Recommender Systems with Evolutionary Algorithms: Many-Objective Optimization for Large-Scale Music Recommendation

Sharwin P. Bobde

A demonstration showing the reliability of serving users recommendations with trade-off for **large music collections**, by leveraging diverse **Recommender Systems** and **Evolutionary Algorithms**.

TU Delft

Recommender Systems with Evolutionary Algorithms: Many-Objective Optimization for Large-Scale Music Recommendation

by

Sharwin P. Bobde

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Friday July 9, 2021 at 15:30 CEST.

Student number:	5011639	
Project duration:	November 13, 2020 – July 03, 2021	
Supervisors:	Dr. Cynthia Liem and Dr. Annibale Panichella	
Thesis committee:	Dr. Cynthia Liem	TU Delft,
	Prof. Dr. Alan Hanjalic	TU Delft, *Responsible Professor*
	Prof. Dr. Peter Bosman	TU Delft and CWI, *External*

TU Delft

Preface

What lies in front of you is the distillation of nine months of effort; a study to understand and solve the scientific and engineering challenges of an extremely niche domain at its cutting edge. Using Evolutionary Algorithms for the Many-objective optimisation problem of user-centric recommendations at a large scale is a scientifically underexamined application. It holds the key to new possibilities, such as interaction design where users can configure their recommender system and thus have more control over how they explore vast amounts of multimedia data. However, is this strategy reliable? How can it be adapted for large amounts of data in a usable manner? Is it adversely affected by the underlying distribution of the data that is available? These questions require attention to make sure these systems will work well in the future. This research was carried out as part of obtaining a Master's degree in Computer Science with a specialization in Data Science and Technology at Delft University of Delft (*Technische Universiteit Delft*). The committee that evaluated and enforced the quality of this thesis consisted of Prof. Dr. Alan Hanjalic as the responsible supervisor and chairperson of the committee, Dr. Cynthia Liem as the regular supervisor, and Prof. Dr. Peter Bosman as the external member of the committee.

As a computer scientist and engineer, I love solving diverse problems that require unique solutions. I had some background in multimedia computing which I acquired while doing my Bachelor's in Pune, India. I was drawn to the Multimedia Computing (MMC) research group at TU Delft, and the admission committee was kind enough to grant me admission to the Master's program.

On the 10th of March, 2020, I met with Cynthia to talk about my thesis. We discussed about the possibilities of exploring Evolutionary Algorithms to make the user-experience of Recommender Systems better for everyone. Before starting the thesis in November of 2020, and after going through the literature, we thought "Can this actually work for real-world scenarios, where users are really diverse and have different intentions while while using Recommender Systems?" Everything that happened after that line of thought has culminated into this thesis report.

This entire research was conducted from my 13m^2 room, in a foreign land during a time when the entire world was descending into chaos. During the thesis timeline, I could not physically meet both my regular supervisors, Cynthia and Annibale. I am extremely glad they are two of the kindest and smartest people I know, and never gave me a chance to question the quality of education I was receiving. They gave me their time and attention at every point in this journey and always pushed for better quality. They criticised me where required and guided me exceedingly well, and for that reason I am a better version of myself today. I thank you from the bottom of my heart.

This research would not have been possible without the people and infrastructure of two extraordinary research groups in Delft: the Multimedia Computing (MMC) research group and the Software Engineering Research Group (SERG).

I would like to thank my parents who are in India during an extremely difficult time and my sister who is acting as an exceptional doctor at the frontlines in Germany. I would like to thank my friends in India to continue to support me from afar. I will also like to thank my friends here in the Netherlands who have become my second family and make me feel at home. I am extremely honoured to be supported by so many people while I can only be in contact with a few. Each one one of us has seen enough pain, death, and sacrifice and still continue to work towards a better future.

To you, the reader, I appreciate your interest in my work. I hope the knowledge within these pages can help you build a fairer world and advance the state-of-the-art to better serve the users.

Sharwin P. Bobde
Delft, 1st of July, 2021.

Abstract

Using Recommender Systems with Evolutionary Algorithms is an extremely niche domain. It holds the key to enabling new user interaction designs, where users can effectively configure their experience with a Recommender System. This thesis answers important questions about the scientific aspects of its application to large-scale data through a rigorous experimental design. We use one of the largest publicly accessible music listening histories dataset to analyse if the methodology works well for large real-world tasks. The dataset has been used to simulate various real-world scenarios for the experimental design.

The methodology fuses the recommendations generated by an unspecified number of recommenders. In this study, we have used three recommenders, which have specialised goals in terms of user-centric metrics. We use three different Evolutionary Algorithms to analyse the capability of different EA strategies for generating a near-optimal set of trade-offs on user-centric metrics. We have performed elaborate qualitative and quantitative analyses of the system to understand how various aspects of the system affect the final set of solutions.

Contents

1 Introduction

Recommender Systems (RS) have become a prominent aspect of our day-to-day life. They aim to present us with a limited set of options that are relevant to the users or which they would like. Here, the true meaning of 'relevant' and 'like' depend on the domain of concern and the user's desires while using a system. For the most part, at present, users do not have a lot of say in what kind of experience their recommendation lists provide *i.e.* they are not highly configurable. Research efforts in the direction of application of Evolutionary Algorithms (EA) for recommendation tasks have attempted to diversify the recommendations or provide multiple options to users (incorporating preference information) at the same time [1, 2, 3].

The scientific domain of providing the user with preference-centric configurable recommendations is extremely niche. The techniques that may enable these interfaces need a lot of interdisciplinary research effort. The usable interfaces for such systems are on the horizon. In this Masters' thesis, we study a technique of late-stage fusion of different RSs *i.e.* combine the output lists of multiple RSs that have specialised objectives such as maximizing recall, serving novel recommendations, being more personalised, etc. We explore and analyse the reliability of a framework that can potentially give users more control over the nature of the presented items by providing a set of trade-offs of the specified objectives through Multi-objective Optimization (MOO). We specifically solve a Many-objective Optimization (MaOO) problem, which is a subset of MOO problems, where the number of objectives we try to optimise is more than three. MaOO problems are considered separately because their high-dimensional nature makes them more difficult to solve. We will also set up an experimental design in a manner that gives us more insight into the true inner workings of the complete pipeline that delivers recommendations.

That being said, it is important to also consider that no matter which scientific solution we come up with, it will be useless if it cannot be used in the industry. For many popular services that require recommendations, the RS needs to be run on large volumes of data that comes from really diverse users. This presents many unforeseeable engineering problems, solutions for which go beyond the discipline of Recommender Systems. Therefore, as an appendage of the scientific task, we will explore good engineering solutions to make such a system feasible to use for a large volume of user data with limited computing infrastructure.

1.1. Music Recommendation

In this thesis, we will demonstrate the framework to understand the problems with large-scale Music Recommendation. The choice of the application domain of music recommendation is apt because it is a problem where there is a true need of a configurable, user-aware personalized recommendation. The application domain demands new avenues for streaming services and users to escape filter bubbles [4]. In the 2020s, our music consumption experience will be shaped by the fact that there are many problems with the recommendations. These arise with companies having millions of users to serve, who have billions of interactions with the available songs. To make sure they have a reliable service, these companies use *industry-standard* techniques for recommendation and data processing that computationally scale well with growing users, but are not necessarily the best methods from a user satisfaction perspective.

Good exploration of large music collections in a reliable fashion is a multipronged problem. Solving this problem will enable a user to find desirable, relevant, and provocative music. Users want music socioculturally and musically related to what they have heard before while still getting novel recommendations through which they can diversify taste [5, 6]. The willingness to strike this balance is different for different users as it is extremely subjective and dynamic (something that cannot even be modelled using their listening behaviour over time). While we try to do this, we have a very narrow window to catch the user's attention and have to show a limited set of items to reduce cognitive fatigue as the user scrolls past tens of recommendations.

1.2. Application Perspective

For any developed solution that addresses the problems in Section 1.1 there will be obstacles external to the core scientific problem. It is best to avoid doing scientific research that can never be used. Real-world data for user interactions which shows implicit-feedback[1] can sum up to tens or hundreds of Terabytes (TiB). While one would think these problems can be solved by simply using the cloud along with Distributed Computing to use *'infinite'* storage and compute power, it does not work. The various disparate facets/modules of the problem have various specialised problems, exacerbated by the nature of the facet.

For instance, data can be stored on cloud storage or data lakes, but if individual data files are not retrieved efficiently, it will make the cloud usage cost skyrocket. If the data is not preprocessed properly distributed computing for pairwise comparison can take an unexpectedly large amount of time due to the required serialization and deserialization while communicating between different nodes. We can process an infinite amount of data and if there is a spill, we use the disk space instead of the main memory, but we cannot do the same for the heap which is required to keep track of all objects used by the program, and the program will terminate if we run out of the allocated space in the main memory and swap.

1.3. Research Questions

Given the problems mentioned above — namely, the investigation of a theorised recommendation framework, making sure it works in a computationally feasible manner, and analysing the reliability of the framework — this thesis will answer the following research questions.

- ***RQ1:*** *How to perform late-stage recommendation fusion, for large-scale user data which gives a set of near-optimal options over user-centric recommendation quality metrics?*
- ***RQ2:*** *How to make the developed recommendation framework work reliably with large-scale user data, in a manner that can be used in a practical industry scenario?*
- ***RQ3:*** *To what extent do variations in the user base and their listening behaviour, through time, affect the delivered recommendations? And how?*

Answering **RQ1** will give a generalised framework that is capable of fusing recommendations from an unspecified number of recommenders with specialised user-centric goals. These RSs will use implicit feedback information. The framework should be able to generate a near-optimal set of fusion solutions with a trade-off for specified user-centric objectives/quality metrics. This includes showing a complete demonstration and validating with real-world data.

Answering **RQ2** will require pushing the developed framework to its extreme limits by adding as much user data as possible and analysing the ability of the framework to process the data efficiently with limited computational infrastructure. This means we have to do a conceptual and practical analysis of the system from a performance perspective. Then implement the framework in such a manner that it avoids the bottlenecks that come with the large volume of data.

Answering **RQ3** necessitates understanding the underlying distribution of users and their behaviour as a whole, and how the recommender systems and the framework behave when presented with the different contexts they will encounter in the real world. This will need us to identify and partition the

[1] implicit-feedback, as opposed to explicit feedback, is when the preferences of a user are inferred from available data, instead of the user giving explicit rating or rankings in the form of 5-star ratings or likes/dislikes.

available data to create different scenarios to validate the operationalization of the developed framework.

1.4. Thesis Objectives

To maintain the quality of the reusable research output, this thesis has a set of objectives that have been attempted to fulfil at every step of the process. These are:

1. User-centric perspective, *i.e. do not blindly use algorithms/techniques without thinking about the user*.
2. Explainable and highly configurable *i.e. can be modified easily for other application domains and systems*.
3. Extendable, *i.e. future researchers can enrich methods with new techniques easily*.
4. Well-developed, reproducible and reusable, *i.e.* ***Open Source***, ***Robust*** *(well coded and understandable, modular, easy to debug) and well documented*
5. Scalable, *i.e. usable for real-world large-scale problems*

The code is separated into two public repositories. This is done because the target audience and tasks of both projects are different. The first repository[2] deals with efficient preprocessing of the large-scale dataset we use, and then transfers it to a graph database. The second repository[3] contains (i) the Scala code and Docker files for scalable recommenders. (ii) The Python code for the Evolutionary Algorithm experiments and visualization tasks, along with the GPU-optimised Nearest Neighbour search algorithm described in Appendix C.

1.5. Thesis Structuring

The thesis report is structured to accommodate a broad audience. Knowledge relating to different theoretical and application domains is presented in an isolated manner. Sections that depend on the other have been linked across the document. External sources are provided to enrich the text and point outward when details fall outside the scope of the thesis. That being said, the thesis is organised as follows:

- Chapter 2 provides some background about the overarching themes of this research. Essential terminologies and motivations that are required to understand the thesis document in detail are provided here.
- Chapter 3 explains why and how we use the Music Listening Histories Dataset. There is also analysis of the data distribution of the same, and information on how we split the data into usable folds for conducting reliable experiments.
- Chapter 4 gives the background of the Collaborative Filtering strategy we will be using to generate recommendations. The chapter also gives an overview of the user-centric metrics we use.
- Chapter 5 gives more information about the Evolutionary Algorithms domain to the uninitiated. Furthermore, the evolutionary operators and the Evolutionary Algorithms we use are explained here. This chapter does not delve too deep into the theory of all chosen methods, and external resources are provided where necessary.
- Chapter 6 communicates our methodology for generating recommendations and the near-optimal trade-off front. This includes explaining the individual Recommender Systems we use and our motivations behind adopting them.
- Chapter 7 elaborates on the experimental design used to verify whether the recommendation framework can reliably generate a near-optimal front. This chapter also includes the essential details required to reproduce the results, such as the computational setup and hyperparameters used for the framework.

[2]Processing the Music Listening Histories Dataset: `https://github.com/sharwinbobde/MLHD-processing`
[3]Scalable Recommender Systems with Evolutionary Algorithms: `https://github.com/sharwinbobde/MLHD-insights`

- Chapter 8 presents the results of the experiments and provides visualizations to show how the recommendation framework behaves under the hood. It also gives concise explanations of the same.
- Chapter 9, gives an overview of the findings of the experiments and lists the contributions that were made in the process.
- Chapter 10 lists and explains the possible threats to the experimental study. It also presents some directions for future research in this niche combination of various domains.
- The Appendices provide elaborate results and visualizations, which have been summarised in an easily digestible format in the main storyline of the report. Appendix C details a research direction we took that did not eventually meet the main storyline. This involved enriching the listening history dataset with music features and finding the nearest neighbours for items/songs using a GPU optimised approximate Nearest Neighbour algorithm.

2 General Background

In this chapter, we will broadly look at the challenges that need to be overcome and some techniques we need to know to answer the research questions. We will first look at the problem from a data perspective and then move on towards how we feasibly develop the recommendation framework in question. The purpose of this chapter is to show, the importance of the problems we are trying to tackle, and the level of challenge we face in doing so. Take note that we will look at the concepts quite broadly, for more details, concepts and critical choices related to the data under consideration, look at Chapter 3, for Recommender Systems, Chapter 4 and Evolutionary Algorithms, Chapter 5.

2.1. Music Consumption

How a user consumes their music depends on a lot of factors. These factors influence recommendations by influencing the data which is used to develop and test the deployed recommendation systems (RS). The critical factors related to this thesis are listed below.

- The set of music services available in the geographical region; example, Spotify[1], YouTube[2], SoundCloud[3], LastFM[4] and Deezer[5] to name a few. Different combinations of services are available in different geographical regions because of varying laws across regions and the services' infrastructure for content distribution.
- Each service has, broadly speaking, a different content distribution mechanism. Services, the likes of Spotify, YouTube, Deezer, etc. rely on their own recommendation algorithm tailored by their teams. Therefore, the user experience and interaction patterns vary significantly across platforms.
- Users exploit different features with varying degrees on different platforms to optimise their experience [7].
- The service's priorities during content distribution.
- The geographical and sociocultural aspects of music relevance and feeling of relatedness.
- The user's interaction pattern based on their real-world state. This encompasses their type of work, lifestyle, and other factors that tell when and where they consume music [8, 9].

These factors have been taken into account while analysing the data we use and for constructing a good experimental setup for the recommendation framework. More about the same will be discussed in Chapter 3. For now, lets consider these factors for understanding biases.

[1]https://www.spotify.com/
[2]https://music.youtube.com/
[3]https://soundcloud.com/
[4]https://www.last.fm/
[5]https://www.deezer.com/en/

2.2. Sources of Biases and their Consequences

Jiawei Chen, *et al.* in a 2015 study explained the various biases in Recommender Systems extremely well [10]. The root problem of these biases is that most research that concerns recommender systems focuses on automating presenting items to users based on the observed data by making a model that fits user behaviour data. The problem with recommending fairly to all users is that the users in this data are not always representative of the true cross-section of the society. Certain users' behaviour has more influence on how the model fits the data. The study pointed out 7 major types of biases in recommendation systems, we mainly consider implicit user feedback in this study, so we are concerned only with the following biases:

- *Conformity Bias:* When users rate an item similarly to other users in the group or listen to an item frequently even if it goes against their personal taste.
- *Exposure Bias:* Users are only exposed to a subset of items, so the unobserved items do not always represent indifferent or negative preferences.
- *Position Bias:* Users tend to interact with items placed higher in a list more often regardless of relevance.
- *Inductive Bias:* Assumptions made by a recommendation model about the underlying data to learn a target function.
- *Popularity Bias:* Popular items are recommended more often than unpopular items regardless of relevance.
- *Unfairness:* Recommendation System systematically and unfairly discriminates against certain individuals or communities.

We leave out *Selection Bias* because we consider only implicit feedback for this study to demonstrate the framework for large volumes of data. In implicit feedback, users don't rate songs they like or dislike. Their preferences are implied (implicit) by what items they consume frequently. Therefore, Selection Bias does not come into the picture. The various biases form a *Feedback loop* or *Bias amplification loop*. This has been shown diagrammatically in Figure 2.1

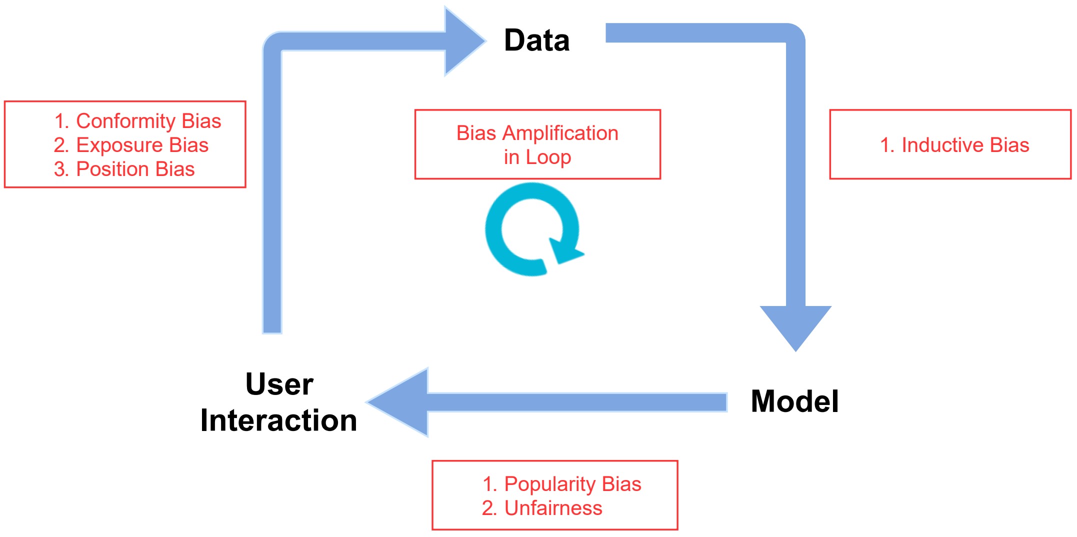


Figure 2.1: Bias feedback loop by Jiawei Chen, *et al.* without Selection Bias [10].

This bias in recommendations leads to *filter bubbles* and *echo chambers* [4, 11]. In the case of music recommendation the two terms have a specialized meanings. *Filter bubbles* restrict the user to expand their taste beyond a limit set of musical styles due to intentional personalization, that the user does not have control over. *Echo chambers* entail that the user gets recommendations from a limited set of musical styles which means the user's taste is not enriched by the system. This is also something the user does not have complete control over. Take note that musical *styles* are ***not*** musical *genres*, because genres can be classified in many ways and do not have a fixed hierarchy which has a global consensus, whereas styles are just qualitatively different to the human ear.

From the specialised meanings of the two terms, *filter bubbles* and *echo chambers*, the solution seems obvious: give the user more explicit control over their recommendations. This might be more easier said than done, because providing explainable and intuitive control over the recommender system, whose impact on the resulting recommendations is obvious and consistent across sociological and cultural differences is a difficult task.

2.3. User Interaction Data

We discussed the music consumption behaviour in Section 2.1. Now, lets see how that interaction information translates to data. Firstly, it is important to note that when a user interacts with an item, we can obtain more than just the $user \rightarrow item$ event. We know for music, items have artists, possible genre tags, community associations, language information, rich features of the music itself, and so on. Therefore, we can link a user to many nodes in a large knowledge graph with rich temporal information.

These networks, even across time slices/windows, can get very large and the interactions are dynamic in nature. Both these factors make processing the information an engineering challenge on top of a scientific one. Moreover, when we develop solutions for these problems in a development environment, they do not always reliably scale up in the production environment because of the different distribution of data and various bottlenecks that come with BigData. In this thesis, we will demonstrate our method on a rich dataset that poses real-world challenges with scale to show how well the system performs.

2.4. Recommendation Fusion

As discussed in Section 2.2, *Inductive Bias* arises due to assumptions made by a recommendation model about the underlying data. Different recommendation models make different types of assumptions, and have different pros and cons. In Chapters 4 and 6 we will see the type of recommenders we use in detail. For now, lets say that we use *diverse* recommender systems (RS) which prioritise a set of user-centric objectives differently. For instance, some RSs prioritize predicting what is relevant to a user, while another can focus on presenting novel/unexplored items. These different recommenders always try to provide a good user experience but have their pitfalls. No selected balance of these values is perfect for the entire user base. Users have dynamic preferences that can not always be anticipated. Therefore, we need a recommender fusion technique which can give the user/streaming service multiple options to select from.

We can achieve this by weighting different RSs with different proportions and then combining the generated recommendations. This gives us a final list of the benefits (and deficits) of the individual RSs together. The fusion is not trivial, the recommendation quality metrics (objectives) do not change linearly with the changes in model. Combinations of models can also lead to poor performance. Multiple combinations can reach the same objective fitness. Thus, fusing recommendations from multiple RSs is no trivial task.

2.5. Many-objective Optimization and Evolutionary Algorithms

Multi-objective optimization (MOO), also known as *multi-criteria optimization* or *Pareto optimization* is a mathematical domain which deals with optimizing multiple objectives (target functions) at the same time [12]. The name Pareto comes after the Italian economist and engineer *Vilfredo Pareto*. In essence, if a problem is non-trivial, there is no single set of parameters to a problem (in the parameter-space or solution-space) where all objectives are optimised at once (in the objective-space). Instead, for different parameters for the given problem in question, there will be an optimal set of solutions, where we see a trade-off for the different objectives.

An example is presented in Figure 2.2, where we see the solutions to a problem map to various objective values in the objective space, and the Pareto-optimal set of solutions are the the nondominated solutions. *Non-dominated* solutions are those for whom there do not exist any other solutions with a better fitness value for one objective function without worsening other objective functions. Or simply, a front where the solutions have the best trade-off for the specified objectives *i.e.* cost/fitness functions.

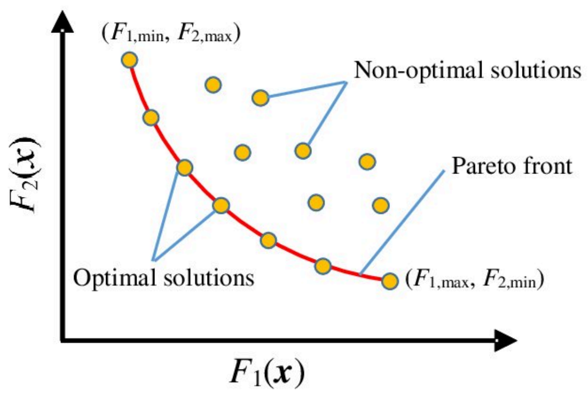
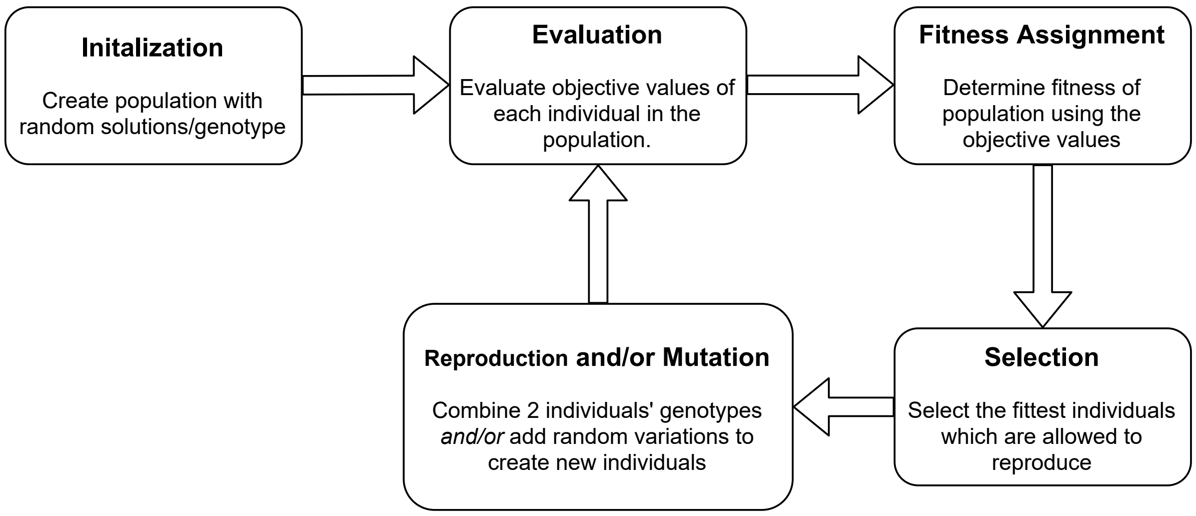


Figure 2.2: Pareto-optimal solutions example for a problem with 2 objectives, where both objectives have to be minimized [13].

