavicius, G. and Tuzhilin, A., 'Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions,' *IEEE transactions on knowledge and data engineering*, vol. 17, no. 6, pp. 734–749, 2005.

[5] Aggarwal, C. C. *et al.*, *Recommender systems*. Springer, 2016, vol. 1.

[6] Anderson, C., 'The long tail,' *Wired Magazine*, vol. 12, no. 10, pp. 170–177, 2004.

[7] Banzhaf, W., Nordin, P., Keller, R. E. and Francone, F. D., *Genetic programming*. Springer, 1998.

[8] Barraza-Urbina, A., Heitmann, B., Hayes, C. and Carrillo-Ramos, A., 'Xplodiv: An exploitation-exploration aware diversification approach for recommender systems,' in *The Twenty-Eighth International Flairs Conference*, 2015.

[9] Bennett, J., Lanning, S. *et al.*, 'The netflix prize,' in *Proceedings of KDD cup and workshop*, Citeseer, vol. 2007, 2007, p. 35.

[10] Bertin-Mahieux, T., Ellis, D. P., Whitman, B. and Lamere, P., 'The million song dataset,' in *Proceedings of the 12th International Conference on Music Information Retrieval (ISMIR 2011)*, 2011.

[11] Beyer, H.-G. and Schwefel, H.-P., 'Evolution strategies–a comprehensive introduction,' *Natural computing*, vol. 1, no. 1, pp. 3–52, 2002.

[12] Borchers, A., Herlocker, J., Konstan, J. and Reidl, J., 'Ganging up on information overload,' *Computer*, vol. 31, no. 4, pp. 106–108, 1998.

[13] Boussaïd, I., Lepagnot, J. and Siarry, P., 'A survey on optimization metaheuristics,' *Information sciences*, vol. 237, pp. 82–117, 2013.

[14] Bridge, D. and Kelly, J. P., 'Ways of computing diverse collaborative recommendations,' in *International conference on adaptive hypermedia and adaptive web-based systems*, Springer, 2006, pp. 41–50.

[15] Brynjolfsson, E., Hu, Y. J. and Smith, M. D., 'The longer tail: The changing shape of amazon's sales distribution curve,' *Available at SSRN 1679991*, 2010.

[16] Cano, P., Gómez, E., Gouyon, F., Herrera, P., Koppenberger, M., Ong, B., Serra, X., Streich, S. and Wack, N., 'Ismir 2004 audio description contest,' 2006.

[17] Cramer, N. L., 'A representation for the adaptive generation of simple sequential programs,' in *Proceedings of the first international conference on genetic algorithms*, 1985, pp. 183–187.

[18] Cui, L., Ou, P., Fu, X., Wen, Z. and Lu, N., 'A novel multi-objective evolutionary algorithm for recommendation systems,' *Journal of Parallel and Distributed Computing*, vol. 103, pp. 53–63, 2017.

[19] Deb, K., Agrawal, S., Pratap, A. and Meyarivan, T., 'A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: Nsga-ii,' in *International conference on parallel problem solving from nature*, Springer, 2000, pp. 849–858.

[20] (). 'Downloads,' [Online]. Available: `https://acousticbrainz.org/download` (visited on 11/04/2020).

[21] Eiben, A. E., Smith, J. E. *et al.*, *Introduction to evolutionary computing*. Springer, 2003, vol. 53.

[22] Eiben, A. E. and Smit, S. K., 'Evolutionary algorithm parameters and methods to tune them,' in *Autonomous search*, Springer, 2011, pp. 15–36.

[23] Ekstrand, M. D., 'The lkpy package for recommender systems experiments: Next-generation tools and lessons learned from the lenskit project,' *arXiv preprint arXiv:1809.03125*, 2018.

[24] Ekstrand, M. D., Harper, F. M., Willemsen, M. C. and Konstan, J. A., 'User perception of differences in recommender algorithms,' in *Proceedings of the 8th ACM Conference on Recommender systems*, 2014, pp. 161–168.

[25] Ge, M., Delgado-Battenfeld, C. and Jannach, D., 'Beyond accuracy: Evaluating recommender systems by coverage and serendipity,' in *Proceedings of the fourth ACM conference on Recommender systems*, 2010, pp. 257–260.

[26] Geng, B., Li, L., Jiao, L., Gong, M., Cai, Q. and Wu, Y., 'Nnia-rs: A multi-objective optimization based recommender system,' *Physica A: Statistical Mechanics and its Applications*, vol. 424, pp. 383–397, 2015.

[27] Guaus, E., 'Audio content processing for automatic music genre classification: Descriptors, databases, and classifiers,' Ph.D. dissertation, Universitat Pompeu Fabra, Barcelona, Spain, 2009.

[28] Guimarães, A., Costa, T. F., Lacerda, A., Pappa, G. L. and Ziviani, N., 'Guard: A genetic unified approach for recommendation,' *Journal of Information and Data Management*, vol. 4, no. 3, pp. 295–310, 2013.

[29] Holland, J. H. *et al.*, *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.

[30] Homburg, H., Mierswa, I., Möller, B., Morik, K. and Wurst, M., 'A benchmark dataset for audio classification and clustering.,' in *ISMIR*, vol. 2005, 2005, pp. 528–31.

[31] Horváth, T. and Carvalho, A. C. de, 'Evolutionary computing in recommender systems: A review of recent research,' *Natural Computing*, vol. 16, no. 3, pp. 441–462, 2017.