While there are 2 ways of solving Multi-objective Optimization Problems (MOPs): *a priori* and *a posteriori methods* [14]. A priori methods require preference information about the various objectives can be expressed before we find the solutions. This leads the search to 'find a goal' we know should exist, and we eliminate other solutions we will not need. A posteriori methods, on the other hand, aim to give solutions that are Pareto-optimal. This is the direction we will be going in, because we need a set of solutions that encapsulates all near-optimal possibilities of maximising user-centric metrics.

Popular approaches to solving MOPs in an a posteriori manner computationally are (i) Mathematical programming programming and (ii) Evolutionary Algorithms. Mathematical programming consists of iterative algorithms which successively improve the quality of the solutions. Some commonly used methods under mathematical programming are ϵ-constant methods[15], branch and bound strategies [16], Normal Boundary Intersection (NBI) method [17, 18] and successive Pareto optimization [19].

Evolutionary Algorithms (EAs), more specifically *Multi-objective Evolutionary Algorithms*(MOEA) here, on the other hand, are general metaheuristic based optimization techniques that fall under guided randomized search algorithms [20]. EAs are family of more specialised search strategies, though this discussion is out of the scope of this thesis. EAs are inspired by biological evolution; where a candidate solutions start as random parameters for the problem and with increasing iterations (called generations) become progressively better. How this is done will be explained in greater detail in Chapter 5. In this study we will use EAs to solve ***Ma****ny-****o****bjective* ***E****volutionary* ***A****lgorithms* (MaOEA), which aims to optimize 3 or more objectives at the same time.



Initalization
Create population with random solutions/genotype
Evaluation
Evaluate objective values of each individual in the population.
Fitness Assignment
Determine fitness of population using the objective values
Selection
Select the fittest individuals which are allowed to reproduce
Reproduction and/or Mutation
Combine 2 individuals' genotypes *and/or* add random variations to create new individuals

Figure 2.3: General Framework for Evolutionary Algorithms (EAs) [20].

A simple organisation of a general EA is given in Figure 2.3. The variables/parameters/genotype of individuals in the simulated population are initialised with random entries to begin with. Then the objective values of each individual are computed *i.e.* how well each individual performs in the objective-space. This leads us to compute the fitness score for the entire population. We now come to the solution improvement part, where we have to select the fittest individuals in our population to allow them to go to the next generation. The selected individuals reproduce *i.e.* combine their genotypes to produce new individuals. The motivation behind this is to improve the solution by passing on *'good genes'* of two individuals to the next generation. Additionally, we also have a mutation operator which adds random changes to the genotype in small quantities. This is done to maintain genetic diversity in the population, but not so much that the good solutions will be lost. These various steps together lead to an incremental improvement of the population as the number of generations increases. This has the following benefits:

- We have to compute the fitness of a limited number of individuals every generation.
- We do not have to do an exhaustive search of the parameter-space. Finding solutions for high-dimensional problem space becomes more efficient.
- We can explore in multiple directions in the parameter-space.
- We can find multiple local optima.
- Stochastic objective functions are not a problem.
- Inherently parallelizable.

2.6. MaOEA for Recommender Systems: Previous Work

Lets see the research efforts that have been undertaken in this application domain in a chronological and categorical order. In 2014, Ning Yang, *et al.* a team of researchers and engineers, demonstrated a technique to recommend highway alignment strategies using MOEA [1]. They could provide multiple strategies for laying down highways with a trade-off between construction cost and environmental impact. Although this may seem tangential to our study, the core idea of the user-system interaction scheme remains the same: a user is presented multiple options and they choose the trade-off they desire and see how the the outcome would be. If they are unsatisfied, they select another outcome with minimal delay. This enables the user to explore how the trade-off feels to them and switch easily. In [1] the users were experts and knew about the domain and selected a trade-off based on domain knowledge, in our case the users would be the general public who select their trade-off based on their subjective preferences, mood or other contextual settings.

Coming back to the domain of Recommendation Systems, in 2015, Bingrui Geng, *et al.* developed an MOO based recommender system that diversified the output of a Collaborative Filtering model [21]. They used the Nondominated Neighbour Immune Algorithm (NNIA) to optimise Precision, Novelty, Diversity, and Congestion (recommendation uniformity). They validated the approach on MovieLens [22], Netflix [23] and Donation Dashboard [24] datasets. In 2019, Chonghuan Xu attempted to adapt the same system for BigData [25]. The system was validated on MovieLens [22], Book-Crossing dataset [26], and a real-world dataset with mobile shopping data. They used a map-reduce technique to perform Collaborative Filtering.

In 2016, Shanfeng Wang, *et al.* from an intelligent perception research lab demonstrated a framework to balance accuracy and novelty in Collaborative Filtering [3] to recommend more items from the long-tail distribution *i.e.* unexplored items. The framework selected top-K items from the items recommended to each user and fused the list with an unexplored (novel) item. They validated the system on the MovieLens 100k [22], Jester [27] and Netflix datasets [23]. They used MOEA/D [28] as their chosen EA for obtaining the front.

In 2018, Nour El Islem Karabadji, *et al.* a group of researchers from France and Algeria, attempted to diversify recommendations by diversifying the top-K users picked in Collaborative Filtering for generating recommendations thereafter [29]. They designed a method where they picked a Pareto-optimal set of users to go in the top-K for each user for whom they were generating recommendations. The objectives to optimise on in their case were user similarity and diversity. This method does have a serious foreseeable bottleneck for application on large systems where if we have millions of users, then the problem being optimised becomes more complicated and the fitness evaluation also takes longer.

In 2020, Xingjuan Cai, *et al.* demonstrated a recommendation system where they performed late-stage fusion of 3 different types of Collaborative Filtering approaches using MOEAs. This was done by finding the weight for weighing different CF approaches, using EAs and using the weights to rerank the recommendations. They aimed to find a trade-off between accuracy (recommendation-recall), novelty, diversity, and coverage. They used NSGA-III, RVEA, GREA, and SPEA2 as the Evolutionary Algorithms for doing the same. The system was tested using MovieLens 1M dataset [22].

All previous methods have their deficits, may it be problem encoding, foreseeable problems with operationalization, testing only on small datasets, only sticking to Collaborative Filtering approaches, or not validating the system properly with a well-constructed experimental design.

2.7. Distributed Computing

Processing real-world music listening data requires a tremendous amount of reliable computational infrastructure: scalable storage, CPUs, main memory, good I/O devices, and possibly GPUs. Doing all required tasks on one massive and expensive computer is not economically feasible for most businesses. The solution is to use several affordable computers, all of whom work together, to solve one big problem at a time by communicating with each other and solving small parts of the problem in parallel. This is known as *Distributed High-Performance Computing*.

Today, *Apache Spark*[6] (will be referred to simply as *Spark*) has become a popular option for processing BigData in a distributed manner [30]. Spark has a rich set of high-level programming tools which enable developers to process their data like SQL tables[7] using Spark *DataFrames*. With Spark, one can simply write a code in a development environment with limited computational resources, using a smaller representative dataset. Once the code is tested and needs to be deployed for production, it can be run on a compute cluster whose resources can be scaled on demand.

2.8. Performance Engineering

We discussed that dealing with large datasets is difficult and can be solved by adding more computational infrastructure and parallelizing the workload. Sadly, this is not enough to make the processing efficient enough to be truly *usable*. Simply using industry-standard techniques will not guarantee the efficient utilization of resources we already have. We therefore need another discipline under Computer Science to make the research output of this thesis oh higher quality: *Performance Engineering*. The objective of Performance Engineering is to improve the nonfunctional aspects of a software product for improving performance, *i.e.* decreasing computational latency, throughput, efficient memory usage, decreasing the computational complexity of workload, and so on. While this is not necessary to make a working solution, it makes the solution more easily adoptable and reduces infrastructure costs in the long term, while conserving energy and reducing a business' carbon footprint.

In the context of this thesis, we will look at the following aspects of the system throughout the development life-cycle to ensure high software quality:

- Reducing storage access frequency and latency.
- Conservatively using memory to eliminate the requirement for using memory-optimised compute infrastructure.
- Use best parallelization practices wherever applicable.
- Use GPUs for shorter time frames, to avoid lengthy CPU parallelized workloads if it makes economic sense.
- Improve the space and time complexity of algorithms.
- Identify future system bottlenecks.

While most of these aspects will not be apparent in the report, they can be spotted through various choices made during development which are well documented inline in the code[8].

[6]https://github.com/apache/spark

[7]SQL tables represent data present in a database in a columnar manner, where every field is a column and every record is a row.

[8]`https://github.com/sharwinbobde/MLHD-insights`

3 Datasets

In Section 2.2 we discussed the sources of bias which plague the present recommender systems and why it is important to give the user control over what they are recommended. We will always be limited to using observational data to derive predictions, so let us look at how we can do predict and recommend items for users while analysing the system well with the target of reducing biases in the recommendation.

We will first talk about the data we have available and how we will use it. Secondly, we will look at the distribution of users' behaviour in the same across various time slices. Then we will discuss how we split this data.

3.1. Background

Let us first discuss the details of the dataset that will be used in our study, along with the technologies and software products we use to properly use it to its full extent. Because this study aims to understand the reliability of the recommendation framework along with the generated recommendations, we have to make sure we use extremely rich information in a great volume to make sure our analysis has good foundations. While we could deal with datasets in a generic manner *i.e. dataset → train-test split → models → numbers*, we will not! Instead, we will enrich the organisation of an existing dataset to suit our experimental design, which takes the temporal effects of applying the framework to the provided data. This is also why we will be going in a nontraditional direction of using a dataset with temporal information (timestamps) as opposed to regular benchmarks such as the Million Songs Dataset [31]. This decision leads us to a path of rigorous system validation, across not only explicitly defined factor levels but also extra-musical factors.

3.1.1. MLHD

In 2017, Gabriel Vigliensoni and Ichiro Fujinaga aggregated the largest dataset for music listening user data that is publicly available[1], and called it the **M**usic **L**istening **H**istories **D**ataset (MLHD) [32]. The dataset dwarfed other datasets that were available at the time, with data from *583,000* users resulting in 27 Billion listening logs. Table 3.1 clearly shows the size comparison in numbers.

There is a more recent music session dataset called the *Music Streaming Sessions Dataset* released in 2018, maintained by Spotify's research and development wing [33]. This dataset is more suitable for analysing Machine Learning applications for a single session of sequential listens and is therefore not appropriate for this study because we want to analyse a recommender system/framework.

Music listening logs in MLHD are organised as user files and distributed as compressed archives, with around 1000 users, each tarball averaging to 1GiB. There are 576 such tarballs with one dummy file MLHD_386.tar. Uncompressed, each tarball results in a directory ≈ 18 GiB in size. The estimated size of the dataset files is 10,350 GiB or **10.11 TiB**.

The massive size is the reason we have chosen this dataset. This can give us a realistic scenario of real-world music logs with several active users; a scenario that will definitely push the recommender framework to its limits, so we can analyse its shortcomings.

[1] Music Listening Histories Dataset

Dataset name	Data source	users	logs	release year
Last.fm Dataset 1-K [34]	Lat.fm	1K	19M	2010
MusicMicro 11.11-09.12 [35]	Twitter	137K	600K	2013
Million Musical Tweets Dataset [36]	Twitter	215K	1M	2013
#nowplaying Music Dataset [37]	Twitter	4.2M	50M	2014
LFM-1B [38]	Last.fm	120K	1B	2016
MLHD [32]	Last.fm	**583K**	**27B**	**2017**

Table 3.1: Size comparison of publicly available music listening histories datasets.

LastFM

The user logs for MLHD are collected from Last.fm[2], a music recommendation website founded in 2002. The company has a *scrobbler service* which logs user's music listening activity (*scrobbles*). It does this not only within its ecosystem but also and outside of it using other media players and streaming services such as Spotify, YouTube. This encompasses 111 external music access tools Last.fm has supported in the past. MLHD has cleaned user logs *i.e.* only users with a minimum of 2 years of activity and at least 10 scrobbles a day are considered. Moreover, it is also cleaned to account for other time and logging inconsistencies.

These properties of the MLHD ensure three things: (i) the data is well sanitised, (ii) it does not have data from users that interact with an RS in a singular way, using a single service. Therefore, there are diverse users, and (iii) RS have to perform well on this diverse data without knowing its origin(which service), and the chances of overfitting to an interaction pattern to show *'better results'* should be lower.

MusicBrainz Identifier (MBID)

Each user file named with a user's *uuid*[3] in MLHD has the columns *timestamp, artist-MBID, release-MBID, recording-MBID*. Here the MBID refers to a uuid issued by the MusicBrainz project[4], under the MetaBrainz Foundation[5] [39]. The project aims to provide a reliable and unambiguous hierarchical schema for music identification. Thus, each MBID is unique and links to various entities that describe music such as artist, recordings, albums, and the music label. This information can be used for future research tasks that necessitate dataset enrichment.

3.1.2. Graph Database: ArangoDB

Let us take a look at the *Data Engineering* side of this project. When presented with the use of large-scale linked data for community-driven research, we have to take a lot of practical concerns into account; open-source software, scalable, ease of use, efficient memory and storage utilization, and of course strong database consistency properties. We go with *ArangoDB*[6] as a solution to managing our data storage, management and retrieval needs. AranagoDB is a performance-centric solution for graph-data management needs, whose feature set is quite broad and covers most of our needs better than other options like *Neo4J* or *OrientDB* [40].

ArangoDB uses *RocksDB*[7], a persistent key-value storage optimised for fast access, to store and retrieve the data at every database node efficiently in a distributed manner [41]. This combination of graph-data centric organisation and access of unstructured documents with key-value pairs, stored and managed in an efficient manner, enables a set of favourable features for solving our research problem of dealing with large-scale data. These beneficial features are as following:

- ArangoDB's data and indexes to be stored on disk and utilize memory conservatively.

[2] https://www.last.fm/
[3] universally unique identifier
[4] https://musicbrainz.org/doc/MusicBrainz_Identifier
[5] https://metabrainz.org/
[6] https://www.arangodb.com/
[7] https://rocksdb.org/

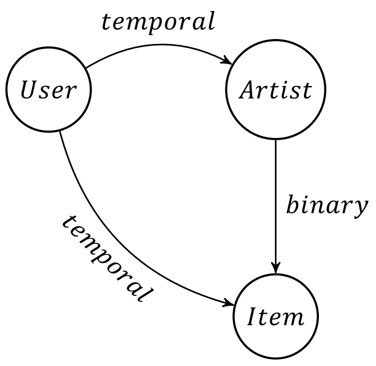


Figure 3.1: Explanation graph extracted from MLHD's user scrobbles.

- Concurrent read-write access.
- Supports large datasets.
- Data is stored in a log-structured tree and compacted at predictable intervals.
- Highly configurable for different workloads and compute infrastructure.
- Supports synchronous and asynchronous replication.
- Steady query performance on both read and write operations.
- Reasonably small storage requirement for database files due to compaction [42].

3.2. MLHD Graph Representation

Now, we shall discuss how we use the MLHD to create a graph representation that we store in the graph database. Figure 3.1 shows the information we have in each user listening history log-file , *scrobble*, and how we can structure it for storing it in ArangoDB. Each scrobble has the user's uuid which forms the user's node.

The $user \rightarrow artist$ and $user \rightarrow item$ link the user to artist uuid and recording uuid, respectively. Both relations also have timestamp information. In this study, we aggregate the timestamps to reveal *how many times a song was listened to (for more than 30 s) in a year*. We call this *listen count* and denote this by the key *year_20XX* in ArangoDB. The data in MLHD is available from the year 2005 until 2013. We also store a binary connection $artist \rightarrow item$ to denote a recording is associated with the particular artist. The presence of the edge in the graph means the relation exists, otherwise not.

To reiterate what was said in Chapter 1, Introduction, the code for processing MLHD parts and transferring them to ArangoDB are accessible through an open-source repository[8]. This method can be adapted for other research endeavours easily, and the data can be easily queried or processed for a wide array of tasks.

3.2.1. Critical Choice

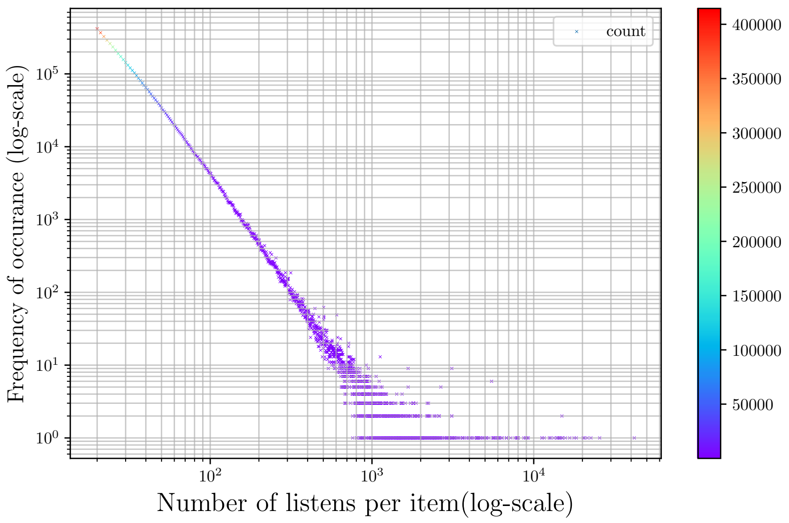
We had to make some critical choices while constructing this graph to make sure our experiments are feasible. These choices affect the storage and computing time required, but more importantly also the quality of recommendations. Taking a different sampling strategy will change the results of this study. For example, taking all interactions, including single listens to songs, will make recommendations much better because we gain information about what the user does not like. Although, doing this will entail a considerable investment in compute and storage resources, in addition to increased latency in generating recommendations.

To explain this visually, see Figure 3.2a that shows a log-log scaled plot of *listen count* vs frequency of occurrence. The number of times a listen count is observed takes form of a *Power Law distribution* [43]. This means a linear-linear scaled graph will show a hyperbola. If we have two listen counts l and

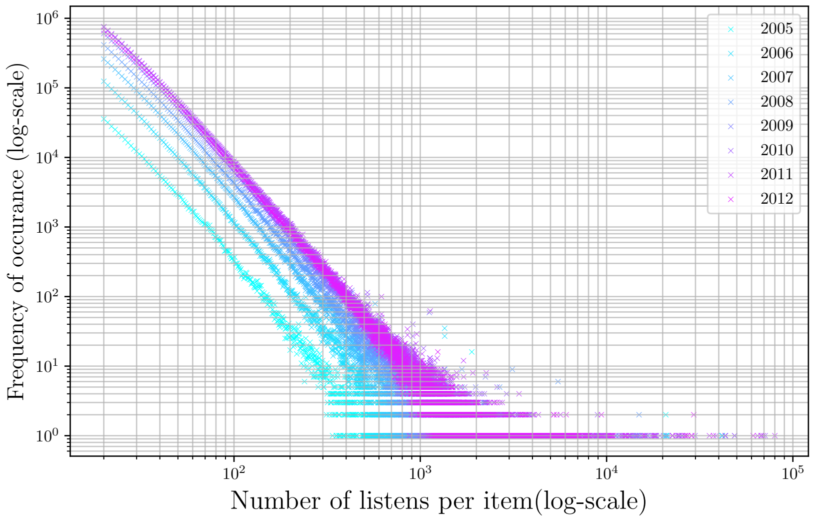
[8]https://github.com/sharwinbobde/MLHD-processing

l', where $l' \ll l$, then taking links that represent data that have l' listen counts will have on average, drastically more observations than l. We therefore cannot record interactions with very few listen counts in the graph because they will increase the size of the dataset (without compression) drastically.

We found the listen count threshold of **20** to be appropriate for our demonstration. This means if a user hears a song 20 times or more, each time for more than 30s, then only will it be recorded in the graph.



(a) Example, listen-count frequency of occurrence for single year (here 2008 as an example)



(b) Comparison of the same across years

Figure 3.2: Listen-count frequencies in the dataset. Note (i) log scaling for both axes. (ii) The distribution is a **Power Law distribution**[43], thus, in our case *users listen to songs a few times hypnotically more than many times, and the majority of the data is about the songs, user doesn't listen often*. (iii) Amount of data increases every year but the nature remains same.

3.2.2. Database in Numbers

Table 3.2 shows the comparison of the size of MLHD's raw text data *v.s.* ArangoDB's compactly stored graph data. Note that the data that needs to be processed now has gone from a Tebibyte[9] scale to a

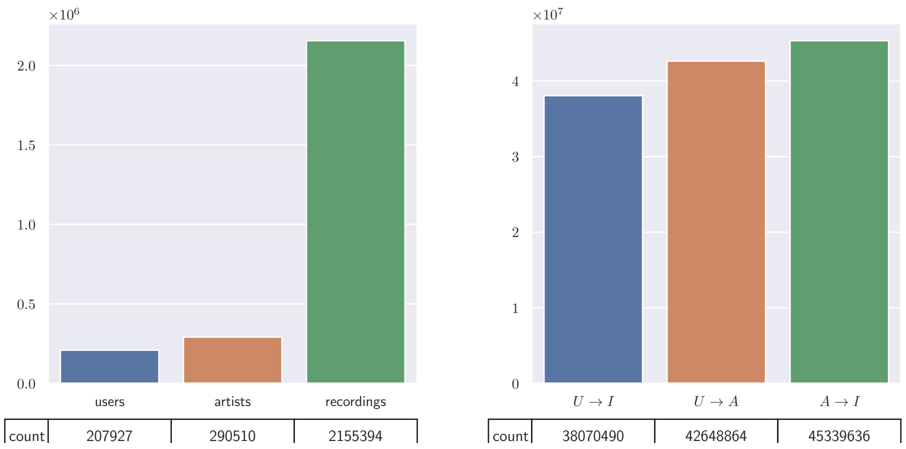
[9] 1024 Gibibyte. Do not confuse with Terabyte which is 1000 Gigabyte.

Gibibyte scale. The reduction in storage requirement means we do not need to rely on cloud providers and network bottlenecks. We can store the dataset on fast and local SSDs[10]. One con of the size reduction is that accessing large edge collections in ArangoDB requires great memory overhead. How we deal with this is detailed in Chapter 7, Experiments.

Number of MLHD parts	Estimated dataset uncompressed size (in ***TiB***)	ArangoDB database size (in ***GiB***)
26	0.46	4.3
101	1.78	14.3
201	**3.53**	**20.3**
301	5.29	28.2
400	7.03	37.1
575	10.11	51.8

Table 3.2: MLHD (user logs text) estimated size versus ArangoDB database files size.

We use only 201 parts of MLHD to make our experiments feasible. The reasoning for doing this even though we can store the entire dataset has been made clear in Section 7.2.1 of Chapter 7, Experiments.



(a) Graph node counts (b) Graph edge counts

Figure 3.3: Barcharts for the aggregated dataset as a graph representation *(for 201 parts of MLHD, of total 575 parts)*

In Figure 3.3 we can see the number of nodes and edges present in the complete graph representation of MLHD. This graph consists of information from the years 2005 to 2013 in the same set of node and edge collections.

[10]Solid State Drives

3.3. User Activity Distributions

Now, we analyse the information we have. In an earlier section, Section 3.2.1, we talked about the nature of the *listen counts* we see in Figure 3.2a. This entails that MLHD, or scrobbles in general, have more information about the song people listen to less often. It is also important to consider that just because a song is played more than 10^4 times in a year does not mean a user necessarily likes it. These could be edge cases in the data where the logs are noisy or the user kept a song on repeat or played a song in a public setting such as a bar or a gym. In Figure 3.2b, we see the same data for different years. We see that year-on-year number of recorded $user \rightarrow item$ interactions increases.

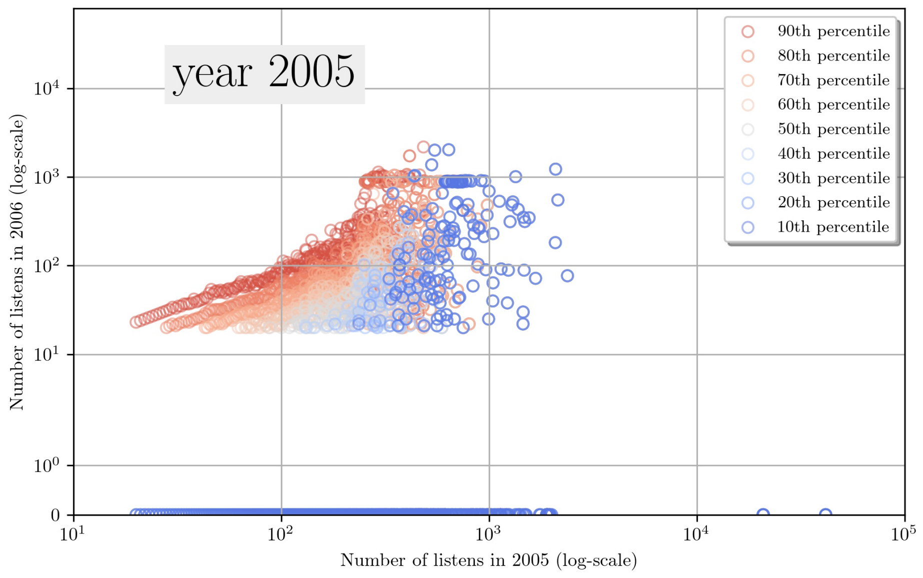
For verifying the $users \rightarrow item$ interactions represent a genuine interest in the items, we decided to make more detailed visualizations. We decided to investigate how *listen counts* in one year are related to the following year. For this we calculate the percentile distributions of the following year's *listen count i.e.* for each unique listen count in year y, what is the p^{th} percentile listen count in year $y+1$ for the same user. Because this is done on a really large edge collection, this needs to be done in an efficient manner[11]. A small mathematical brush-up, the p^{th} percentile value, where $0 > p \geq 100$ is the smallest value in the data for which p percent of the data is less than the value. For example, at 60^{th} percentile value, 60% of the quantity of data is below the value.

The animation in Figure 3.4 shows the percentile value distribution over the years. The lower percentiles are placed over the higher percentiles, otherwise a lot of critical information is lost. We see that a general pattern across different years is that, for any user, the increased listen count in year y is directly correlated with increased listen count in year $y+1$. One striking observation is that every year, after a certain threshold, the lower percentiles strikingly jump up in the distribution and do not follow the trend. This effect becomes visually stronger year by year. In many cases, the 10th percentile (90% of the observations) are uncharacteristically high.

We can only speculate why this happens because we do not have more information to prove any hypothesis. We speculate that this happens due to users' listening behaviour changing because of the extra-musical factors.

There is no way to concretely prove the above speculation. The only thing for certain is that the listening distribution (observed listening behaviour) changes across the years. As this study aims to check the reliability of the recommendation framework, we need to check if these varying distributions across years affect the performance of the RSs or the quality of the generated front by the EAs.

[11] Calculated using the *approximate quantiles* function (approxQuantile) in Spark [44]. The algorithm has a deterministic bound. If the Spark DataFrame has N elements and if we request the percentile at probability p up to error err, then the algorithm will return a sample x from the DataFrame so that the exact rank of x is close to $(p * N)$. More precisely, $floor((p - err) \cdot N) \leq rank(x) \leq ceil((p + err) \cdot N)$. The Spark implementation is an improvement over the Greenwald-Khanna algorithm [45] (with some speed optimization).



.

Figure 3.4: **[Embedded Animation]** Percentile distribution of favourability of items/songs the following year, depending on listen counts across all $user \rightarrow item$ interactions. To be interpreted as, *"If a song is heard 'X' times in a year, It is heard for 'Y' times the following year"*. Note (i) To see the animation open the PDF in native viewer and not on a web browser (ii) To interact with controls enable PDF forms for the PDF viewer. (iii) If you cannot manage to run the animation then individual plots are present in Appendix A.

3.4. Dataset Division

The analysis in the previous section and Chapter 2, Background, motivate our need to take temporal factors into account. We will now suggest our method of dividing the MLHD for the recommendation framework under study for fusing the outputs of specialised RSs using EAs. We are splitting the dataset according to the year information of the scrobble timestamps. To give a real-world perspective, this is equivalent to making recommendations on 31st December at 23:59:59. with data being collected from 1st January at 00:00:00 of the same year.

This manner of temporal split presents more real-world edge cases that require good performance in production. The edge cases which might lead to fewer $user \rightarrow item$ interactions can be numerous and unforeseeable. A non-exhaustive list of examples to show how this might occur is here:

- A user uses and has linked a Last.fm supported service which most often recommendations items the user plays on replay. The user might not have linked other services which give diverse recommendations.
- A user joins Last.fm later on in a year, perhaps in November, and does not have many $user \rightarrow item$ interactions by Dec 31st of the year (when we are making the recommendations).
- A user only uses one streaming service with a particular music delivery strategy.
- A user might simply enjoy listening to a niche genre of music to repeat even if they are exposed to other music. This niche genre might not have the recording uuids from MetaBrainz.

The data we verify on, be it nodes such as users, artists and items, or edges between them, needs to be fairly sampled across the users. Non-mainstream users who show more exploratory behaviour will have less data in the train and test sets and give poor recommendations to these users [46, 47]. We organise the dataset in a manner to make user-centric metrics computed on the recommended lists reflect a near-average perceived experienced over all users. We will consider the following points to make sure the recommendation quality metrics calculated on these dataset slices are as close as possible to the cross-section of real-world users.

Year	$train$	$test_{RS}$	$test_{EA}$
2005	**4949**	**1599**	**1621**
2006	14126	4613	4579
2007	28041	9206	9286
2008	**45177**	**15044**	**14977**
2009	64993	21533	21552
2010	84456	28115	28106
2011	93772	31222	31210
2012	**91807**	**30536**	**30510**

Table 3.3: User counts, across years, for membership in dataset splits $train$, $test_{RS}$ and $test_{EA}$.

1. Test dataset(s) is large enough in proportion to training data.
2. Minimize the chances of overfitting while optimizing on the test data distribution in any manner.
3. Results do not depend on which users or interactions are picked and placed in a set.
4. Dataset division does not entail non-uniform sampling of data. For example, under-sampling minority users.

To make sure every year's data split has valid users, we first find *subscribed users* in a year *i.e.* users with *listen counts* > 0. Then we will sample the users and split them into train and two test sets, $test_{RS}$ and $test_{EA}$, with the percentages *60%, 20%, 20%* respectively. Table 3.3 shows the number of users in each split of the dataset by year.

The choice of two test sets $test_{RS}$ and $test_{EA}$ makes sure our results are not affected by the process of tuning the RSs or the EAs. The set $test_{RS}$ can be used to test that the RSs scale well with size and perform as expected in terms of user-centric metrics. We can decide hyperparameters for the EA, such as population size and the number of generations, and generate the Pareto-set using The RSs' output on $test_{RS}$. The second test set $test_{EA}$ is used to make sure that the generated front's quality is not affected by the users or interactions in a particular test set.

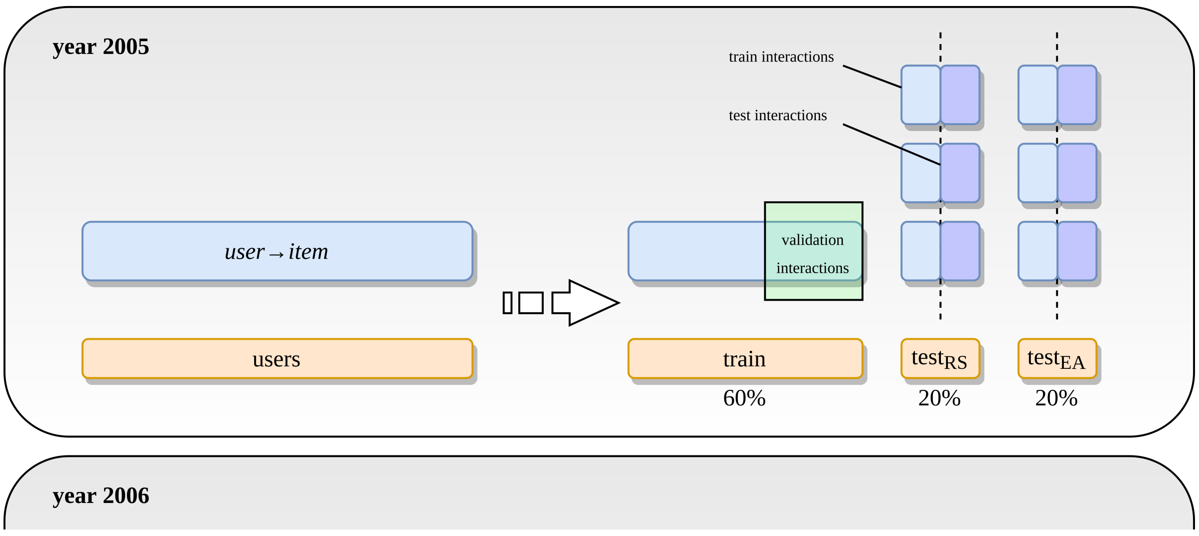


Figure 3.5: Illustration explaining the division of MLHD by year. Users are split into $train$, $test_{RS}$ and $test_{EA}$ sets. The $user \rightarrow item$ interactions for each test set are split into 50%-50% randomly, thrice. A part of the train-interactions is used as validation interactions to train RSs in a traditional fashion.

We will then split the $user \rightarrow item$ interactions for each user in that particular $test_X$ set in 50%-50% randomly into test-test and test-train interaction sets for it. We will do this thrice and denote these interaction-sets by δ_i, where $i \in \{1, 2, 3\}$. This strategy is adopted to make sure our user-centric metrics do not depend on how we sample interactions from available data and is illustrated by Figure 3.5.

Sampling users first and adding interactions, instead of directly $user \rightarrow item$, interactions will ensure that we do not undersample the activity of users who have less number of interactions in a year.

4 Recommender Systems

In this chapter, we see the concepts required to understand the inner workings of the Recommender Systems we will use later. Moreover, detailed information on the user-centric metrics we use as objectives to optimise is also provided. We also give a high-level interpretation of these metrics to make them easy to understand for people who might not be familiar with this domain.

4.1. Background

Let us see some background about the concepts that help us build the Recommender Systems. These techniques help us make scalable recommenders and by understanding them we uncover any underlying sources of threats that may affect the study.

4.1.1. Collaborative Filtering

Collaborative Filtering (CF) has proved to be a successful and well-researched method for building Recommender Systems [48]. CF makes the assumption that if two users A and B interact with items similarly, then they will act similarly in the future. For instance, if the two users' interests and interaction patterns overlap extremely well in terms of their music taste and consumption, then it would be natural to assume A would like to hear what B has explored but they themselves have not. In essence, CF systems look at the interaction between users and/or items to filter similar users/items and recommend items that a user has not heard before.

In the context of the thesis, it is important to know that there are 3 types of CF. Firstly, *Memory-based CF* which maintain a full matrix of interactions and find similar communities to recommend items. These become problematic when the number of users and items increase because then you need to maintain a really large sparse matrix. Secondly, *Model-based CF* use the interaction data to estimate behaviour or learn a model to make predictions. The model can be any mathematical construct or a parametric model that can predict interaction information such as rating data, listening counts, etc. without storing all the interactions at once. Thirdly, *Hybrid CF* techniques combine regular CF with content-based recommenders to recommend items, not only based on observed interactions, but also based on features of the content that is being recommended, such as language for movies, genres for music, and tags for GIFs[1].

In this thesis, we are mostly concerned with a subset of Model-based CF techniques called *Matrix Factorization*. These techniques try to approximate a target Y, which can be ratings, listens, views etc. as a product of two smaller matrices [49]. Equation 4.1 shows the approximation task. Here let's consider $U \in \mathbb{R}^{I\times K}$ is a matrix storing the latent factors for I, users and $M \in \mathbb{R}^{K\times J}$ is a matrix storing the latent factors for J, items.

$$U^{I\times K} M^{K\times J} \approx Y \tag{4.1}$$

[1]Graphics Interchange Format (GIF), an image format than can show animation or a series of images.

4.1.2. Alternating Least-Squares

Alternating Least Squares (ALS) is an iterative form to perform large-scale Matrix Factorization [50]. Spark has its own implementation of ALS which enables performing Collaborative Filtering in a distributed computing scenario. The overview of the algorithm from [50] is as follows:

- **Step 1.** Initialize matrix M assigning the average rating for an item as the first feature and small random numbers for the remaining entries.
- **Step 2.** Fix M, solve for U by minimizing the regularised objective function.
- **Step 3.** Fix U, solve for M by minimizing the objective function similarly.
- **Step 4.** Repeat Steps 2 and 3 until the stopping criterion is satisfied.

ALS uses weighted-λ-regularization regularised update to avoid overfitting on the new updates. The choice is empirically proven to avoid overfitting the test data when the number of (latent) features or the number of iterations is increased.

The Spark implementation enables block-wise[2] iterative updates using a modified ALS approach for interactions that give implicit feedback [51, 52]. This blocked implementation, groups the users and items into two blocks, and reduces the communication required for computing the result. It does this by only sending one copy of each user vector to each item block on each iteration, and only for the items' blocks that require a particular user's feature vector. This is done by precomputing which blocks need the update by seeing the $user \to item$ links (which blocks of items it will contribute to) and $item \to user$ links information (which of the feature vectors it receives from each user block it will depend on). This allows us to send only an array of feature vectors between each user block and product block, and have the item block find the users' ratings and update the items' features based on these exchanged messages.

4.2. User-centric Metrics

To optimise on how a recommender performs from a user's perspective, we need quantitative metrics that reflect an approximation of the quality of recommendations. For this study, we use the following user-centric recommender performance metrics: Mean Average Recall (*MAR*), Coverage (*Cov*), Novelty (*Nov*) and a modified Personalization (*modPers*) metric [53, 54, 55, 56]. In the implementation[3] of calculating these metrics, a great focus is given to performance to make sure the fitness evaluation for the EA takes less time.

To make explanation of the metrics clearer, we will be using a set of standard notations. U is the set of all users, R is a recommended list, $|\cdot|$ is the cardinality of a set, and $\cap$ is the intersection of two sets.

4.2.1. Mean Average Recall

$MAR@k$ is the average recall over all users $u \in U$ in their recommendation lists R_u of length k. It tells on average how many entries present in a recommendation list of a particular size k are actually heard by the user. Here, T_u denotes the items actually heard by the user u.

$$MAR@k = \frac{1}{|U|} \sum_{u \in U} \frac{R_u \cap T_u}{|T_u|} \tag{4.2}$$

Although this metric is good for analysing users with different behavioural patterns and listening frequencies, it is susceptible to biases. For example, finding a recommended item belonging to the user's items in known relevant items in the test set can overfit on how the items were presented/recommended to the user. This is a result of the bias feedback loop in recommendations [10].

[2]blocks (partitions) of a Spark DataFrame
[3]Metrics Evaluation code available here.

4.2.2. Coverage

$Cov@k$ is the ratio of length of the set of unique item predictions made, R_u of length k, for all users R_U to the catalogue Cat set.

$$Cov@k = \frac{R_U \cap Cat}{|Cat|} \tag{4.3}$$

This metric informs how well a recommender can serve the available items in the available catalogue. From a user perspective, it informs if they have access to a broad set of recommendations or does the recommendations instead potentially pigeonhole the user by only recommending a small portion of the catalog.

4.2.3. Novelty

In this study, Novelty, $Nov@k$, is the proportion of items recommended to a user that comes from the long tail distribution. Here the metric is the ratio is the number of items in R_u that have their total listens below the 66^{th} percentile value Γ in the catalogue, to the length of the recommendation list k, averaged over all users $u \subset U$. Simply said, on average, how many songs recommended to the user come from the long tail distribution: unexplored songs or songs that are new and do not have too many listens yet.

$$Nov@k = \frac{1}{|U|} \sum_{u \in U} \frac{R_u \cap \Gamma}{k} \tag{4.4}$$

4.2.4. Modified Personalization

$modPers@k$ in this study is a modified version of the Personalization/Diversity metric which tells the recommendation similarity across users. The modifications are made to *(i)* consider the edge cases caused by the recommender not being able to generate recommendations for certain problematic users (with less listening data) and *(ii)* to make the computation of this metric more space and compute efficient and parallelizable.

The original description of the metric Personalization $Pers@k$ is as follows, where R_i is the 2D matrix showing the binary item co-occurrence for users represented by rows and columns. $s(m, n)$ is the cosine similarity of lists of items recommended m and n for 2 different users.

$$Pers = 1 - \left(\frac{\sum_{m,n \in R_i, m \neq n} s(m,n)}{\frac{1}{2}|R_i|(R_i - 1)} \right) \tag{4.5}$$

We modify this definition to calculate it with $modPers@k$ to compute the same for users whose $|R_u| \geqslant k$ and extract the list only till top k; we denote these users as $U_{|R_u| \geqslant k}$. The cosine similarity measure also becomes simpler to compute in this case, and the mathematics reduces to the following.

$$modPers@k = 1 - \left(\frac{1}{U_{|R_u| \geqslant k}} \sum_{m,n \in R_{U_{|R_u| \geqslant k}}} \frac{m \cap n}{k} \right) \tag{4.6}$$

Here we only need to compute the similarity values for an upper triangular matrix of users and the computation for each pair of users can be parallelized. Moreover, we store the similarity values in a space-efficient 1D upper-triangular matrix representation. Numba[4] has been used to make the performance optimisation for this metric. Numba is an open-source just-in-time (JIT) compiler for Python, which converts Python code snippets to extremely fast machine code and even supports simpler parallelization, which is again close to machine code [57].

[4]http://numba.pydata.org/

5 Evolutionary Algorithms

This chapter is intended to give sufficient background specifically about Evolutionary Algorithms to understand the subsequent chapters. Although the motivations to use EAs have been made clear in Chapter 2, it lacks certain nuances as to the true complexity of MOO problems. This chapter will also detail individual EAs and solution quality indicators at length, which will be simply be referenced in the following chapters for better knowledge organisation.

5.1. Pareto-Optimal Solutions

Pareto-optimality in Multi-Objective Optimization has a specific focus on optimizing the decision variable space (chosen objectives) for real-world problems. These are optimal solutions over the set of given objectives when there is no solution that performs better at all given objectives at the same time. Simply said, the set of Pareto-optimal solutions, which makes a Pareto front, is a boundary beyond which there are no solutions (trade-offs) where an objective can be improved without sacrificing the other.

Some important considerations are which make MOO problems difficult are listed here:

1. We do not necessarily know the shape of the Pareto front before optimizing.
2. We do not always know the extent to which every single objective can be improved.
3. We can not foresee if objectives improve/worsen in a convex manner when making changes to the parameter-space.
4. We can not foresee if objectives improve/worsen in a proportional manner with respect to each other *i.e.* objectives can be influenced by factors external to the encoded problem (parameter-space).
5. Visualization of the Pareto front generated by an EA for qualitative analysis is difficult.

5.1.1. Definitions and Notations

For our study, these points are incredibly important to take into account while analysing the quality of the resulting solutions at the end of the EA pipeline. This will become more clear in Chapter 8, Results.

To understand further the material we will need the definition of *domination* [58]. Consider we want to goal of optimization is given by equation 5.1, where $\vec{x}$ is a real-valued individual in the population. The function $f(\vec{x})$ consists of k individual objective functions.

$$maximise\ f(\vec{x}) = (f_1(\vec{x}), f_2(\vec{x}), ..., f_k(\vec{x})) \tag{5.1}$$

Given two decision vectors $\vec{a}$ (or a) and $\vec{b}$ (or b), there are these possible outcomes for the fitness relation for MOO problems (MOPs) besides complete dominance:

- $f(\vec{a}) > f(\vec{b})$, when all objectives of $\vec{a}$ are better than that of $\vec{b}$.
- $f(\vec{b}) > f(\vec{a})$, when all objectives of $\vec{b}$ are better than that of $\vec{a}$.

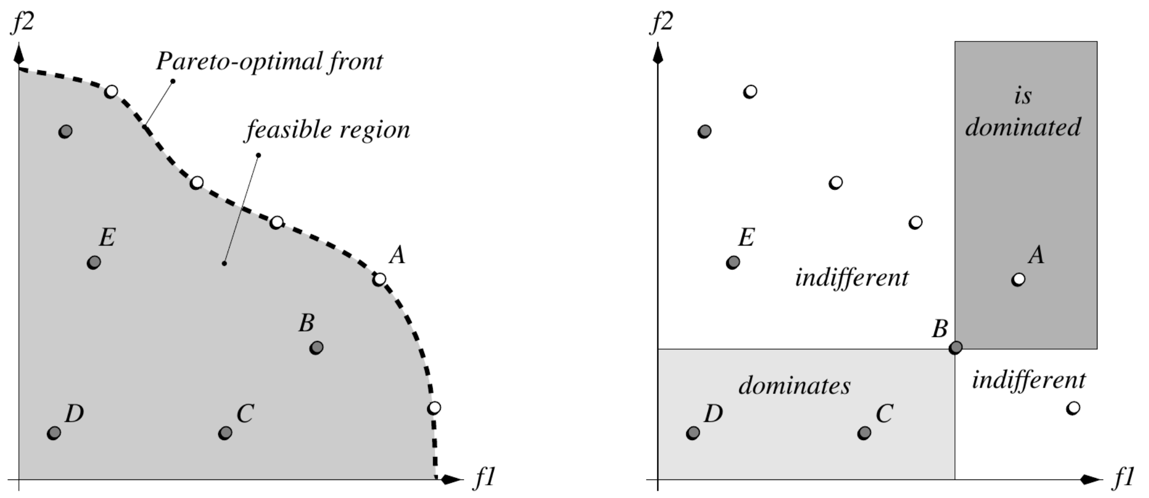


Figure 5.1: Illustration of (i) concept of Pareto-optimality in objective space (left) and (ii) the dominance relations of solutions in objective space (right) [58]

- $f(\vec{a}) \geq f(\vec{b})$, when all objectives of $\vec{a}$ are equal to or better than that of $\vec{b}$.
- $f(\vec{b}) \geq f(\vec{a})$, when all objectives of $\vec{b}$ are equal to or better than that of $\vec{a}$.
- $f(\vec{a}) \geq f(\vec{b}) \wedge f(\vec{b}) \ngeq f(\vec{a})$, when all objectives of $\vec{a}$ are indifferent to that of $\vec{b}$.

These cases can be seen visually in Figure 5.1 where we have an example of a two-objective MOP. We use special notations to denote these cases for convenience. These notations are given in 5.2.

$$\begin{array}{lll} \mathbf{a} \succ \mathbf{b} & (a \text{ (Pareto) dominates } b) & \text{iff } f(\vec{a}) > f(\vec{b}) \\ \mathbf{a} \succeq \mathbf{b} & (a \text{ weakly (Pareto) dominates } b) & \text{iff } f(\vec{a}) \geq f(\vec{b}) \\ \mathbf{a} \sim \mathbf{b} & (a \text{ is indifferent to } b) & \text{iff } f(\vec{a}) \ngeq f(\vec{b}) \end{array} \tag{5.2}$$

Lets wrap it all together in a simple language. Firstly, a dominates b *i.e.* $a \succ b$, if a performs better on all objectives than b. Secondly, a weakly dominates b *i.e.* $a \succeq b$, if a performs better on one or more objectives than b, but others do not worsen. Lastly, a is indifferent to b *i.e.* $a \sim b$, if a or b perform better on some objectives while worsening other objectives (make a trade-off).

5.2. MaOEA Considerations

While using Evolutionary Algorithms for MaOO (objectives > 3) there are some points of concern that need to be kept in mind before strategizing the implementation(s).

1. *High dimensionality*: Adding more objectives increases the dimensionality of the objective-space. This means a greater proportion of solutions found become nondominated. Thus we need a greater set of solutions to find the true Pareto efficient set of solutions, or the selection pressure needs to be higher to find a near-optimal representation of the Pareto front.
2. *Diversity*: While we can find solutions that are Pareto-optimal, another layer of complication is that the set of solutions we obtain have to be well distributed over the front. Having many solutions with trade-off over a few of the objectives that are easy to search, are not the same as a well-distributed set of solutions.
3. *Recombination operation*: in high-dimensional space recombination of two solutions that are fit but distant in the objective-space will result in a new distant solution. Therefore, recombination needs to be done in a more controlled manner for steadier evolution of solutions.