[32] Hu, X. and Downie, J. S., 'Exploring mood metadata: Relationships with genre, artist and usage metadata.,' in *ISMIR*, 2007, pp. 67–72.

[33] Jannach, D., Lerche, L. and Zanker, M., 'Recommending based on implicit feedback,' in *Social Information Access*, Springer, 2018, pp. 510–569.

[34] Kaminskas, M. and Bridge, D., 'Diversity, serendipity, novelty, and coverage: A survey and empirical analysis of beyond-accuracy objectives in recommender systems,' *ACM Transactions on Interactive Intelligent Systems (TiiS)*, vol. 7, no. 1, pp. 1–42, 2016.

[35] Koren, Y., 'The bellkor solution to the netflix grand prize,' *Netflix prize documentation*, vol. 81, no. 2009, pp. 1–10, 2009.

[36] Kotkov, D., Konstan, J. A., Zhao, Q. and Veijalainen, J., 'Investigating serendipity in recommender systems based on real user feedback,' in *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*, 2018, pp. 1341–1350.

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

[38] Lamere, P., *Artist similarity, familiarity and hotness*, 2009. [Online]. Available: `https://musicmachinery.com/2009/05/25/artist-similarity-familiarity-and-hotness/`.

[39] Lin, Q., Wang, X., Hu, B., Ma, L., Chen, F., Li, J. and Coello Coello, C. A., 'Multiobjective personalized recommendation algorithm using extreme point guided evolutionary computation,' *Complexity*, vol. 2018, 2018.

[40] Marler, R. T. and Arora, J. S., 'Survey of multi-objective optimization methods for engineering,' *Structural and multidisciplinary optimization*, vol. 26, no. 6, pp. 369–395, 2004.

[41] McNee, S. M., Riedl, J. and Konstan, J. A., 'Being accurate is not enough: How accuracy metrics have hurt recommender systems,' in *CHI'06 extended abstracts on Human factors in computing systems*, 2006, pp. 1097–1101.

[42] (2016). 'Million song dataset echo nest mapping archive,' [Online]. Available: `https://labs.acousticbrainz.org/million-song-dataset-echonest-archive/` (visited on 11/04/2020).

[43] Nakatsuji, M., Fujiwara, Y., Tanaka, A., Uchiyama, T., Fujimura, K. and Ishida, T., 'Classical music for rock fans? novel recommendations for expanding user interests,' in *Proceedings of the 19th ACM international conference on Information and knowledge management*, 2010, pp. 949–958.

[44] Onuma, K., Tong, H. and Faloutsos, C., 'Tangent: A novel,'surprise me', recommendation algorithm,' in *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2009, pp. 657–666.

[45] Pan, R., Zhou, Y., Cao, B., Liu, N. N., Lukose, R., Scholz, M. and Yang, Q., 'One-class collaborative filtering,' in *2008 Eighth IEEE International Conference on Data Mining*, IEEE, 2008, pp. 502–511.

[46] Rendle, S., Freudenthaler, C., Gantner, Z. and Schmidt-Thieme, L., 'Bpr: Bayesian personalized ranking from implicit feedback,' *arXiv preprint arXiv:1205.2618*, 2012.

[47] Ribeiro, M. T., Lacerda, A., Veloso, A. and Ziviani, N., 'Pareto-efficient hybridization for multi-objective recommender systems,' in *Proceedings of the sixth ACM conference on Recommender systems*, 2012, pp. 19–26.

[48] Smyth, B. and McClave, P., 'Similarity vs. diversity,' in *International conference on case-based reasoning*, Springer, 2001, pp. 347–361.

[49] Tzanetakis, G. and Cook, P., 'Musical genre classification of audio signals,' *IEEE Transactions on speech and audio processing*, vol. 10, no. 5, pp. 293–302, 2002.

[50] Vargas, S. and Castells, P., 'Rank and relevance in novelty and diversity metrics for recommender systems,' in *Proceedings of the fifth ACM conference on Recommender systems*, 2011, pp. 109–116.

[51] Wang, S., Gong, M., Li, H. and Yang, J., 'Multi-objective optimization for long tail recommendation,' *Knowledge-Based Systems*, vol. 104, pp. 145–155, 2016.

[52] Wang, S., Gong, M., Ma, L., Cai, Q. and Jiao, L., 'Decomposition based multiobjective evolutionary algorithm for collaborative filtering recommender systems,' in *2014 IEEE Congress on Evolutionary Computation (CEC)*, IEEE, 2014, pp. 672–679.

[53] Zhou, T., Kuscsik, Z., Liu, J.-G., Medo, M., Wakeling, J. R. and Zhang, Y.-C., 'Solving the apparent diversity-accuracy dilemma of recommender systems,' *Proceedings of the National Academy of Sciences*, vol. 107, no. 10, pp. 4511–4515, 2010.

[54] Zhou, T., Ren, J., Medo, M. and Zhang, Y.-C., 'Bipartite network projection and personal recommendation,' *Physical review E*, vol. 76, no. 4, p. 046 115, 2007.

[55] Ziegler, C.-N., McNee, S. M., Konstan, J. A. and Lausen, G., 'Improving recommendation lists through topic diversification,' in *Proceedings of the 14th international conference on World Wide Web*, 2005, pp. 22–32.

[56] Zuo, Y., Gong, M., Zeng, J., Ma, L. and Jiao, L., 'Personalized recommendation based on evolutionary multi-objective optimization [research frontier],' *IEEE Computational Intelligence Magazine*, vol. 10, no. 1, pp. 52–62, 2015.