4. *Representation of trade-off surface*: with increasing dimensionality, exponentially more points are needed to represent the surface. This necessitates a larger population or smarter approximations of the front.
5. *Performance metrics*: when the number of objectives increase, we have to invest more compute time to calculate the value of objectives. Depending on the problem we aim to solve and the objectives this can result in exponential increase. These values are used for algorithm performance, this causes a lot of latency for the entire algorithm. For instance, hypervolume used to see the normalised area dominated by the Pareto front takes exponentially more computations.

These aspects make the tuning of EAs difficult to solve and analyse MOPs in a feasible amount of time. Therefore, one must not consider that we can simply throw an EA at any problem and can solve it effectively without much human input.

5.3. EA Operators

Evolutionary Algorithms use operators such as *crossover*, *mutation* and *selection* [59, 60]. These operators are used to improve the population quality over a number of generations. They induce the evolutionary nature of the algorithms and their selection and tuning are key to the solution convergence. Let us discuss what these operators do and talk a little about how they tie into this study.

5.3.1. Crossover

The crossover operator is responsible for the recombination of two parent genotypes by exchanging part of their one parent's chromosome with the other's to create a new offspring/solution. Crossover boils down to being an extensive local search technique over multiple generations. This is because the controlled crossing of two fit parents in a well-encoded EA problem should result in a good solution.

SBX Operator

Simulated Binary Crossover (SBX) for continuous real-valued search space is a very popular crossover operator[61]. The authors of the EAs we use, NSGA-III and SPEA2, themselves recommend the SBX operator [62, 63]. The motivation of the SBX approach is driven by the success of the binary crossover operators. It works well on problems having multiple optima and problems without known bounds on the real values. Although better, this operator is considerably computationally expensive compared to its counterparts such as linear crossover or blend crossover.

It uses a spread factor, β, which denotes the ratio of the difference between children and parents' genes. With $\beta > 1$ leading to expansive search and $\beta < 1$ leading to contracting search; $\beta = 1$ leads to no change. Both cases result in modifying the gene by sampling a value from two different distributions which direct the search, and this is followed by making two offsprings.

5.3.2. Mutation

The mutation operator provides a mechanism for global exploration by adding random variations to offsprings. Adding slight variations to the chromosome results in solutions that are in a solution-subspace which might be less explored. Without controlled mutation, a population cannot explore the solution space effectively. Both crossover and mutation will provide the diversity for new solutions. Mutation provides better diversity, although well-distanced solutions in the solution space may drive the population away from converged characteristics.

Polynomial Mutation Operator

We used the Polynomial Mutation [64] operator, suggested by Kalyanmoy Deb and Samir Agrawal, for two of our chosen EAs in this study. We used the literature and recommended settings from other studies to select this operator as it works well for real-valued problems/genotypes. In this operator, a polynomial probability distribution is used to perturb a solution in a parent's vicinity. The probability distribution in the neighbourhood of a variable value is adjusted so that no value outside the specified range $[a, b]$ is created by the mutation operator. This operator is computationally more expensive than random mutation but performs better.

It has two parameters that need to be tuned, namely, the probability of mutation and the distribution index. The probability is usually set as $1/n$, where n is the number of real-valued variables used for encoding the problem. The general research practice is to set the distribution index to 20 [65, 66].

5.3.3. Selection

The selection operator has 2 roles: (i) selecting fit solutions to proceed to the next generation, and (ii) providing a driving force or *selection pressure* which forces convergence of the population. Thus, this operator enables the population to evolve towards a goal. In this study, the selection mechanism varies depending on the chosen algorithm. This is because the selection is driven by the mathematical motivations of each algorithm. This will be discussed in Section 5.4 when we talk about the EAs relevant to this study.

5.4. EAs Used

In Section 2.6, of chapter Background, we saw different algorithms were used by different research groups in their research endeavours. While many chose specific algorithms because of their perceived strength for the problems they were solving, many others simply picked a set of different algorithms. We want to see how different EAs perform against each other, but also make sure we use a diverse set of EAs while keeping the number of experiments limited. We decided to adopt the following EAs because they are well used in the literature for comparison, which aim to solve the problem of MaOO using different mathematical motivations.

To clarify, the intention of this section is not to give exhaustive information about each algorithm, but rather to give a peek into the theory which tells how these algorithms differ from each other. In Chapter 8, Results, we will analyse if these differences in algorithms make any significant changes to the recommendation framework under study; *they don't*[1].

5.4.1. NSGA-III

The Nondominated Sorting Genetic Algorithm III (NSGA-III) [62], is an improvement over its predecessor, NSGA-II [67]. Both algorithms focus on effectively preserving elites (well-performing individuals) in every generation. NSGA-III has a strict focus on maintaining diversity in the population, which is done by supplying and adaptively updating well-spaced reference points. NSGA-II used a *crowding distance*[2] to preserve niche solutions across generations. The details of the crowding distance sadly fall outside the scope of this thesis, please take a look at [67] for the same. A detailed discussion of the NSGA-III falls beyond the scope of this thesis, but broadly speaking, NSGA-III preserves the niche solutions and selects them for reproduction in the following manner:

1. *Classification of population into Nondominated Levels*: Identifying nondominated fronts [69] and assigning membership of these levels to individuals in the population.

2. *Determination of reference points*: determining a set number of points on a hyper-plane that are well distributed in the objective-space. In the original 2014 paper, [62], the authors use Das and Dennis' symmetric approach for determining a normalised hyperplane in the objective space and finding well-separated points[17]. An example of the same is shown in Figure 5.2.

3. *Adaptive Normalization of Population*: The objective values are normalised and we take the extreme vectors (maximised for each vector). These are matched with the extreme points for the normalised hyperplane.

4. *Association Operation*: Each individual in the population is associated with the closest reference point on the adapted hyperplane.

5. *Niche-Preservation Operation*: The associations of individuals with reference points, and the perpendicular distance of individuals to the reference lines are used to select individuals that are added to the population of the next generation.

The authors suggest using the SBX operator to create offsprings for the next generation because it results in offsprings that are closer to the parents in the objective space.

[1]This information about the results and conclusion is placed here to give the reader a fair expectation how much they need to know about different EAs in detail to understand the result.

[2]It measures the relative isolation of $\vec{x}$ from the individuals in the population. The greater the crowding distance, the greater is its isolation. It is used to place new points in relatively unexplored regions and as far away from the existing points as possible [68].

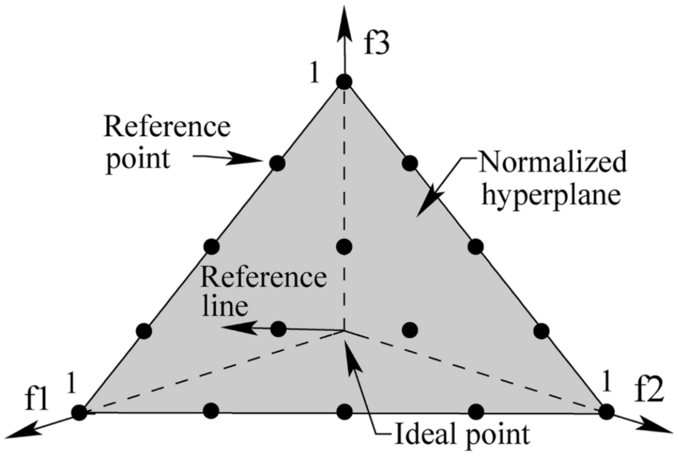


Figure 5.2: Fifteen structured reference points are shown on a normalized reference plane for a three-objective problem with [62]

5.4.2. GDE3

Generalised Differential Evolution, version 3 (GDE3) is an EA based on the Differential Evolution (DE) approach to solve MOO problems [70]. The principle of DE for searching the global optimum in constrained spaces was introduced by Storm and Prince [71, 72]. DE is a parallel direct search method that utilizes D-dimensional parameter vectors and improves the solutions through an evolutionary process. The performance of DE in a specific optimization problem depends largely on both the trial vector (search direction) generation scheme and the choice of the control parameters: population size, mutation factor, and crossover probability.

Understanding GDE3 requires some additional concepts. Firstly, for a real-valued nonlinear programming (NLP) problem, a solution that satisfies the following equality and inequality constraints and the above variable bounds is called a *feasible* solution[64]. Other solutions are called *infeasible*. Note that we have J number of greater-than inequality constraints, K number of equality constraints, N real-valued parameters in the genotype, and x_l, x_u are the lower and upper bounds for the genes, respectively.

- Minimizing $f(\vec{x})$, the objective function.
- Subject to:
  - greater than inequality constraints $g_j(\vec{x}) \geq 0$ where $j = 1 \dots J$
  - equal to inequality constraints $h_k(\vec{x}) \geq 0$ where $k = 1 \dots K$
  - real-value limits $x_l \leq x_i \leq x_u$ $i = 1 \dots N$

Furthermore, we need the concept of *constraint domination*. Here, $\vec{x}_1$ constraint dominates $\vec{x}_2$ *i.e.* $\vec{x}_1 \succ_c \vec{x}_2$ iff any of the following conditions are met.

- $\vec{x}_1$ is feasible and $\vec{x}_2$ is not.
- $\vec{x}_1$ and $\vec{x}_2$ are infeasible and $\vec{x}_1 \succ \vec{x}_2$ in constraint violation space.
- $\vec{x}_1$ and $\vec{x}_2$ are feasible and $\vec{x}_1 \succ \vec{x}_2$ in objective-space.

The first version of GDE improved on DE by modifying the selection rule; replacing an old vector with a trial vector for the next generation if it weakly dominates the old vector. The second version of GDE was modified to decide based on the crowdedness when the trial and the old vector were feasible and nondominating each other. **GDE3** improved on the previous GDEs by changing the following in the selection rule:

- In the case of infeasible vectors (breaking constraints), the trial vector is selected, if it weakly dominates the old vector in the constraint violation space, otherwise the old vector is selected.

- In the case of feasible and infeasible vectors, the feasible vector is selected.
- If both vectors are feasible, then the trial is selected if $\vec{x}_{trial} \succ_c \vec{x}_{old}$ in the objective-space. If$\vec{x}_{old} \succ_c \vec{x}_{trial}$ then the old vector is selected. If neither vector dominates the other, then both vectors are selected for the next generation.

Besides these small specialisations in selection and mutation operators, GDE boils down to the original DE methodology. It has two parameters of DE as the control parameters as well; Crossover rate CR and mutation scaling factor F. The parameter CR controls the crossover operation, as it represents the probability that an element for the $\vec{x}_{trial}$ is chosen from a linear combination of 3 randomly chosen $\vec{x}_i$ instead of $\vec{x}_{old}$. It precipitates to CR controlling the rotational invariance of the search. The parameter F controls the speed and robustness of the search through the amount of mutation.

5.4.3. SPEA2

The original Strength Pareto Evolutionary Algorithm (SPEA) was designed in 1999 [73]. The EA focused on three main aspects. Firstly, approximating the set of Pareto-optimal trade-offs in a single generation using clustering. Secondly, finding the fitness of individuals dependent on the number of nondominated points that dominated it. Lastly, preserving population diversity by archiving solutions

The improvements made by Strength Pareto Evolutionary Algorithm (**SPEA2**) [63], in 2001 are as follows:

- Improved fitness assignment with relation to other individuals, both nondominated and dominated.
- nearest-neighbour density estimation for a better guided search process. It is an adaptation of k^{th}nearest neighbour algorithm [74], where the density at any point is a (decreasing) function of the distance to the k^{th} nearest data point.
- New archival method that keeps an archive of constant size and the archive is truncated in a manner to preserve boundary solutions.

5.5. Performance Measurement

To evaluate the quality of the population/Pareto front, we will look at the following quality indicators. We have taken these varied indicators to measure the front quality because they all have their caveats[3], and gating more information about the front will lead to better statistical analysis with the same experimental design.

5.5.1. Hypervolume

Hypervolume (HV), *a.k.a.* Lebesgue measure [75], is a well-favoured metric by many EA practitioners to measure the volume of the objective space that is dominated by the Pareto front. Generally, the hypervolume is favoured because it captures in a single scalar both the closeness of the solutions to the optimal set and, to some extent, the spread of the solutions across the objective space. The hypervolume of a given set of points S is the size of the portion of objective space that is dominated by at least one point in S, with respect to a reference point. Unfortunately, hypervolume is costly to calculate for MaOO problems ($>$ 3 objectives).

For our purpose, we use an improved method of computing this metric called the dimension-sweep algorithm [76]. It is a recursive algorithm that avoids repeated dominance checks and recalculation of partial hypervolumes. Note that in this study, we will calculate the volume of the region dominated by the generated front, therefore the higher HV is better.

5.5.2. GD and IGD

The Generational Distance (GD) and Inverse Generational Distance (IGD) are widely used quality indicators to analyse the distance of a generated front, measured from a reference front (known as near-optimal front) [77]. Take into account for d objective functions we will have a $d-1$ dimensional surface as the generated front and reference front. To check the quality of different runs of the algorithm, we need to check how close both fronts are to each other. The GD [78] and IGD [79] are given by the

[3]These caveats will not be discussed as they lie outside the scope of the thesis.

following equations 5.3 and 5.4 respectively. Here, consider a candidate generated front $A = a_1, ..., a_N$ and the reference front $F = y_1, ..., y_M$. Note that M does not necessarily need to be equal or even close to N. d_i^p is the minimal p-norm distance from a_i to any point in F. $\bar{d}_i^p$ is the minimal p-norm distance from y_i to any point in A. In our case, both distances are Euclidean.

$$GD(A) = \frac{1}{N}\Big(\sum_{i=1}^{N} d_i^p\Big)^{\frac{1}{p}} \tag{5.3}$$

$$IGD(A) = \frac{1}{M}\Big(\sum_{i=1}^{M} \bar{d}_i^p\Big)^{\frac{1}{p}} \tag{5.4}$$

In essence, the difference between GD and IGD is that GD is the average minimum distance of the generated front from the reference front, whereas IGD is the average minimum distance of the reference front from the generated front. This distinction is important because in most cases $M \neq N$. Observing both quality indicators is important because we do not want our conclusions to be affected by variations in the number of points in the generated or reference fronts.

5.5.3. Epsilon Indicator

We use the *Additive Epsilon* indicator (EP) [80] that measures the smallest value to be added to a point in the candidate generated front, A, to make it non-dominated with respect to some point in the reference front, F. It is therefore the smallest value $\epsilon \in \mathbb{R}$ *s.t.* for any solution in F, there is at least one solution in A that is worse by ϵ [81]. In this thesis, we will refer to it as EP. The EP informs us about the nature of the generated front in terms of relative equivalence or improvement over the reference front. Keep in mind that if sets $A \equiv F$ or it dominates F the ϵ is 0, otherwise it is positive.

6 Methodology

To reiterate what was said in Chapters 1 and 2, the recommendation framework we plan to study takes an unspecified number of independent Recommender Systems (RSs) with specialised user-centric goals. It then combines the recommendations generated by these using Evolutionary Algorithm(s) (EAs). The EAs weigh the RSs in different ways and see how they perform on the user-centric recommendation metrics. The EAs efficiently search this problem space (weights) to search for a set of near Pareto-optimal solutions (combinations of weights) which give a trade-off over the different metrics.

For understanding the division of the dataset into $train$, $test_{RS}$ and $test_{EA}$ please read Chapter 3, Section 3.4. Here, when the information on whether the test dataset is $test_{RS}$ $test_{EA}$ does not matter, we use the notation $test_X$.

6.1. Recommenders

Let us see the RSs we used to generate recommendations and the motivations behind them. We will talk about the *expected behaviour* of each RS given our dataset. The RSs are implemented in Spark and meant to be scalable solutions for handling large amounts of data. More details about the implementation can be derived through reading Chapter 4, Recommender Systems and Chapter 7, Experiments.

6.1.1. User-Item Collaborative Filtering

Firstly, we use a simple User-Item Collaborative Filtering (CF) approach. We denote this by the notation $RS_{U\to I}$. Given the set of users in the $train$ set with all their $user \to item$ interactions, we generate recommendations for users in $test_X$ using their train interactions. If the recommendation works well, the recommended items should contain items in the test interactions of $test_X$.

6.1.2. User-Artist Collaborative Filtering

Secondly, we take a nontraditional approach by using CF to recommend artists to users. After we have the $user \to artist$ recommendations, we use an expansion method on the list to link $artist \to item$. We will use the notation $RS_{U\to A}$ for this recommender [1].

The motivation behind this method is that artists are loyal to their *musical styles*, and collaborative recommendations rooted in musical styles instead of the popularity of individual items may make the recommendations more diverse. We particularly attempt to sample items uniformly from the recommended set of artists for each user. This way, artists who are ranked lower still see their music appear in the recommendations if they are deemed to be relevant. The items for a particular artist (that they are associated with) are sampled in a random fashion.

6.1.3. Tailored Recommender

Lastly, we use a tailored approach to recommend items based on user's test interactions. We will use the notation RS_T for this approach. Provided with the $user \to item$ interactions in $test_X$, we do the

[1] Although the notation $RS_{U\to A\to I}$ would be more appropriate, we will go with the shorthand $RS_{U\to A}$.

following:

- Observe their $item \rightarrow artist$ relation to recommend other items from the artist the user has already heard. The artists are sorted by user's listening counts for each artist within the given time window (one year in this study).
- Like the previous recommender $RS_{U \rightarrow A}$, items are sampled uniformly across the artists, we know the user like, and all items are sampled randomly from the artists' known set of items.

The motivation behind this approach is that a simple logical expansion, knowing the user's observed listening behaviour, contrasts the highly generalised nature of the CF approaches that are rooted in Machine Learning disciplines. Moreover, recommending without generalising over other users may improve the personalisation of the recommended list and provide the user with novel items.

6.2. Recommenders with Evolutionary Algorithms

Let us now see how we use the RSs along with EAs from a technical perspective in this study. This includes formalising the problem encoding, selecting the objectives to optimise, and listing which EAs we use to solve the MaOO

6.2.1. Problem Encoding

To use the EA we need to mathematically encode the problem under consideration. We need to combine the outputs of n RSs and find the weights w_i for each RS, where $i \in \{1, 2, ..., n\}$. These weights are used to rerank the recommendation lists, after which they can be combined and simply sorted in descending order to get the processed list R_{proc}. We take the maximum list size of each RS's recommendations as k, combine them, and *trim* to remove duplicated items with higher rank. We keep the resulting list of maximum size K. For this study, we have kept $k = K$. Figure 6.1 illustrates how this looks from a broad view. For our problem, with 3 recommenders, the resulting recommendations will be made using 3 weighing coefficients, as shown in equation 6.1:

$$R_{proc} = trim\Big(desc\Big(\{w_1 \times RS_{U \rightarrow I}\} \cup \{w_2 \times RS_{U \rightarrow A}\} \cup \{w_3 \times RS_T\}\Big)\Big) \tag{6.1}$$

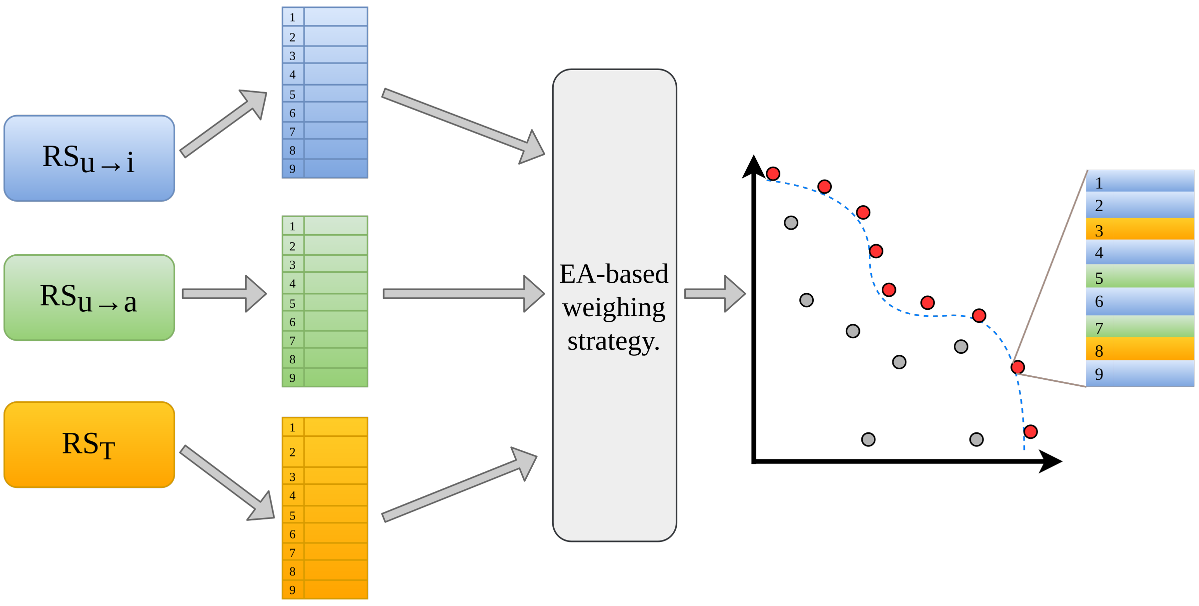


Figure 6.1: Diagram showing fusion of recommendation lists using an EA to generate a set of solutions that result in a near-optimal objective front (with 2 objectives as an example).

In 2020, Xingjuan Cai, *et al.* [2] used real-valued array of length $n - 1$ to encode the problem. They encoded such a way that sum of the $\sum w_i = 1$. They calculated the last n^{th} weight could be found using $w_n = 1 - \sum_{i=1}^{n-1} w_i$. This was done to reduce the number of variables in the problem encoding.

We go in a different approach and have n variables and no constraints on $\sum w_i$. This adds an extra dimension to the solution space, but this makes the optima easier to identify in the solution space. This is done because the dimension should help propagate information about the last recommender during crossover more easily.

The value for each weight is bounded between 0.001 and 1.0. This is done because (i) it bounds the search space, (ii) the lower bound becoming negative ($w_i < 0$) would flip the order of items in the list, and (iii) the lower bound becoming 0 ($w_i = 0$) would give the same rank to all items in a particular RS's list and descending sort will become meaningless.

6.2.2. Optimisation Objectives

In Section 2.6, of chapter Background, we saw what types of objectives were used by different research groups in their endeavours. The user-centric recommendation metrics we use as objectives to *maximise* use are Mean Average Recall (*MAR*), Coverage (*Cov*), Novelty (*Nov*) and a modified Personalization (*modPers*) metric. These metrics have been described in detail along with the motivations for their use in Section 4.2. All these objectives theoretically range from 0 to 1, but practically each of them has different bounds for real-world tasks, and they vary in range.

To summarise the referred Section 4.2, we calculate these metrics over the set of all users $u \in U$, on their recommendation lists R_u of length k. Here, T_u denotes the items actually heard by the user u, obtained from the test interactions. The catalogue Cat is the set of all items that can be recommended in a year. $U_{|R_u| \geqslant k}$ denotes users whose recommendation lists are of size greater than k.

$$
\begin{aligned}
MAR@k &= \frac{1}{|U|} \sum_{u \in U} \frac{R_u \cap T_u}{|T_u|} \\
Cov@k &= \frac{R_U \cap Cat}{|Cat|} \\
Nov@k &= \frac{1}{|U|} \sum_{u \in U} \frac{R_u \cap \Gamma}{k} \\
modPers@k &= 1 - \left(\frac{1}{U_{|R_u| \geqslant k}} \sum_{m,n \in R_{U_{|R_u| \geqslant k}}} \frac{m \cap n}{k} \right)
\end{aligned}
\tag{6.2}
$$

6.2.3. Evolutionary Algorithms

We use three different EAs to study how effectively we can obtain the near-optimal set of solutions. In this section, we will not go into too much detail about each of the algorithms. For a more detailed description of each algorithm, please look at Section 5.4. We use these algorithms because they take different mathematical approaches for solving MOPs.

NSGA-III (Nondominated Sorting Genetic Algorithm III) [62] is a reference direction guided search EA. These reference directions/points need to be provided when the algorithm is initialized. NSGAIII prioritises adding offsprings to unexplored directions. Each reference direction seeks to find a well-distributed front of non-dominated solutions.

GDE3 (Generalised Differential Evolution) [70] is an EA that relies on the mathematical foundations of Differential Evolution (DE). It has optimisations which make it well suited for solving MaOO problems. In essence, it searches for a global optima or the Pareto-optimal set by creating trial vectors which are individuals with small randomised perturbations, while keeping track of good solutions.

SPEA2 (Strength Pareto Evolutionary Algorithm) [63] searches for the Pareto-optimal set by approximating the shape of the front. It makes sure the offsprings move in a direction where the front is approximated to be. It estimates the shape of the front using a density estimation algorithm and keeps track of diverse solutions while doing so.

7 Experiments

In this chapter, we will talk about how we structure the experiments for reliable validation of the recommendation framework. We will also discuss the computing environment that we used to show what it takes to reproduce this study. We also note the hyperparameters for the different modules of this project to make any discrepancies relating to performance and results clearer for the readers of this thesis document.

7.1. Experimental Design

We decided to study 3 different factors that may affect the performance of the recommendation framework. These factors are listed in Table 7.1 with the considered levels for a *full-factorial* experimental design [82]. Here is the motivation for choosing them along with the questions we aim to answer.

- *Year*: the year of the MLHD user data. The experiment is designed in such a way that we have data available for one year and we make recommendations at midnight of the New Year's Eve (31st Dec, 23:59:59). We want to investigate if the change in user base and interaction patterns across different years, which we saw in Chapter 3, affect the recommender systems and/or the quality of the generated fronts by the EAs.
- *Test set type*: the test set we tune on, $test_{RS}$, vs an independent test set $test_{EA}$. It is important to investigate if given a similar distribution of data, user-base, and interaction patterns, does simply testing the system on a new set of users affect the results? If so, then to what extent?
- Algorithm: these are the different EAs we have considered in the methodology. The motivation for these particular algorithms is given in Chapter 5. We want to investigate if given a similar data distribution, does simply using a different EA change the quality of the generated front? If so, then in what way?

Factors	Levels		
Year	2005	2008	2012
Test set type	$test_{EA}$	$test_{RS}$	
Algorithm	NSGA-III	GDE3	SPEA3

Table 7.1: Factors considered for the full factorial experimental design, along with levels for each.

We will quantitatively analyse the performance of different EAs using the front quality indicators and the time required to finish the specified number of generations of the EA ($Time$). The quality indicators are: Generational Distance (GD), Inverse Generational Distance (IGD), Additive Epsilon indicator (EP), and Hypervolume (HV). These indicators have been described in Chapter 5, under

Section 5.5, Performance Measurement. Furthermore, we will qualitatively analyse the values for user-centric metrics achieved by different recommenders and performance indicator values for the EAs, across the different factors under study.

In Chapter 3 we explained that while preprocessing MLHD, besides making two test sets ($test_{EA}$ and $test_{RS}$), we also made 3 different interaction sets for each. This was done by splitting every test users' $user \rightarrow item$ interactions 50-50%, 3 times, randomly. This was done to make sure that our results for our RS metrics and EA generated fronts are not sensitive to how we sample the user interactions each year. Lets denote these interaction sets for each of the 3 test sets by δ.

To make sure we have good randomization for the experiments besides the δ we will also take 3 repetitions, r, to make sure we have sampled enough randomized starts for each combination of levels for the factors. The final number of experiments are given by the mathematical working 7.1. That makes a total of 162 runs.

$$\begin{aligned} N_{full} &= |year| \cdot |test_X| \cdot |algo| \cdot (|\delta| \cdot r) \\ &= 3 \cdot 2 \cdot 3 \cdot (3 \cdot 3) \\ &= 3 \cdot 2 \cdot 3 \cdot 9 \\ &= 162\ runs \end{aligned} \tag{7.1}$$

It is important to note that GD, IGD and EP require a reference front to compute the distance from. We do not know what is the optimal solution for every unique problem. We take a union of the Pareto sets of all solutions, find the Pareto set of this union, and use it as the reference front. A unique (real-world) problem here is the combination of $(year, test_X)$, where $year \in \{2005, 2008, 2012\}$ and $test_X \in \{test_{RS}, test_{EA}\}$. For each unique problem, there exist 3 randomised sampling sets of edges which result in 3 sets of train-test interactions, δ, and 3 repetitions of every EA on it, just to make sure our results are not affected by the various critical choices we make in this study. Therefore, every $(year, test_X, EA)$ combination has 9 runs in total.

7.2. Compute Environment

It is important to explain the computing environment in detail for the good reproduction of this research. Although the software output for this thesis is extremely modular and configurable, reproducing it on another computational setup will lead to different results. In many cases, without the proper setup, one would not be able to perform many intermediate actions the system requires.

Group	**Property**	**Local Setup**	**Server**
Compute	CPU	Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz	Intel(R) Xeon(R) CPU E5-2650 v3 @ 2.30GHz
	CPU count	1	2
	Total #CPU cores	6	10
	Total #threads	12	40
Memory	Main Memory Amount	16GiB, DDR4, 2667MHz	128GiB
	Swap used/needed (greater number)	40GiB	8GiB
Storage	Storage Type	SSD	Network attached SSD RAID
	Amount required	< 512GiB	< 128GiB

Table 7.2: Hardware setup details for better insight into experiment execution choises.

The setup details are given in Table 7.2. Many critical decisions were motivated by the available compute infrastructure. ArangoDB[1] was hosted on the local setup because of the available SSD[2] stor-

[1]refer to Section 3.1.2 for background.
[2]Solid State Drive

age, as opposed to the server's Network-attached SSD RAID[3]. Querying data from ArangoDB collections does require a lot of memory to access the collection indexes and other bookkeeping necessities for streaming the data to Spark applications to the server. ArangoDB took up at most 15GiB main memory and 40 GiB of swap memory to reliably send records to the Spark jobs. The time for the same is recorded in Table 7.3, which is collected from the Spark history server.

Recommender	**ArangoDB collections**						Time
	$user$	$artist$	$items$	$user \rightarrow item$	$user \rightarrow artist$	$artist \rightarrow item$	
$RS_{U \rightarrow I}$	✓		✓	✓			22 min
$RS_{U \rightarrow A}$	✓	✓	✓	✓	✓	✓	35 min
RS_T	✓	✓	✓	✓	✓	✓	29 min

Table 7.3: ArangoDB collections required for each RS and the time required to fetch data before actual compute begins. Note the collections with → are edge collections, otherwise node collections.

7.2.1. Selecting 200 MLHD parts

The MLHD data is split into 575 parts, each containing about 1000 user log files. We cannot use all 575 parts because the number of runs of different recommenders and algorithms our experimental design requires is too great. Using the entire MLHD would take exponentially more time for all runs. For a clearer picture, we need to generate recommendations using 3 different RSs for 3 different data splits by year, each having 2 types of test sets ($test_{EA}$ and $test_{RS}$), each having 3 numbered sets. After that, we need to run the Evolutionary Algorithms according to the experimental design.

We performed a heavy analysis on MLHD_000.tar to understand the data, thus we are not blind to its biases and need to take *additional* 200 parts. Thus, we used only 201 MLHD parts of the 575 parts for this study. With 201 MLHD parts, all Spark jobs took 414.2 minutes (6hr 54 min). The EA jobs took 3963.7 minutes (66hr, 4min). This totals up to 4377.9 minutes, or a ***little over 3 days*** of compute usage. This does not even account for weeks of setting up the testbed to make sure the experiments will run reliably and complete within 1 week. We do this by scaling up from smaller to larger sizes of the dataset, to detect and eliminate any bottlenecks.

7.2.2. Recommender Hyperparameters

We wanted to keep the parameters for the Alternating Least Squares Collaborative Filtering [50] similar across different runs of the experiment. This is because we did not want our experiments to be affected by excessive hyperparameter tuning [83]. To reiterate, we move from a smaller dataset to a larger one while deciding the parameters. We need to select the parameters for the collaborative filtering aspects of the RSs. We decided to set enough model flexibility to accommodate the largest problem sets (year-split 2012) with 50 MLHD parts and stop when we see no improvement in RMSE[4] on the interactions in all three validation sets. We adopt these parameters and simply run on the larger set of data, which is supposed to scale well without any human intervention.

As the $user \rightarrow artist$ interactions are less than $user \rightarrow item$ we decided to adopt the settings tuned for $RS_{U \rightarrow I}$, for the $RS_{U \rightarrow A}$. These values are *10 latent factors*, a maximum of *10 iterations of CF*, and the *regularizing* parameter and *alpha* as 1.0. The only parameter we had to change across the year-splits was the *numBlocks*, due to the memory limitations. The numBlocks were set to 10, 28, and 48 for the years 2005, 2008, and 2012, respectively.

7.2.3. Evolutionary Algorithm Hyperparameters

Table 7.4 shows the hyperparameters set for the EAs for the performed experiments. As noted in the methodology, Section 6.2.1, we take k items from the recommendations of each RS and combine them to get a list of size K. For these experiments, we kept $k = K = 10$.

We use NSGA-III and SPEA2 with Polynomial mutation and SBX crossover which have been described in Chapter 5. GDE3 used the recommended setting for CR and F were set as 0.5 [66]. NSGA-III

[3]Redundant Array of ~~Inexpensive~~/Independent Disks
[4]Root mean square error

requires the reference point generation method as an input. We kept it the same as NSGA-III's original paper [62], where they use a Das and Dennis's method of sampling well-spaced points on a symmetric hyperplane [84]. We decided to go with 35 points, which symmetrically fit on a hyperplane[5]. This meant our population had to be greater than 35. Our computational setup could handle a population of 40 while using memory optimally on the largest unique problem *i.e.* utilized up to 112 GiB out of 128 GiB of memory to calculate the fitness of every individual in the population.

EA	**Hyperparameter**	**Hyperparameter specification**	**Value**
NSGA-III	Polynomial Mutation	probability	⅓
		distribution index	20
	SBX (Crossover)	probability	1
		distribution index	20
	Reference Points [84]	number of points	35
SPEA2	Polynomial Mutation	probability	⅓
		distribution index	20
	SBX (Crossover)	probability	1
		distribution index	20
	Offspring population size	-	40
GDE3	Crossover Rate (CR)	-	0.5
	Mutation Scaling factor (F)	-	0.5

Table 7.4: Hyperparameters for the Evolutionary Algorithms used in the experimental setup.

[5]Using the formula $H = \binom{M+p-1}{p}$, where M is the number of objectives and p is the number of partitions in the hyperplane edge. By taking $M = 4, p = 4$ we get 35 points.

8
Results

In this chapter, we see the results of the conducted experiments and more importantly the interpretations of these. We will first analyse how the Recommender Systems (RS) perform individually on user-centric metrics. Then we will see how RSs compare to each other under different contexts. This will show us the capabilities of different RSs and how they handle situations external to the recommendation task. We will then analyse the performance of the Evolutionary Algorithms (EA) in a similar manner and then proceed to the quantitative tests to see which factors affect the quality of the generated front, and how they are affected.

At the end of this chapter, we will see some insightful visualizations. First, '4D' plots and parallel coordinate plots to see the objective fronts generated by different EAs. Then we will see how the different RSs are weighed to solve every unique problem case *i.e.* $(year, test_X)$ combination.

8.1. Recommenders

Let us analyse the behaviour of the Recommender Systems we use before going into how well they combine. It is important to understand how these models behave to properly understand what the EAs are working on to generate near-optimal fronts. It will also enable us to fathom to what extent can EAs balance potentially temporally unstable recommenders to still create good solutions.

Independent Behaviour

It is important to understand how the recommenders behave in a normal setting where we simply push out items and see how well they perform on user-centric metrics. In the methodology, Chapter 6, we gave an expectation for the behaviour of the RSs. Each RS has a slightly specialised goal which should lead to differences in user-centric metrics. In addition, recall that these metrics are not equal *i.e.* although they all range from 0 to 1, they have different practical bounds and these bounds change depending on the distribution of the underlying data.

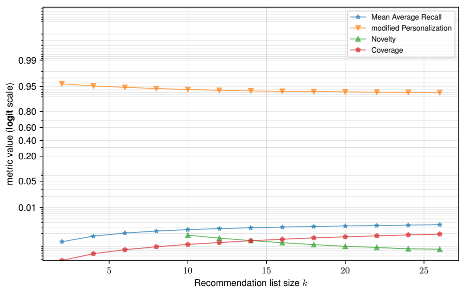
Figure 8.1 shows an example of the 3 different RSs' comparative performance on the user-centric metrics for the year 2008. We see the performance at difference list size k to see how RSs prioritise these metrics and how do they converge. All figures from this analysis are provided in Appendix D.

From the provided figures, here and in the appendix, we can note the following:

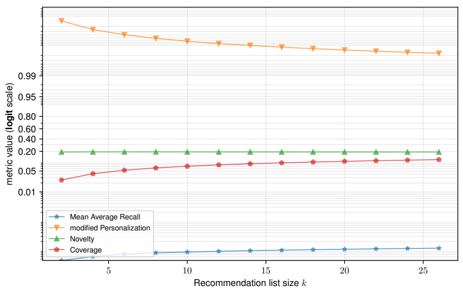
- All recommenders have a really low MAR. This depends on the underlying data the RSs tries to model and is averaged over really diverse users, while not removing users who have fewer observations. As we preserve a lot of minority users this low value is expected.
- $RS_{U \to I}$ while showing consistent behaviour on MAR, performs abysmally on coverage and novelty. Especially in 2012, we see that Novelty becomes extremely poor and unstable. It also shows consistent performance on the personalisation metric at different list sizes *i.e.* personalisation does not improve or worsen much visually as the list size increases and more items are considered.
- $RS_{U \to A}$ shows steady behaviour in terms of MAR. It clearly prioritises personalisation and we see it fall as more items are considered. It steadily recommends novel items and the coverage increases and converges as the list size increases.

- RS_T shows extremely tailored personalisation. We cannot see personalisation on the plots because the plots are logit-scaled and personalisation approaches the value 1.0 (maximum), which means it cannot be mapped on the plot. Coverage and MAR increase and converge as the list size increases. Novelty is always steady.

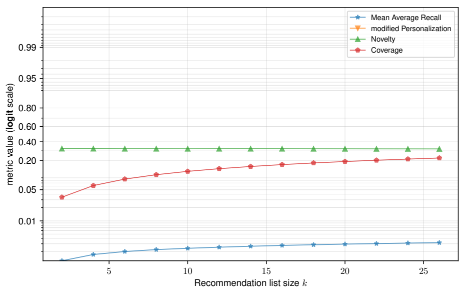
These points being made, we cannot do a side-by-side comparison of different models using this visualization method. We need a better way to analyse the comparative behaviour of these models.



(a) Recommender System $RS_{U \to I}$



(b) Recommender System $RS_{U \to A}$



(c) Recommender System RS_T

Figure 8.1: Example of different recommenders' metrics achieved on $test_{RS}$. Here example year is 2008, for all the 3 years under study see the appendix, Appendix D.

Comparison Across Recommenders

We analyse the behaviour of the RSs compared across different recommenders to understand how individual metrics are valued and prioritised by each recommender. Again, We see the performance at different list sizes k, but also across different years to see how the models perform under the same distribution of data. We can see the visualisation of the same in Figure 8.2. From the data we can observe that:

- The trends for Coverage remain consistent across recommenders. Coverage of $RS_{U \to I}$ is less than that of $RS_{U \to A}$, which is less than RS_T.
- The trend for MAR is inconsistent. $RS_{U \to A}$ always performs worst. In 2005 and 2012, RS_T performs better than $RS_{U \to I}$, but in 2008 the pattern is reversed.
- The trend for Novelty is regular but changes through time. RS_T always performs better than $RS_{U \to A}$, which always performs better than $RS_{U \to I}$. The inconsistency is in the amount by which it performs better; we see the performance gap between $RS_{U \to A}$—RS_T and $RS_{U \to A}$—RS_T equalising as we move forward in time.
- The comparative trend for Personalisation is consistent. $RS_{U \to I}$ always performs worse than the other RSs. The rate at which the performance of $RS_{U \to I}$ degrades is also much worse than the others.

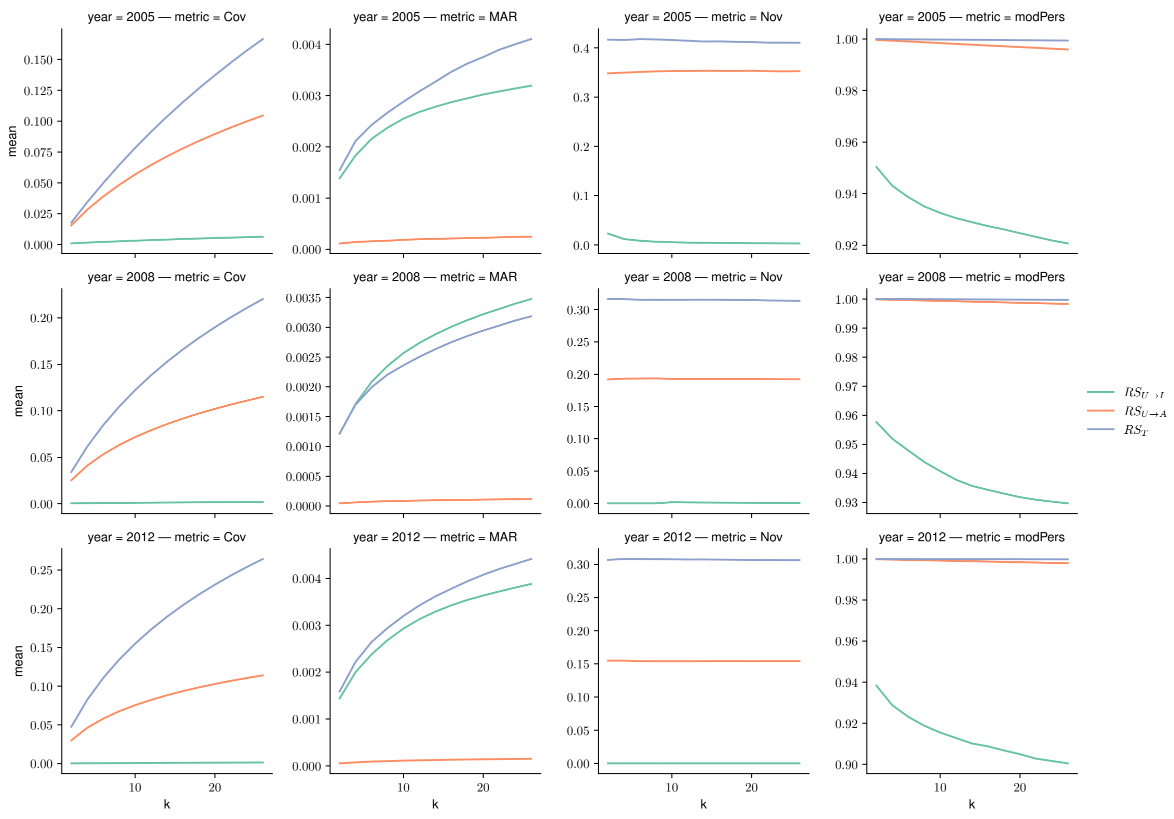


Figure 8.2: Recommender Systems' comparison of performance **across recommenders**. *Note (i) X-axis has list sizes k going from 2 to 26 with increments of 2 (ii) compare vertically (starting from, go top-down) (iii) y-axis limits different for all subplots.*

Comparison Across Years

Recall that we take 3 different years under study, and in Section 3.3 we saw that the data distribution and usage behaviour change across years. It would be natural to theorise a change in how different RSs model the distribution at any given time point to make recommendations. We can see the visualisation of the same in Figure 8.3. In the visualisations we can observe that:

- Coverage of $RS_{U\to I}$ decreases starkly as the years progress *i.e.* simple collaborative filtering recommends less proportion of items as we move forward in time. Coverage of both $RS_{U\to A}$ and RS_T increases as we move in time and more data is available.
- The changes in MAR for the three RSs are observably different. $RS_{U\to I}$ performs quite similarly across different years, but in general improves over years. MAR of $RS_{U\to A}$ worsens going from 2005 to 2008 but improves when we move to 2012. The MAR for RS_T, as well, worsens going from 2005 to 2008 but improves when we move to 2012, but unlike $RS_{U\to A}$, the MAR for 2012 is greater than 2005.
- The Novelty across the years behaves observably differently for the three models. The novelty of $RS_{U\to I}$ sharply drops across all list sizes as the years go on, and the shape of the curves shows near-zero novelty for years 2008 and 2012 for lower list sizes. $RS_{U\to A}$ and RS_T both show a drop in novelty as the years progress, but the comparative gap for RS_T is smaller.
- The Personalisation of $RS_{U\to I}$ drops consistently across all list sizes as the years go on. That of $RS_{U\to A}$ first increases from 2005 to 2008, and then slightly decreases. The personalisation of RS_T consistently increases as years go on.

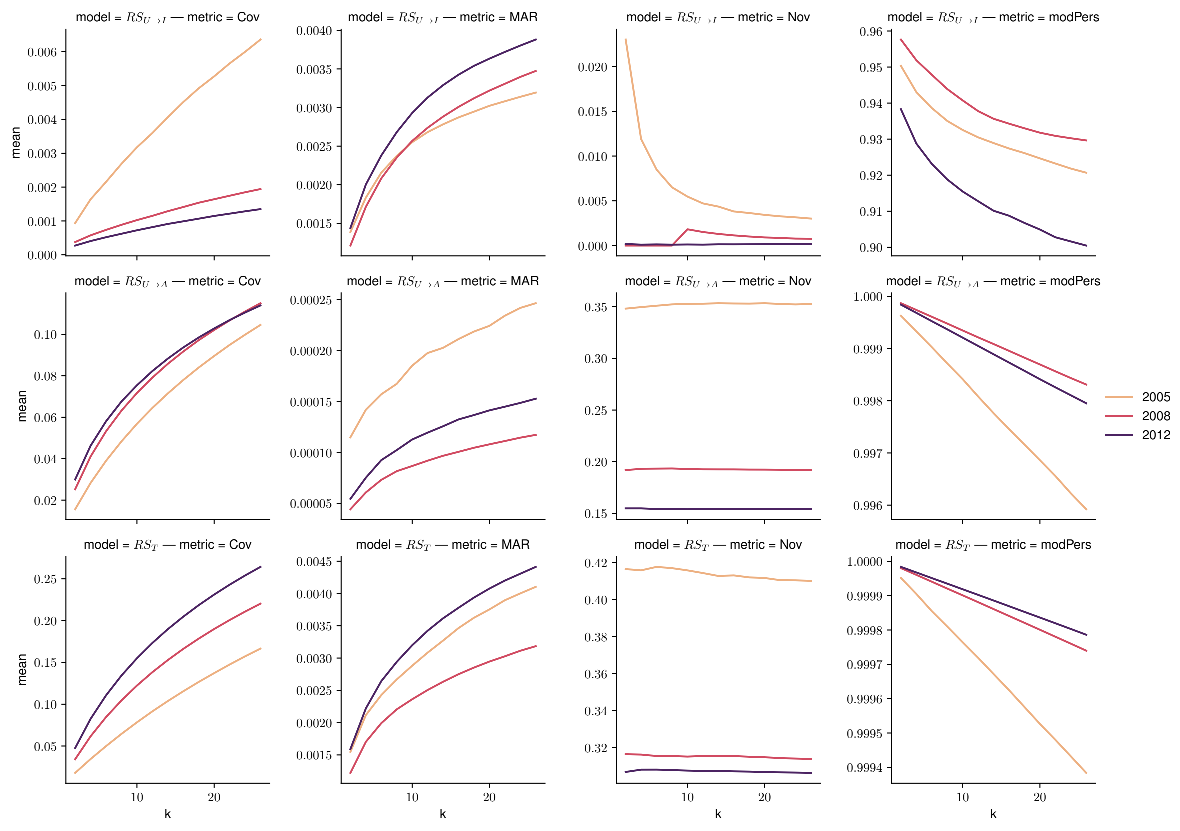


Figure 8.3: Recommender Systems' comparison of performance **across years**. *Note (i) X-axis has list sizes k going from 2 to 26 with increments of 2 (ii) compare vertically (top-down, left-right) (iii) y-axis limits different for all subplots.*

8.2. EA Experiments

We have seen how starkly different the RSs perform under different temporal contexts, and in relation to each other under those contexts. Now let us move on to see how well Evolutionary Algorithms perform while weighing these recommenders to generate a near-optimal front to maximise the user-centric metrics.

In Section 7.1 we described the experimental design used to reliably verify how well different EAs can solve the aforementioned MaOO problem. We will first analyse the obtained results visually to note any obvious trends or uncharacteristic variations we see. Then we will move on to the quantitative significance test to analyse empirically how different factors in the experiment affect the targets (front quality and run-time).

8.2.1. Quality Indicators

We will again make visualizations to analyse the results across factors. First across different EAs, then across the temporal context, and lastly across test sets ($test_{RS}$, $test_{EA}$). For these, visualizing *swarm-plots*[1] seemed to be fitting because we can show the distribution of the data without any quantitative summarization (which would mean loss of information) and we do not need to worry about overlapping data points.

For understanding certain observations, recall that we do 3 different samplings with 3 repetitions ($3 \times 3 = 9$) for every *unique problem* set. Every year presents 2 unique problems which give the reference front used to calculate the GD, IGD, and EP (ϵ). Therefore, if an EA solves the problem consistently across different runs, then it is natural to see 3 clusters of 3 points each in a quality metric.

[1]A categorical scatterplot with non-overlapping points. For each category, points are translocated to match the value on the numerical axis as close as possible to the original without any data-point overlap.

Comparison Across Algorithms
In Figure 8.4 we can see the different algorithms coloured differently. The year increases from top to bottom, and each column shows a different quality indicator. We can observe the following from the data:

- The GD values of the three EAs (NSGA-III, GDE3, and SPEA2) are very similar every year.
- The IGD values for GDE3 and SPEA2 are quite consistent. NSGA-III shows variation in IGD compared to other EAs. This may be because of the number of reference point hyperparameter settings of the algorithm.
- The Epsilon value (EP) is consistent across the three EAs for 2005. The years 2008 and 2012 have NSGA-III showing slight variation in observations compared to others.
- The Hypervolume (HV) is consistent for the three EAs every year.
- The comparative Time required for the EAs to complete their runs varies a lot as the years change (problems change). SPEA2 takes much longer than the others in 2005, observably more by 27s on average (18% increase). GDE3 takes longer than others in 2012, observably more by 2min 30s on average (4% increase).

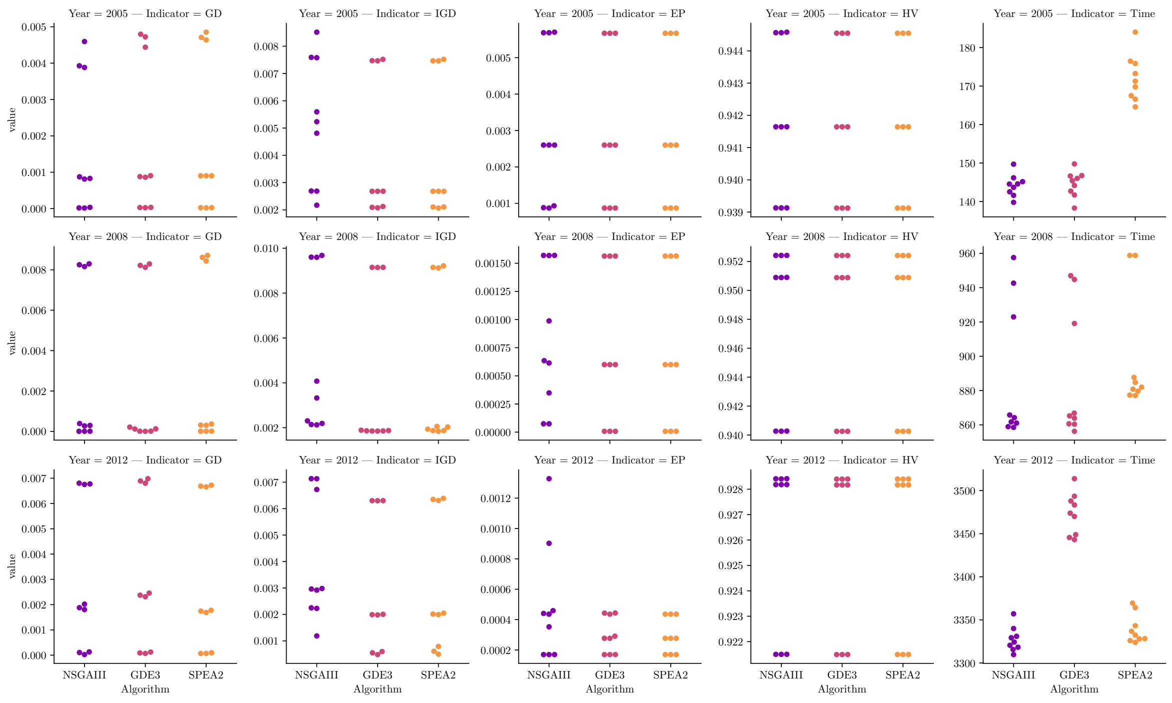


Figure 8.4: Swarmplots of EA quality indicator values for comparison **across different EAs**, for runs on $test_{RS}$

Comparison Across Years
In Figure 8.5 we can see observations from different years coloured differently. The algorithms change from top to bottom, and each column shows a different quality indicator. We can observe the following from the data:

- The GD for each year shows slight variations, with one of 2008's problems having a particularly high GD. Overall, all algorithms seem to show consistent trends in GD.
- The IGD, like the GD, shows slight variations across years and every year observably has one problem which performs *comparatively* worse than the others.

- The EP steadily reduces as the years go on *i.e.* the minimum required shift of a point to make the point nondominated (better) compared to a point in the reference front lowers. It gives single-point information and should be taken with a grain of salt. Moreover, recall that all metrics have different practical bounds.
- The HV for the year 2012 is lower than 2005 and 2008 *i.e.* dominated region (volume) by the front reduced in 2012. Thus, we cannot achieve similar performance with the three RSs across time.
- The Time required for a single run of an EA increases as the years go on *i.e.* more user data is available, and the fitness function takes longer to compute.

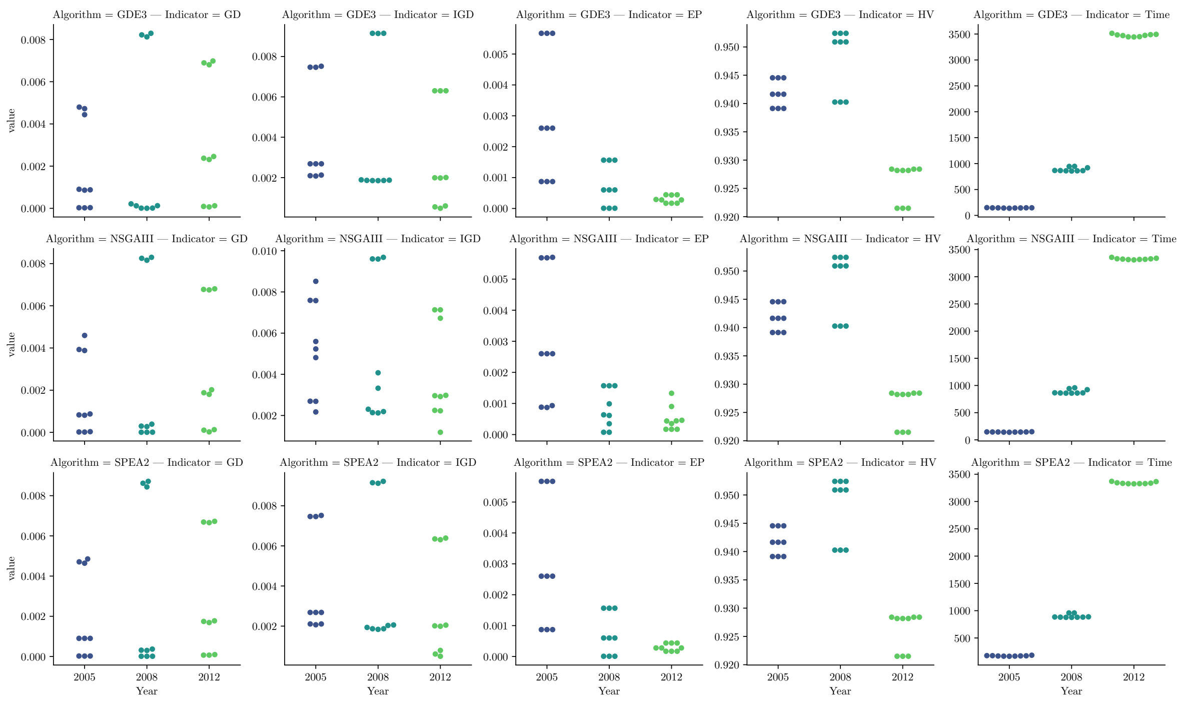


Figure 8.5: Swarmplots of EA quality indicator values for comparison **across different years**, for runs on $test_{RS}$

Comparison Across Test Sets

In Figure 8.6 we can see two different test sets ($test_{RS}$, $test_{EA}$) coloured differently. This analysis is important to note if the performance changes when we test the framework with different users. The visualization pattern is the same as the analysis across years in Figure 8.4, but with more data points and different implications. Ideally, clusters (of 3 points) for different test sets should be nearby each other. The year changes from top to bottom, and each column shows a different quality indicator. We can observe the following from the data:

- The GD for both test sets shows visually similar behaviour for the three years and the three algorithms.
- While the IGD for $test_{EA}$ is comparatively lower than $test_{RS}$ in 2005, it increases in comparison over years. The algorithm-wise inconsistency for NSGA-III is presented for both test sets.
- The EP for 2005 and 2008 showed similar behaviour for the two test sets. The trend changes in 2012 where the EP for $test_{EA}$ (unseen users) is on average higher than that for $test_{RS}$ (users optimised for) *i.e.* performance is comparatively worse for unseen users.
- The hypervolume is visually much lower (worse) for $test_{EA}$ (unseen users) than $test_{RS}$ for the year 2005. Years 2008 and 2012 have mixed performances.
- The time required to finish the runs is similar for the two test sets.

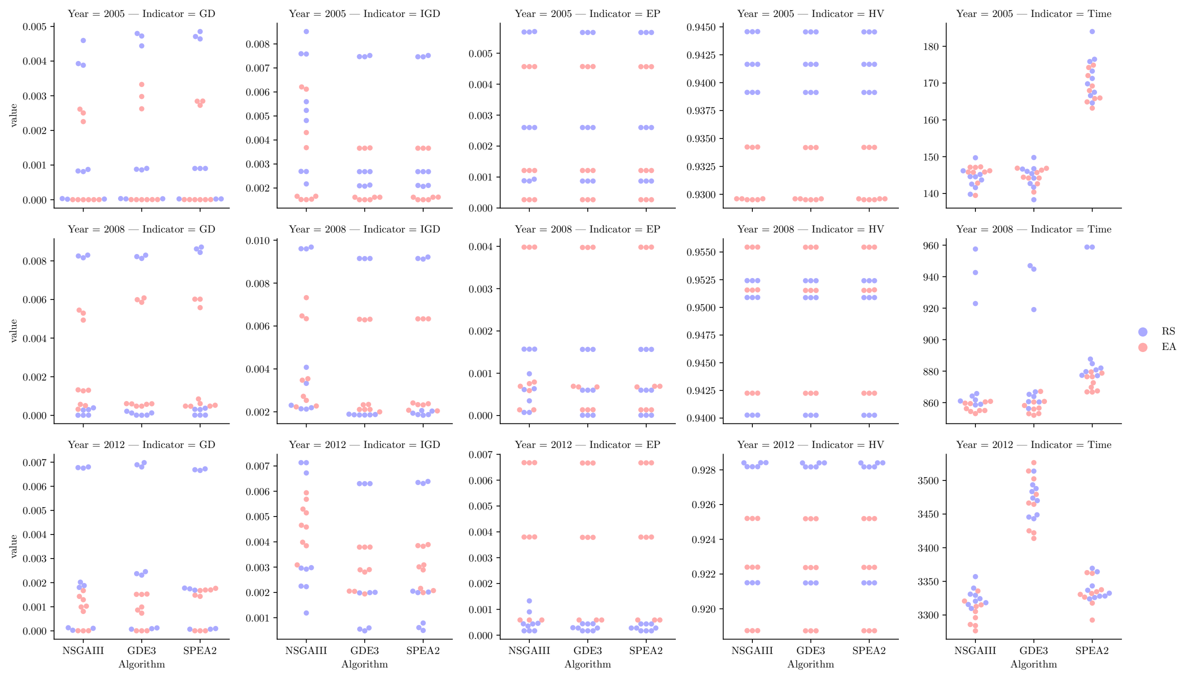


Figure 8.6: Swarmplots of EA quality indicator values for comparison **across different test sets** $(test_{RS}, test_{EA})$

8.2.2. Quantitative Significance

Now we elicit which differences in the achieved quality indicator values are actually significant, through quantitative analysis of the experiment runs.

We go with the Mann Whitney U Test (Wilcoxon Rank Sum Test) paired statistical significance test [85, 86], used with Bonferroni correction[87]. We chose the Wilcoxon test because of 3 reasons: (i) The normalcy assumptions for the more popular parametric methods such as ANOVA do not stand true with our observations, (ii) we go with groupwise paired tests to observe the median shift instead of assuming the observations come from a set distribution, and (iii) the Wilcoxon test is commonly used to perform quantitative performance comparison for EAs [88]. The Wilcoxon-Mann-Whitney test has the following hypotheses:

H_0: The two populations are equal (if $p \geq \alpha$).

H_1: The two populations are not equal (if $p < \alpha$).

The Bonferroni correction is a multiple-comparison correction used when several dependent or independent statistical tests are being performed simultaneously. When making multiple comparisons, the alpha value may be appropriate for each individual comparison, but it is not appropriate for the set of all comparisons. To avoid excess spurious positives, the alpha value needs to be lowered to account for the number of comparisons being performed.

We use the Pairwise Vargha and Delaney's A (VDA) and Cliff's Delta (CD) for effect size. These are usually adopted in cases where a Wilcoxon-Mann-Whitney test is used [89]. VDA ranges from 0 to 1, with 0.5 indicating stochastic equality, and 1 indicating that the first group dominates the second. CD ranges from -1 to 1, with 0 indicating stochastic equality, and 1 indicating that the first group dominates the second.

The table containing the p-values for the Wilcoxon-Mann-Whitney test, and the table containing the values for VDA and CD for significant interactions is presented in Appendix B. To make the results clearer, we have summarised the interpretations of the numbers in Table 8.1. The table is colour-coded for ease of understanding.

Grouping	Comparison		Indicator									
			GD		IGD		EP		HV		Time	
			p	better (lower)	p	better (lower)	p	inform (lower)	p	better (higher)	p	better (lower)
Year	2005	2008	✓	2005			✓✓	2008	✓✓	2005 $(CD = 0.08)$	✓✓	2005
	2005	2012							✓✓	2005	✓✓	2005
	2008	2012							✓✓	2008	✓✓	2008
Set Type	$test_{EA}$	$test_{RS}$	✓	$test_{EA}$			✓✓	$test_{RS}$				
Algorithm	GDE3	NSGAIII			✓✓	GDE3						
	GDE3	SPEA2										
	NSGAIII	SPEA2			✓	SPEA2						
Algorithm : Set Type	No significant effects											
Year : Set Type	2005 $test_{EA}$	2005 $test_{RS}$	✓✓	2005 $test_{EA}$	✓✓	2005 $test_{EA}$			✓✓	2005 $test_{RS}$		
	2005 $test_{EA}$	2008 $test_{EA}$	✓✓	2005 $test_{EA}$	✓	2005 $test_{EA}$			✓✓	2008 $test_{EA}$	✓✓	2005 $test_{EA}$
	2005 $test_{EA}$	2008 $test_{RS}$	✓✓	2005 $test_{EA}$	✓	2005 $test_{EA}$			✓✓	2008 $test_{RS}$	✓✓	2005 $test_{EA}$
	2005 $test_{EA}$	2012 $test_{EA}$			✓✓	2005 $test_{EA}$			✓✓	2005 $test_{EA}$	✓✓	2005 $test_{EA}$
	2005 $test_{EA}$	2012 $test_{RS}$	✓✓	2005 $test_{EA}$			✓✓	2012 $test_{RS}$	✓✓	2005 $test_{EA}$	✓✓	2005 $test_{EA}$
	2005 $test_{RS}$	2008 $test_{EA}$					✓✓	2008 $test_{EA}$	✓✓	2008 $test_{EA}$	✓✓	2005 $test_{RS}$
	2005 $test_{RS}$	2008 $test_{RS}$					✓✓	2008 $test_{RS}$	✓✓	2008 $test_{RS}$	✓✓	2005 $test_{RS}$
	2005 $test_{RS}$	2012 $test_{EA}$							✓✓	2005 $test_{RS}$	✓✓	2005 $test_{RS}$
	2005 $test_{RS}$	2012 $test_{RS}$					✓✓	2012 $test_{RS}$	✓✓	2005 $test_{RS}$	✓✓	2005 $test_{RS}$
	2008 $test_{EA}$	2008 $test_{RS}$									✓✓	2008 $test_{EA}$
	2008 $test_{EA}$	2012 $test_{EA}$							✓✓	2008 $test_{EA}$	✓✓	2008 $test_{EA}$
	2008 $test_{EA}$	2012 $test_{RS}$							✓✓	2008 $test_{EA}$	✓✓	2008 $test_{EA}$
	2008 $test_{RS}$	2012 $test_{EA}$					✓✓	2008 $test_{RS}$	✓✓	2008 $test_{RS}$	✓✓	2008 $test_{RS}$
	2008 $test_{RS}$	2012 $test_{RS}$							✓✓	2008 $test_{RS}$	✓✓	2008 $test_{RS}$
	2012 $test_{EA}$	2012 $test_{RS}$					✓✓	2012 $test_{RS}$	✓✓	2012 $test_{RS}$		

Table 8.1: Interpretation of Wilcoxon-Mann-Whitney test, VDA, and CD statistics which are recorded in tables B.1 and B.2 in the Appendix. Note (i) ✓indicates $p < 0.05$, and ✓✓indicates $p < 0.01$. (ii) $test_{RS}$ and $test_{EA}$ are colour-coded to easily distinguish between the seen and unseen test-sets (for quality indicators) respectively. (iii) CD value only shown when very close to 0 (slight domination).

In the table, we see important interactions of factors. Note that although we analysed, we have not listed the *Algorithm:Year* interactions and the three-way interaction between *Algorithm:Year:Test Set*. These values are mostly significant when we compare an EA of one year with another and not appropriate. This because when we change the unique problem, the reference front changes and we know the hypervolume across the year changes. This data does not give any valuable information for analysing the experiment.

By analysing the values for the groupings in Table 8.1, we can interpret the following:

Single factor grouping

- The year does not affect the IGD of the generated front. The GD worsens from the year 2005 to 2008 ($p < 0.05$) but no significant change after that. The Epsilon (EP), like the IGD, worsens from the year 2005 to 2008 ($p < 0.01$) but no significant when moving to 2012. The Hypervolume (HV) is always better ($p < 0.01$) for the earlier year, this was also discussed in the visual analysis. The time required to complete runs increases significantly ($p < 0.01$) as we when we move on to later years.
- The change in test set type ($test_{RS}$, $test_{EA}$) entails testing on unseen users. By doing so, only the GD is affected ($p < 0.05$) and we see that the unseen users perform better. Likewise, the EP, which gives an indication about single points on the front, improves on average ($p < 0.01$).
- Considering the three different EAs, we see that both GDE3 ($p < 0.01$) and SPEA2 ($p < 0.05$) perform better on IGD, compared to NSGA-III. There is no significant difference between the performances of NGDE3 and SPEA2.

Interaction grouping

- Algorithm and test set type do not have significant interaction *i.e.* quality of the front, based on any quality indicator, is not affected significantly by using the EAs on unseen users. This entails that the system is robust against variation in users and is reliable for practical usage.
- The interaction between year and test set type is significant across the board. Lets break this down:
  - GD is always better for the year 2005 with $test_{EA}$ ($p < 0.01$) when a different year or $test_{RS}$ is considered.
  - IGD is better for the year 2005 with $test_{EA}$ in most cases, *except when* year 2012 with $test_{RS}$ is considered.
  - Epsilon (EP) is usually worse at a later year and in most cases worse on $test_{RS}$.
  - Hypervolume (HV) shows significance in changes but there is no stable trend, except for the fact that 2012 is always worse when compared to 2005 or 2008.
  - Time required for runs is always significantly lower ($p < 0.01$) for the earlier year. For some reason, during the 2008 experiment run, changing the test set to $test_{EA}$ significantly reduced the run time. We are not aware as to why this happened.

8.2.3. Front Objectives and Solutions Visualizations

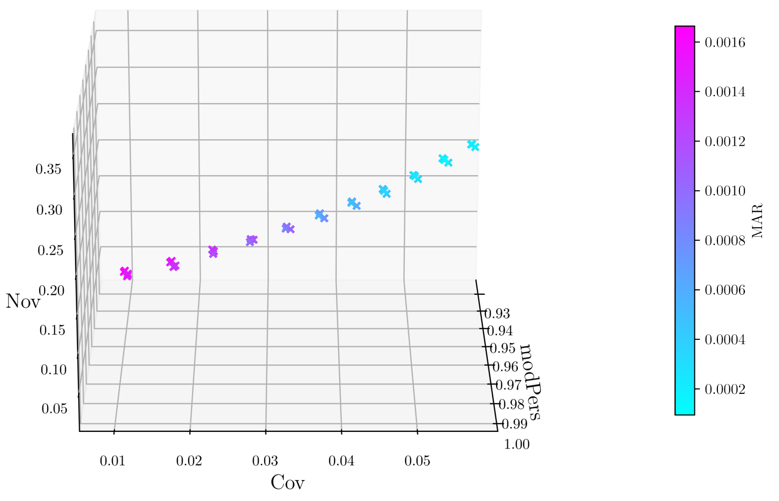
To analyse the quality of the Pareto front to spot relations, EA practitioners resort to visualising the solutions on the front. These relations might not be very clear through the numerical indicators. In Figure 8.8 we see the Pareto fronts generated at the end of the runs of different EAs plotted on a parallel coordinate plot. The fronts are divided by the year and test set ($(year, test_X)$), the divisions we call *unique problems*.

Animation 8.7, gives an example of how this front looks in a multidimensional space. It shows a '4D' visualization where Nov, Cov, and modPers form the 3 dimensional Euclidean space, and MAR is on the 4th dimension, denoted by the colour of the markers.

In Figure 8.8, we can see that the trend of different objective values of the front across different years is similar, although there are some small differences that can be identified if we look too closely. Note that the value range of the objectives is different every year because the problem changes. We also see that the quality of the front is consistent for the same year when we move from $test_{RS}$ to $test_{EA}$ *i.e.* the front does not visually when the users are different.

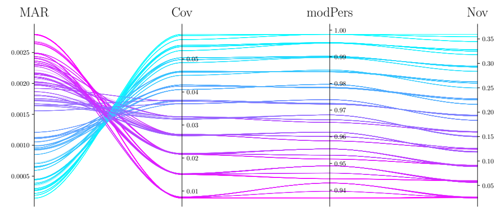
We can also visualise how the EAs weigh different recommenders to reach these objective fronts. In Figure 8.9, we can see the relative weights interpreted for the different unique problem sets. The unique problems have the same colourmaps as their counterparts in Figure 8.8. These interpreted weights for each recommender are obtained through the equation 8.1. We see that the trend of weighting different models per year and depending on the test users ($test_{RS}$, $test_{EA}$) is not the same and shows striking differences in many cases. That being said, we know that the Pareto front is not observably different, besides in terms of hypervolume. This means the EAs are able to deal with changing contexts that affect the RSs and weigh the RSs such that they still generate near-optimal fronts that are consistent across seen and unseen users.

$$\bar{wt}(R_X) = \frac{wt(R_X)}{\sum_i wt(R_i)} \tag{8.1}$$

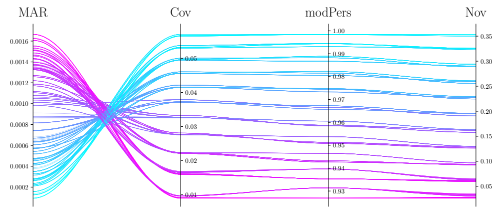


.

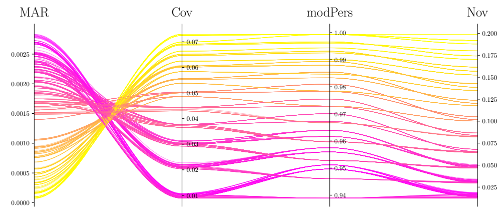
Figure 8.7: **[Embedded Animation]** Example *'4D'* plot for Pareto fronts visualization in objective-space for year 2005 on the $test_{EA}$. Note (i) the 4^{th} dimension is colour. (ii) To see the animation open the PDF in native viewer and not on a web browser (iii) To interact with controls enable PDF forms for the PDF viewer.



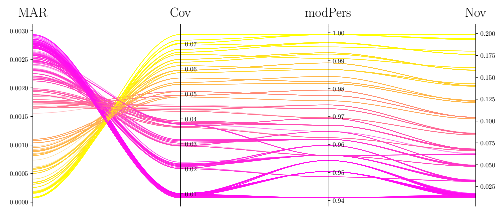
(a) Year 2005, $test_{RS}$



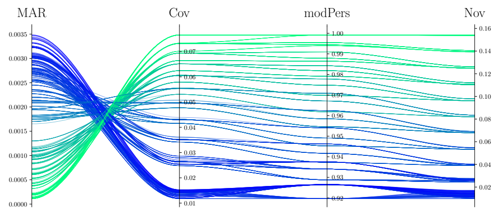
(b) Year 2005, $test_{EA}$



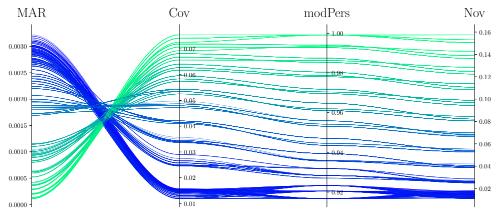
(c) Year 2008, $test_{RS}$



(d) Year 2008, $test_{EA}$

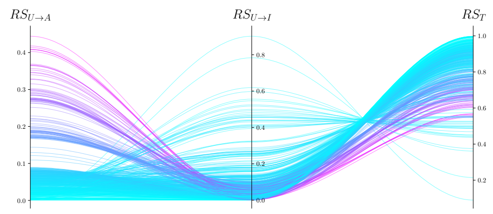


(e) Year 2012, $test_{RS}$

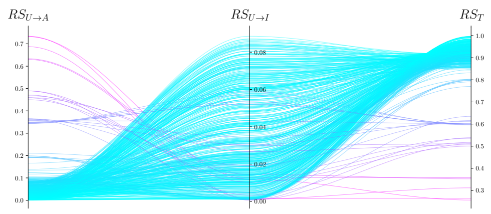


(f) Year 2012, $test_{EA}$

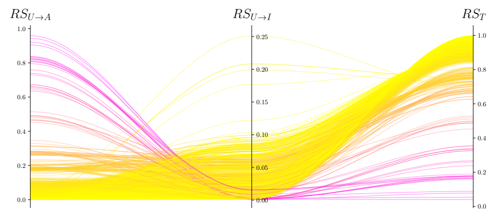
Figure 8.8: Unions of objective-space positions of Pareto front solutions found by different EAs (NSGA-III, GDE3, SPEA2) across different runs and test sets, split by year and year and test-set type. Fronts with the same year shown with the same colour-map to check consistency of quality. *Note that the axes are normalised (check axis values)*.



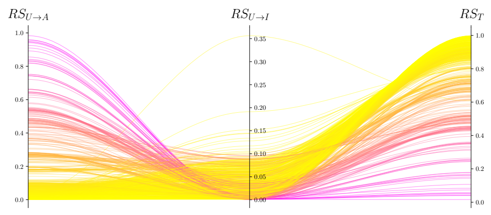
(a) Year 2005, $test_{RS}$



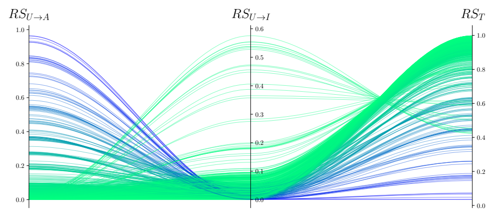
(b) Year 2005, $test_{EA}$



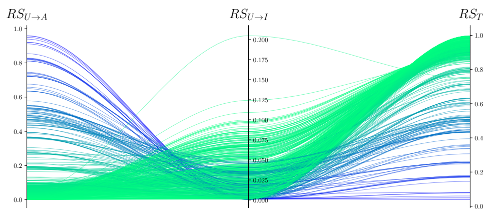
(c) Year 2008, $test_{RS}$



(d) Year 2008, $test_{EA}$



(e) Year 2012, $test_{RS}$



(f) Year 2012, $test_{EA}$

Figure 8.9: Union of Recommender reranking weights interpreted from the genotype of Pareto fronts found by different EAs (NSGA-III, GDE3, SPEA2) across different runs and test sets, split by year and year and test-set type. Weights from the same year shown with the same colour-map to check the consistency of quality. *Note (i) the axes are normalised (check axis values) (ii) lower weight means smaller rank and placed higher in recommendation list i.e. lower means more influence*.

9 Conclusions

With the quantitative results presented in the previous chapter, we can now summarise the findings from the experiemnts to answer the research questions that were posed in Chapter 1, Introduction. We will also separately list the contributions that were made in the process of doing this research.

9.1. Findings

Let us categorise the conclusions we reach according to the research questions to answer the critical scientific and practical concerns in a targeted fashion.

RQ1: How to perform late-stage recommendation fusion, for large-scale user data which gives a set of near-optimal options over user-centric recommendation quality metrics?

We have presented a methodology for the recommender framework, well-rooted in theory, whose outputs have been qualitatively and quantitatively verified using a robust experimental design. We demonstrated the methodology on one of the largest public music listening histories dataset. The large-scale data in combination with the experimental design presented various real-world scenarios. We made the demonstration particularly for the task of music recommendation, although this strategy is suitable for a wide array of application domains that use large-scale data.

RQ2: How to make the developed recommendation framework work reliably with large-scale user data, in a manner that can be used in a practical industry scenario?

We used state-of-the-art engineering solutions to make this framework work well with large-scale data. This involves making Recommender Systems work in the format of distributed computing to generate recommendations for large volumes of data. The EA pipeline is also parallelised and well suited for performance-tuned compute environments. The implementation has been made open-source for industry use and can be configured and modified easily.

RQ3: To what extent do variations in the user base and their listening behaviour, through time, affect the delivered recommendations? and how?

The individual Recommender Systems perform very differently under different temporal contexts. This is because of changing user base and user behaviour patterns. The Recommender Systems fit on these different data distributions differently to recommend items, and thus perform variably on user-centric metrics. The recommender framework with Evolutionary Algorithms is able to fuse the recommendations from these Recommender Systems, showing high variability across different temporal contexts, to generate near-optimal fronts.

These fronts, when analysed over different EAs with quality indicators such as the Generational Distance (GD) and Inverse Generational Distance (IGD) computed within their own temporal context, perform quite well. This means the year does not affect the performance of these metrics. It performs poorly on the Hypervolume indicator (the volume of objective space that is dominated) across temporal contexts because the combined capabilities of these recommenders do not warrant consistent reach through time for the nondominated region *i.e.* as years go on, the recommenders we use cannot maximise the objectives in the same manner as previous years.

This implies that although this combination of Recommender Systems and Evolutionary Algorithm is a good strategy to generate fusion solutions. However, practitioners should be cautious while deploying these methods, as they are not a cure-all for the problems that plague the domain of Recommender Systems.

9.2. Contributions

Enabling the scientific rigour and practicality of this research requires working on modules that go beyond the main storyline of generating reliable Pareto sets for weighing different recommenders. Here we list our contributions that relate to this thesis report.

- The recommendation framework methodology under study that combines the disciplines of Recommender Systems and Evolutionary Algorithms.
- Making the Music Listening Histories Dataset (MLHD), more usable by translating its information to a scalable graph database.
- Analysis of MLHD to uncover user behaviour patterns, and subsequently a dataset division strategy for reliable validation of Recommender Systems and Evolutionary Algorithms on the dataset under changing contexts.
- Demonstrating the variation of different scalable recommenders on MLHD under different situations and making qualitative and quantitative comparisons in terms of user-centric metrics.
- Performance optimisations in various parts of the implementation which are the reason for being able to complete these experiments feasibly.

10 Limitations and Future Work

Now that we have concluded the study, we will take a critical look at the system. We will first point out the possible limitations and threats of the experimental design. Then we will look at what research endeavours can be undertaken in the future regarding improving the recommendation framework or applying it in different application domains.

10.1. Limitations and Threats

It is important to look at our own work critically and give others a peek into the limitations. The research about reliability of the recommendation framework ironically has notable threats that can make its operationalization problematic for real-world usage. There are several modular aspects that build up the system and there are things that can go wrong if it is adopted without any modifications. Here are some critical factors or choices we made to look out for in our study:

1. The sampling we threshold chose (*listening count* > 20) to consider which edges we put in the user interaction graph described in Section 3.2.1 may affect the results considerably. Setting the threshold lower means significantly increasing the size of the graph data, but making the information much more richer.

2. In Section 7.2.1 we discussed why we selected 200 (201 with MLHD_000.tar). We wanted make our 162 runs of the experiment finish in a feasible amount of time. Taking all 575 parts would mean more compute time for getting the recommendations from the three RSs and much longer fitness evaluation time for the EAs, especially for year 2012 which has the most amount of data. We tested the scalability of the framework by incrementally increasing the dataset size and making optimizations until the largest problem size required around 1hr of compute time. A threat may be that we do not see some problems which may lie *'beyond the horizon'* when we add more data.

3. In Chapter 7 we present our experimental design. This is how we take the levels for factors: from the available set of years (2005-2012) we take three years, use three EAs, and two sets of users ($test_{EA}$ and $test_{RS}$). Only after thorough analysis did we internally decided these levels for the factors were justified to understand the variations in the workings of this framework with sufficient depth. There could be a possibility that we miss nuanced changes from one year to the next.

4. We did not do *extensive* hyperparameter optimisation for the Collaborative Filtering for the recommenders $RS_{U\rightarrow I}$ and $RS_{U\rightarrow A}$. This was because of three reasons: (i) extensive optimization could in principle overfit for one of the test sets ($test_{RS}$) and we would get poor results, (ii) We wanted to see how the recommenders behave across years so we kept the same hyperparameters across years to make sure we see the evolution of these recommenders without human intervention, and (iii) In real-world, with larger amount of data, extensive hyperparameter optimisation for CF would not be possible and we would give sufficient amount of parameters and *trust* the system works (without a validation with scientific rigour possible).

5. We selected for metrics which are considered as user-centric according to literature, see Section 4.2 for more. These are Mean Average Recall, Novelty, Coverage and Personalisation. We in no way mean to suggest that these capture all objectives the user wants to be optimised. Although we would have liked to investigate more metrics, it would make make the fitness evaluation more compute intensive and increase the compute requirement for the experiments.

6. The metrics are also chosen because they are theoretically bounded between 0 and 1. We say that the practical bounds are very different. This means that optimisation of each objectives does not present equivalent growth in indicators. In this study we still consider the comparative growth/decline of Pareto front quality indicators so it should not be an issue because the .

7. Possible data imbalance *may* be at play in the experiments, we investigated the dataset to confirm this was not the case. We know how the distribution of users and their behaviour changes across years in MLHD. We split the dataset temporally according to the years. When we run metrics we average over all users to find the the metric at a particular list level. We investigated the standard deviation for the metrics at various list levels across years and it was minimal. Yet there could be some aspects the variations of users have that affect the metrics in unforeseen ways.

8. We selected the EAs' hyperparameters which affect the compute significantly, to optimally utilise the compute infrastructure (40 compute threads with 128 GiB memory). These were the population size and the number of generations. We get observably consistent performance across the board with differently sampled interactions and repetitions. That being said, changing compute infrastructure will change the results significantly and fitness evaluation may not even be possible with lower memory.

10.2. Future Improvements and Research Directions

The study of Evolutionary Algorithms and Multimedia Recommendations is an extremely niche domain. A majority of the audience of this thesis report will not have each disciplinary background to understand every aspect. Thus, here we will discuss possible future directions to improve various modules of this research in a categorical manner.

10.2.1. Data Usage

To improve how data is retrieved efficiently from ArangoDB, one can investigate the use of *arangodb-spark-datastore*[1]. This new connector for spark enables pushing the Spark DataFrame operations, such as filter, down to ArangoDB query used to fetch the data. This ensures only relevant data is fetched from ArangoDB and the entire collection of nodes and edges need not be fetched for performing any task related to recommendation. This functionality is currently under development by the engineers at ArangoDB, and is open to open-source contributions.

10.2.2. Dataset Enrichment And Hybrid Recommender

We investigated enriching the MLHD with features for songs using AcousticBrainz. The idea was to use these features to perform graph-based recommendation or a nearest-neighbour recommendation. A considerable amount of time during the thesis period was used to do this. This did not work out because of a few unforeseeable reasons and did not eventually converge into the main storyline of this thesis. We came up with a novel strategy to deal with the large volume of AcousticBrainz features to find the nearest neighbours. See more about this in Appendix C

10.2.3. Recommenders and Evolutionary Algorithms

We tried to incorporate recommenders in this study which have diverse goals. This was a study mainly about an interdisciplinary application and rigorous resting thereof, with a limited duration of 9 months. The study can benefit by going investigating the results with the same or similar experimental design with more diverse Evolutionary Algorithms.

More rigorous computation in terms of EAs would also be beneficial. Running more EAs for a longer time and seeing how each EA converges and which EAs are better for which .

[1] https://github.com/arangodb/arangodb-spark-datasource

10.2.4. User Experience

This study was about the scientific problem of MaOO in user content delivery. We do optimisation and tests to make sure we are modelling the problem with little biases and it would benefit the user. An MOO multimedia content delivery system has never actually been studied with real users. Thus, we do not know how well users will receive it and if they find it useful.

Extensive user studies are required to give researchers a direction in terms of what users desire regarding MOO multimedia content delivery systems. Do users find this system useful? Do they want to be presented sliders to indicate their preferences or give predefined categories which map to the the Pareto front? What metrics do users want to be optimised?

10.2.5. Beyond Music Recommendation

We decided to go with the MaOO problem of music recommendation because of its importance as we analysed it and the collective knowledge set and skills we have. It would be interesting to use the same methodology for large-scale recommendations for other application domains such as recommendations for movies of different genres, news articles with various topical objectives, online retail stores for relevance objectives, etc.

A

Appendix: User Behaviour Plots

In this Appendix lie the plots referenced in Chapter 3, Datasets, for understanding the changing distribution of $user \rightarrow item$ interactions. These plots are used to make explain that the changing data distribution across time needs to be taken into account while reliably validating that the recommendation framework under study works well.

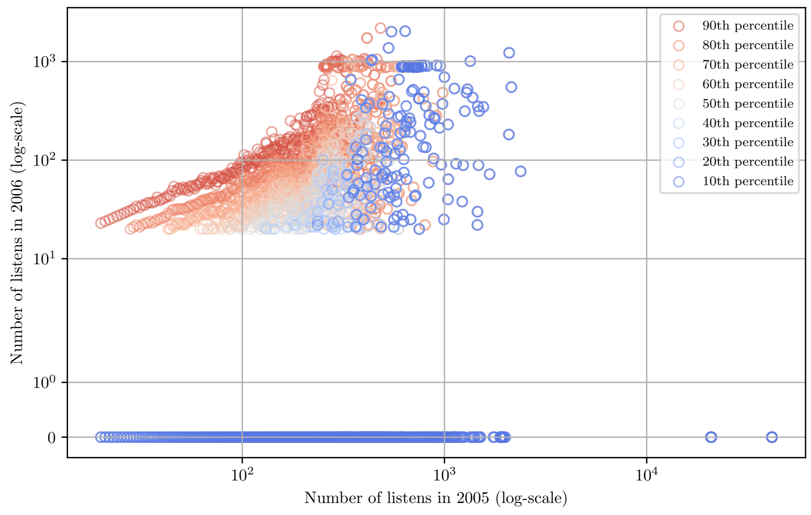


Figure A.1: Year **2005** percentile distribution of favourability of items/songs the following year, depending on listen counts across all $user \rightarrow item$ interactions. To be interpreted as, *"If a song is heard 'X' times in a year, It is heard for 'Y' times the following year"*. Note (i) Max (100^{th} percentile) and Min (0^{th} percentile) are removed. (ii) Points have alpha < 1.0 and are superimposed from 90^{th} to 10^{th} percentile (backwards to forwards). (iii) Observe the qualitative change of percentile distribution across years.

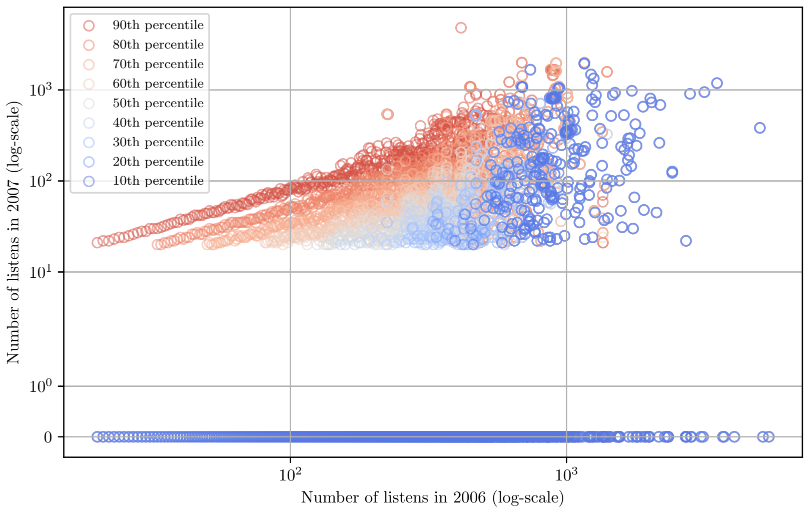


Figure A.2: Year **2006** percentile distribution of favourability of items/songs the following year, depending on listen counts across all $user \rightarrow item$ interactions. To be interpreted as, *"If a song is heard 'X' times in a year, It is heard for 'Y' times the following year"*. Note (i) Max (100^{th} percentile) and Min (0^{th} percentile) are removed. (ii) Points have alpha < 1.0 and are superimposed from 90^{th} to 10^{th} percentile (backwards to forwards). (iii) Observe the qualitative change of percentile distribution across years.

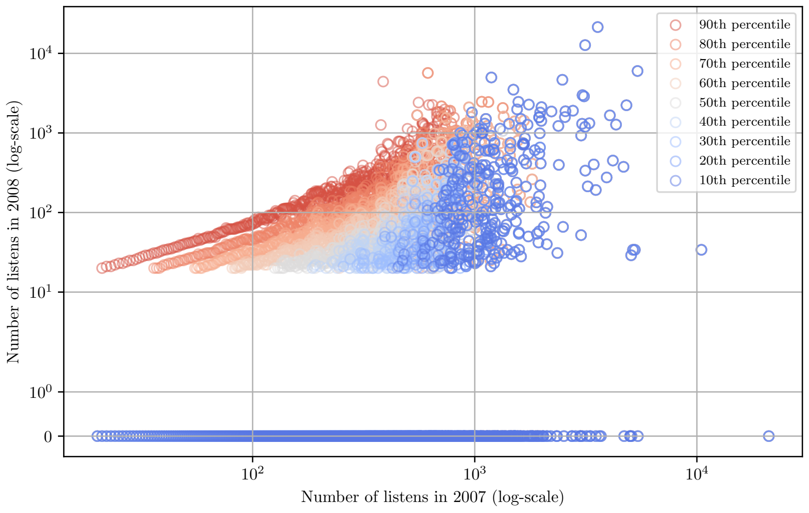


Figure A.3: Year **2007** percentile distribution of favourability of items/songs the following year, depending on listen counts across all $user \rightarrow item$ interactions. To be interpreted as, *"If a song is heard 'X' times in a year, It is heard for 'Y' times the following year"*. Note (i) Max (100^{th} percentile) and Min (0^{th} percentile) are removed. (ii) Points have alpha < 1.0 and are superimposed from 90^{th} to 10^{th} percentile (backwards to forwards). (iii) Observe the qualitative change of percentile distribution across years.

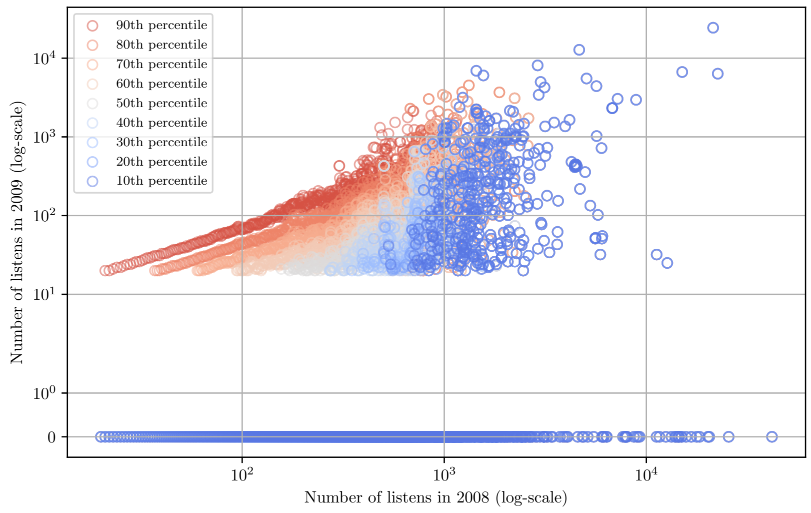


Figure A.4: Year **2008** percentile distribution of favourability of items/songs the following year, depending on listen counts across all $user \rightarrow item$ interactions. To be interpreted as, *"If a song is heard 'X' times in a year, It is heard for 'Y' times the following year"*. Note (i) Max (100^{th} percentile) and Min (0^{th} percentile) are removed. (ii) Points have alpha < 1.0 and are superimposed from 90^{th} to 10^{th} percentile (backwards to forwards). (iii) Observe the qualitative change of percentile distribution across years.

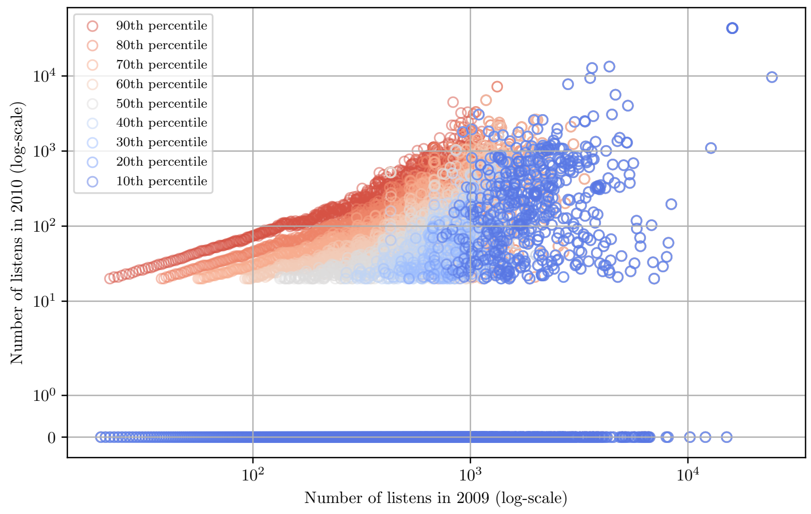


Figure A.5: Year **2009** percentile distribution of favourability of items/songs the following year, depending on listen counts across all $user \rightarrow item$ interactions. To be interpreted as, *"If a song is heard 'X' times in a year, It is heard for 'Y' times the following year"*. Note (i) Max (100^{th} percentile) and Min (0^{th} percentile) are removed. (ii) Points have alpha < 1.0 and are superimposed from 90^{th} to 10^{th} percentile (backwards to forwards). (iii) Observe the qualitative change of percentile distribution across years.

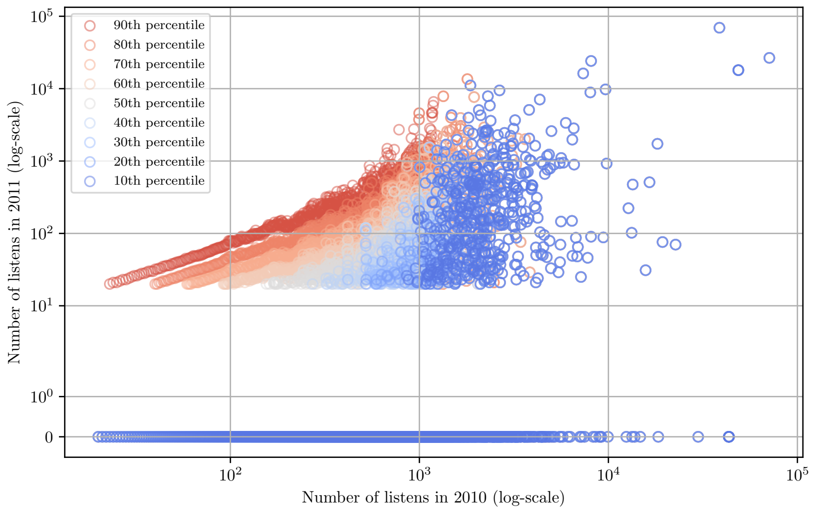


Figure A.6: Year **2010** percentile distribution of favourability of items/songs the following year, depending on listen counts across all $user \rightarrow item$ interactions. To be interpreted as, *"If a song is heard 'X' times in a year, It is heard for 'Y' times the following year"*. Note (i) Max (100^{th} percentile) and Min (0^{th} percentile) are removed. (ii) Points have alpha < 1.0 and are superimposed from 90^{th} to 10^{th} percentile (backwards to forwards). (iii) Observe the qualitative change of percentile distribution across years.

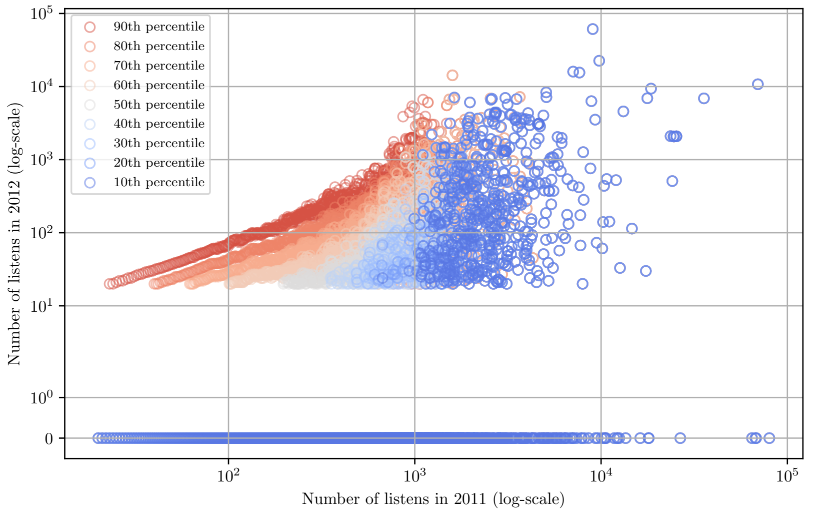


Figure A.7: Year **2011** percentile distribution of favourability of items/songs the following year, depending on listen counts across all $user \rightarrow item$ interactions. To be interpreted as, *"If a song is heard 'X' times in a year, It is heard for 'Y' times the following year"*. Note (i) Max (100^{th} percentile) and Min (0^{th} percentile) are removed. (ii) Points have alpha < 1.0 and are superimposed from 90^{th} to 10^{th} percentile (backwards to forwards). (iii) Observe the qualitative change of percentile distribution across years.

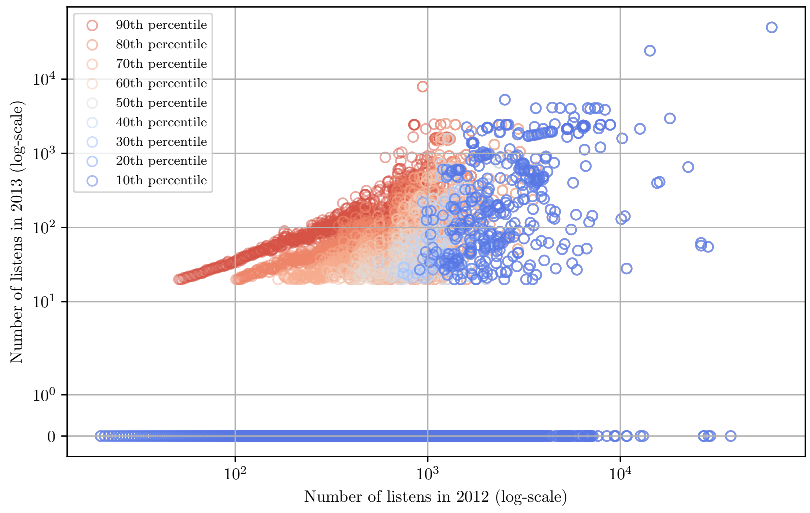


Figure A.8: Year **2012** percentile distribution of favourability of items/songs the following year, depending on listen counts across all $user \rightarrow item$ interactions. To be interpreted as, *"If a song is heard 'X' times in a year, It is heard for 'Y' times the following year"*. Note (i) Max (100^{th} percentile) and Min (0^{th} percentile) are removed. (ii) Points have alpha < 1.0 and are superimposed from 90^{th} to 10^{th} percentile (backwards to forwards). (iii) Observe the qualitative change of percentile distribution across years.

B Appendix: Experiments Tables

In this appendix lie the tables with the recorded observations for the quantitative analysis of the experiments. Table B.1 shows the Wilcoxon-Mann-Whitney Test's summary of p-values, and Table B.2 shows the Vargha and Delaney's A (VDA) and Cliff's delta (CD) for the significant entries in the earlier table.

The interpretation of these tables is presented in Chapter 8, Results, in Table 8.1 where it should be understandable by the general audience of this thesis report.

Grouping	***Comparison***		***Indicator***				
			GD	IGD	EP	HV	***Time***
Year	2005	2008	**0.0231873**	0.464511	**4.75E-05**	**2.53E-13**	**1.01E-18**
	2005	2012	0.12497213	1	5.94E-02	**1.01E-18**	**1.01E-18**
	2008	2012	1	1	1	**1.01E-18**	**1.01E-18**
Set Type	$test_{EA}$	$test_{RS}$	**0.0106946**	0.3230458	**0.0070405**	0.0780662	0.38379
Algorithm	GDE3	NSGA-III	1	**0.0085586**	1	1	1
	GDE3	SPEA2	1	1	1	1	1
	NSGA-III	SPEA2	1	**0.01567897**	1	1	0.0695531
Algorithm : Set Type	*No significant values*						
Year : Set Type	2005 $test_{EA}$	2005 $test_{RS}$	**0.0071075**	**0.00472601**	0.52006434	**4.40E-09**	1
	2005 $test_{EA}$	2008 $test_{EA}$	**0.0071146**	**0.02166169**	0.76795572	**4.47E-09**	**1.54E-14**
	2005 $test_{EA}$	2008 $test_{RS}$	**0.0071128**	**0.01414798**	0.76768652	**4.47E-09**	**4.54E-09**
	2005 $test_{EA}$	2012 $test_{EA}$	1	**0.009118159**	0.52439155	**4.47E-09**	**1.54E-14**
	2005 $test_{EA}$	2012 $test_{RS}$	**0.0058498**	1	**0.01145572**	**4.47E-09**	**1.54E-14**
	2005 $test_{RS}$	2008 $test_{EA}$	1	1	**6.63E-03**	**1.45E-05**	**1.54E-14**
	2005 $test_{RS}$	2008 $test_{RS}$	1	1	**2.81E-05**	**7.08E-03**	**4.54E-09**
	2005 $test_{RS}$	2012 $test_{EA}$	1	1	1	**4.48E-09**	**1.54E-14**
	2005 $test_{RS}$	2012 $test_{RS}$	0.6713244	0.1999437	**2.23E-08**	**4.48E-09**	**1.54E-14**
	2008 $test_{EA}$	2008 $test_{RS}$	0.5927924	1	1	5.45E-01	**2.61E-03**
	2008 $test_{EA}$	2012 $test_{EA}$	1	1	0.7763765	**1.54E-14**	**1.54E-14**
	2008 $test_{EA}$	2012 $test_{RS}$	1	1	0.3919277	**1.54E-14**	**1.54E-14**
	2008 $test_{RS}$	2012 $test_{EA}$	1	1	**0.006760535**	**4.54E-09**	**4.54E-09**
	2008 $test_{RS}$	2012 $test_{RS}$	1	1	0.579516357	**4.54E-09**	**4.54E-09**
	2012 $test_{EA}$	2012 $test_{RS}$	0.0064372	1	**2.63E-08**	**0.004798012**	1

Table B.1: Wilcoxon-Mann-Whitney (Wilcoxon Rank Sum Test) test, with Bonferroni correction, p-value statistics. Significant p-values ($p < 0.05$) are highlighted in **bold**.

Grouping	Comparison		Indicator									
			GD		IGD		EP		HV		Time	
			VDA	CD	VDA	CD	VDA	CD	VDA	CD	VDA	CD
Year	2005	2008	0.3512	-0.2976			0.7407	0.4814	0.4814	0.0833	0	-1
	2005	2012							1	1	0	-1
	2008	2012							1	1	0	-1
Set Type	$test_{EA}$	$test_{RS}$	0.3838	-0.2324			0.6226	0.2452				
Algorithm	GDE3	NSGA-III			0.3333	-0.3334						
	GDE3	SPEA2										
	NSGA-III	SPEA2			0.6560	0.3120						
Year : Set Type	2005 $test_{EA}$	2005 $test_{RS}$	0.2222	-0.5556	0.2140	-0.5720			0	-1		
	2005 $test_{EA}$	2008 $test_{EA}$	0.2222	-0.5556	0.2469	-0.5062			0	-1	0	-1
	2005 $test_{EA}$	2008 $test_{RS}$	0.2222	-0.5556	0.2373	-0.5254			0	-1	0	-1
	2005 $test_{EA}$	2012 $test_{EA}$			0.2277	-0.5446			1	1	0	-1
	2005 $test_{EA}$	2012 $test_{RS}$	0.2181	-0.5638			0.7654	0.5308	1	1	0	-1
	2005 $test_{RS}$	2008 $test_{EA}$					0.7778	0.5556	0.1111	-0.7778	0	-1
	2005 $test_{RS}$	2008 $test_{RS}$					0.8765	0.7530	0.2222	-0.5556	0	-1
	2005 $test_{RS}$	2012 $test_{EA}$							1	1	0	-1
	2005 $test_{RS}$	2012 $test_{RS}$					0.9767	0.9534	1	1	0	-1
	2008 $test_{EA}$	2008 $test_{RS}$									0.2016	-0.5968
	2008 $test_{EA}$	2012 $test_{EA}$							1	1	0	-1
	2008 $test_{EA}$	2012 $test_{RS}$							1	1	0	-1
	2008 $test_{RS}$	2012 $test_{EA}$					0.2222	-0.5556	1	1	0	-1
	2008 $test_{RS}$	2012 $test_{RS}$							1	1	0	-1
	2012 $test_{EA}$	2012 $test_{RS}$					0.9753	0.9506	0.2222	-0.5556		

Table B.2: Vargha and Delaney's A (VDA) and Cliff's delta (CD) statistics for pairs of factors with quantitatively significant ($p < 0.05$) Wilcoxon Rank Sum Test p-value.

C

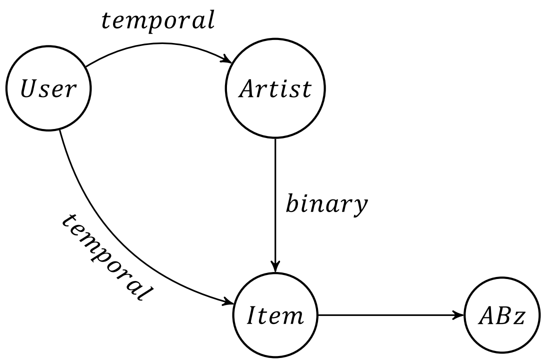
Appendix: AcousticBrainz Integration and GPU-optimised Nearest Neighbours

A significant portion of the thesis timeline involved the investigation of enriching MLHD with features from another dataset. The idea was to use these features for graph-based recommendation or a nearest neighbour strategy. These endeavours failed because of two reasons (i) lack of time and (ii) the sparsity of annotations when we use only 200 MLHD pieces with *listen counts* ≥ 20. Nevertheless, this work can be useful for the next wave of research in this domain. The contributions made here relate to processing a large volume of music descriptors (features) to annotate the MLHD and then find the approximate nearest neighbours for items with features.

C.0.1. AcousticBrainz

The AcousticBrainz project[1], is under the MetaBrainz Project, talked about in Chapter2. It is an open community-driven database of music descriptors of music files using a multitude of Music Information Retrieval Algorithms [90]. It provides high and low level information about music which also has MBIDs. The feature extraction for the same is primarily done using on Essentia[2], an open-source command-line music feature extractor[91].

Using the AcousticBrainz API, one can obtain music features in a JSON[3] format. Note that although the response has a fixed schema, not all features are recorded for different songs because of the natural evolution of the services and features being added at different points of time.



$$\begin{tikzcd}
User \arrow[r, bend left, "temporal"] \arrow[dr, bend right, "temporal"'] & Artist \arrow[d, "binary"] & \\
& Item \arrow[r] & ABz
\end{tikzcd}$$

Figure C.1: Explanation graph extracted from MLHD's user scrobbles and annotated with AcousticBrainz (ABz) features.

Furthermore, it should be noted from a Data Science perspective that although music descriptors from AcousticBrainz can give information about *how the music sounds*, these features are not *always*

[1]https://acousticbrainz.org/
[2]https://essentia.upf.edu/streaming_extractor_music.html
[3]JSON (JavaScript Object Notation) is a human-readable textual data-interchange format.

reliable when we talk about ground truth. In 2020, Chris Mostert, *et al.* in a Masters thesis and a subsequent paper demonstrated that certain descriptors in AcousticBrainz show unexpected behaviour on unseen, *'in the wild'*, data [92, 93]. The high-level features may have some conceptual problems ,and all features in general were susceptible to problems because of the different contexts under which the features were extracted.

The AcousticBrainz team is working on annotating MLHD. Their code[4] for fetching AcousticBrainz features for recording MBIDs present in MLHD.

The use of JSONs leads to the raw dataset size to be 257.7 GiB. We clean the dataset (2 million records) and keep only 1366 feature columns that are common in most records. The partitioned and compressed dataset in ORC[5] format sums up to 44.8 GiB in size.

C.1. GPU-Optimised Nearest Neighbours Search (GoNN)

The many system limitations while querying k-nearest neighbours on large datasets (millions of samples) with samples having thousands of features, led to the development of a custom solution. **G**PU-**o**ptimised **N**earest **N**eighbours Search or *GoNN*, conveniently pronounced as *'gone'*, combines efficient feature transformation and a distributed randomised directed search algorithm for approximate nearest neighbour search. The development of this algorithm has the following goals with the following goals:

- Reduce the number of compute and I/O operations required for finding a sortied list of nearest neighbours.
- Near-constant space requirement of main memory and graphics memory.
- Reduce trivial calculations; because $dist_{x_i \to x_j} = dist_{x_j \to x_i}$.
- Guarantee eventual convergence to the true solution, even if the approach is grounded in approximation.

We developed an algorithm during this study using custom CUDA kernels. These kernels were developed in Python using *numba's*[6] open source just-in-time (JIT) CUDA compiler. It enables writing CUDA kernels in Python, instead of traditional C or C++. This is done to remain close to the Python aspect remaining code of the developed framework [57, 94].

C.1.1. Random Hyperplane LSH

This is the first step of GoNN. As GPUs have less memory, we need to use some tricks to make large datasets fit on a single GPU device. Binary hashing using random hyperplanes with bias helps us with this. This technique preserves the Euclidean distance between data points well, given a large number of hashing bits (hyperplanes) [95]. It also outperforms many other state of the art hashing techniques. The explanation of this is given in Figure C.2. We see the provided data points that are uniformly distributed in a 2D space. The hypervolumes between Every data point has a binary array where each bit corresponds to a hyperplane's hash bit. The bit is 0 or 1 depending on which side of the hyperplane the point is on. In the figure, points with the same binary sequence are labelled the same. It would be natural to realise that increasing the number of hyperplanes increases the resolution of the hashing.

It is important to note why we are using a binary representation. We reduce the size of the features significantly by storing a string of 32 hash bits as an integer. This makes the computation on the GPU space-efficient. Moreover, computing Hamming distance as the distance metric is computationally much faster than computing the traditional Euclidean distance.

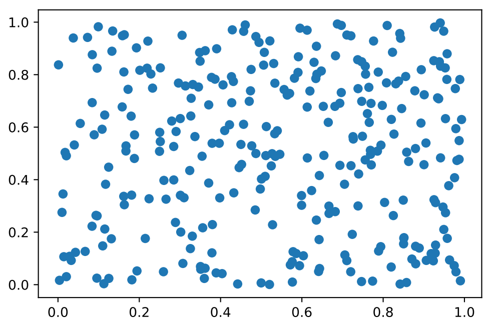
For a small theoretical demonstration, we will compare the CPU clock cycles for Euclidean distance of the floating point feature array compared to the equivalent Hamming distance with efficiently stored random hyperplane LSH bits. The formula for Euclidean distance is given by equation C.1. The algorithm for Hamming distance is provided in Algorithm 1.

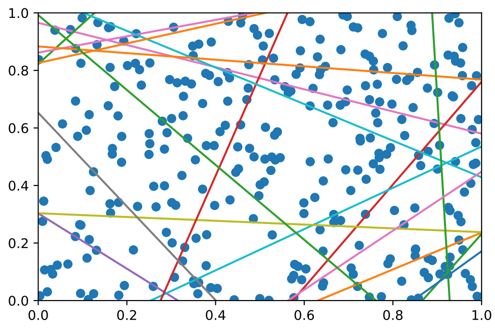
$$d(p,q) = \sqrt{(p_0 - q_0)^2 + (p_1 - q_1)^2 + \dots + (p_n - q_n)^2} \tag{C.1}$$

[4]Script on GitHub: https://github.com/MTG/acousticbrainz-labs/tree/master/mlhd

[5]The Optimized Row Columnar (ORC) file format provides a highly efficient way to store Hive data. An ORC file contains groups of row data called stripes, along with auxiliary information in a file footer. At the end of the file a postscript holds compression parameters and the size of the compressed footer.

[6]http://numba.pydata.org/


(a) 2D uniformly distributed data


(b) LSH-bias hyperplanes superimposed on the data

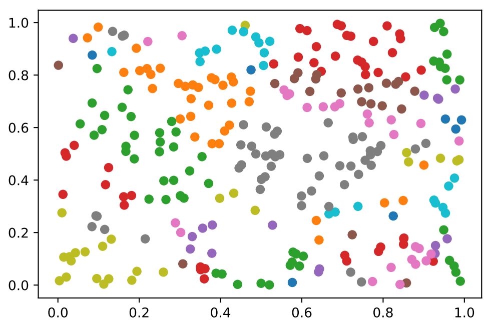

(c) Resulting labels after LSH, *items in the same hypervolume have the same labels/colour*

Figure C.2: Caption

We will take the operation clock cycle values from the instruction set for *AMD Ryzen 5000, Zen3 micro-architecture*[7], which is a 64bit CPU [96]. Note that the comparison will miss a lot of nuances around different architectures, the presence of hyper-threading, parallelization, etc, but will still be useful while comparing properties that will affect throughput and latency. The comparison is given in Table C.1. We see that Hamming distance has much lower CPU latency and uses faster operations in general.

Euclidean distance for m float features				Efficient Hamming distance for n int (n*32 bits)			
operation	required count	Op-count	latency	operation	required count	Op-counts	latency
FSQRT	1	1	25	XOR	n	1	1
ADD/SUB	2m-1	1	6 to 7	AND	5×n	1	1
FMUL	m	3	-	SHR	5×n	1	1
				ADD/SUB	6×n	1	1

Table C.1: Clock cycles comparison for Euclidean distance with floats vs hamming distance on integer bits.

The Population Count (popcount) algorithm is used for counting the flipped bits (crossed hyperplanes) and the Hamming distance. It is taken from Henry S. Warren's book *"Hacker's Delight"*[97]. It follows a divide-and-conquer strategy, shown in Figure C.3, in which the original problem (summing 32 bits) is divided into two problems (summing 16 bits), which are solved separately, and the results are combined (added, in this case). The strategy is applied recursively, breaking the 16-bit fields into 8-bit fields, and so on. For a 32 bit integer, it can be completed in $log_2(32) = 5$ steps. Each step takes logical operations, addition and subtraction only.

[7]Instruction table available here

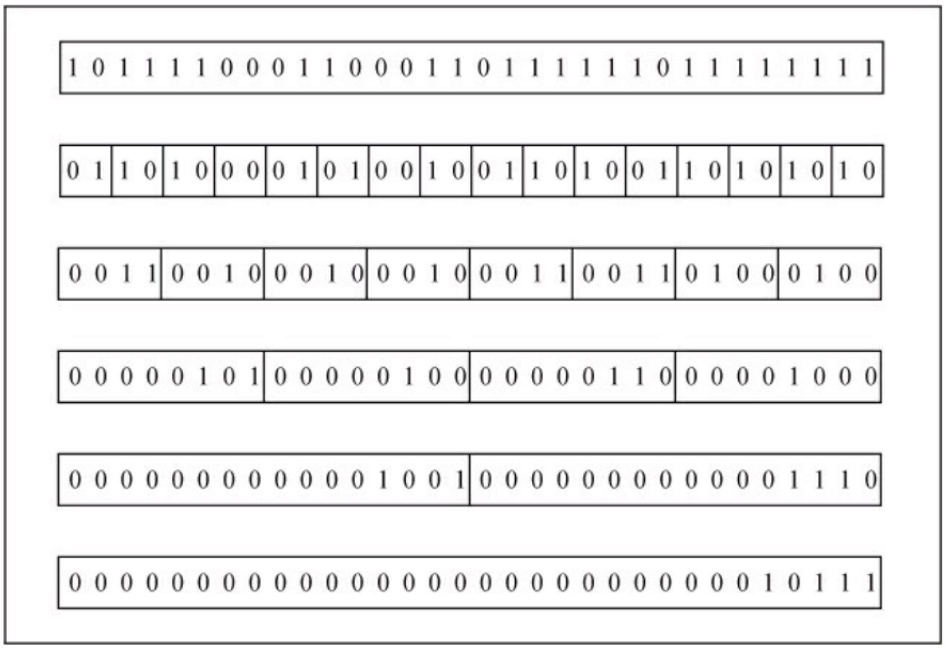


Figure C.3: Population count divide and conquer steps example [97]

Algorithm 1: Hamming distance: $dist(a, b)$

```
Input: 32 bit integer arrays $a$ and $b$ with every bit as LSH hash bit.
Result: hamming distance between two binary hashes, with hash value in each bit of two
        integer arrays.
Initialise len as size(a) = size(b);
Initialise d as 0
for $i \leftarrow 0$ to $len$ do
    // XOR to find flipped bits (crossing hyperplanes)
    x = a[i] $\oplus$ b[i]
    // popcount: counting number of 1s
    x -= x » 1 & 1431655765
    x = (x & 858993459) + (x » 2 & 858993459)
    x = x + (x » 4) & 252645135
    x += x » 8
    x += x » 16
    d += x & 63
end
return d
```

C.1.2. Search Algorithm

The search algorithm as a whole uses the concepts of Distributed Algorithms and Curriculum Learning [98]. In every iteration, every data point (i) searches for its nearest found neighbours in a pseudo-random manner and then updates the known neighbours and distance to neighbours, then (ii) It looks at its neighbour's set of neighbours and memorises them for its next search. These two steps are called the *search step* (Algorithm 3) and *update step*(Algorithm 4) respectively. Both steps are run by several processors for different data points in parallel.

The search step is followed by an update step in every iteration and after starting each step, the CPU waits for all computations of the GPU to finish. Algorithm 2 gives the overview of the complete nearest-neighbour search algorithm. Lets consider that we have N points and $H^{N \times h}$ contains the hashes of h integers containing the string of LSH bits. We take the total curriculum size C, which tells the number of points added in each curriculum and I iterations per curriculum.

Running the algorithm requires the following data structures:

- $search_i$ is an $N \times SEARCH\_SIZE$ integer array stores the index of items to look at in an iteration. Left half of it is filled with neighbours of neighbours in the $update()$ step. The right half maintains pseudo-random numbers which do not change. Every iteration for getting random numbers a different row's right half is used.
- $archive_i$ stores the row index of the nearest neighbour neighbours. These indexes are always in a sorted order (by distance) and when an item is inserted in the array, it is inserted in the correct place to keep it sorted.
- $archive_d$ stores the hamming distance to the neighbours in $archive_d$. It is used to find where to insert the neighbours in $archive_i$.

Algorithm 2: Overall GoNN flow: $GoNN()$

```
Input: $self_i$, $search_i$, $archive_i$
Result: updated $search_i$, with first half having neighbours of closest neighbours.
FLAG_tempering = False for curriculum_level← 0 to int(N/C) + 2 do
    rows = int(C ×(curriculum_level + 1));
    if rows ≥ROWS_MAX then
        rows = ROWS_MAX;
        FLAG_tempering = True;
    if FLAG_tempering is True then
        I = I ×5;
    for itr ← 0 to I do
        search();
        sync();
        update();
        sync();
    end
end
```

C.1.3. Observations

Because of the archive, the solution (quality of neighbours) cannot get worse until the new curriculum starts. Because of curriculum learning, each data point in the early curriculum can understand the distribution of data and find the solutions fast because the number of data points to search through is small. The points introduced in the later curriculum piggyback off the improvement of the earlier curriculum.

Points in the earlier curriculum always have a better solution than the later ones. Therefore, we use a tempering phase at the end which lasts longer than the set inter-curriculum iterations.

The distribution of the data does affect the performance. Data derived from a well spread out uniform distribution performs much better than that of several tightly clustered Gaussians. This is because the tight nature of the clusters entails fewer implied neighbour links to points which are in their own cluster. This means pseudo-random search takes longer to direct the solution.

C.2. GoNN Demonstration

Here you will see the demonstration of the GoNN algorithm applied to 10,000 data points with 200 features that come from three different distributions. The task in every case is the same. Find the nearest 20-neighbours. In every iteration, every data point looks at 200 other data points to add to its archive. In every case, GoNN is able to find the nearest 50 neighbours in the set. These runs do not last very long and take

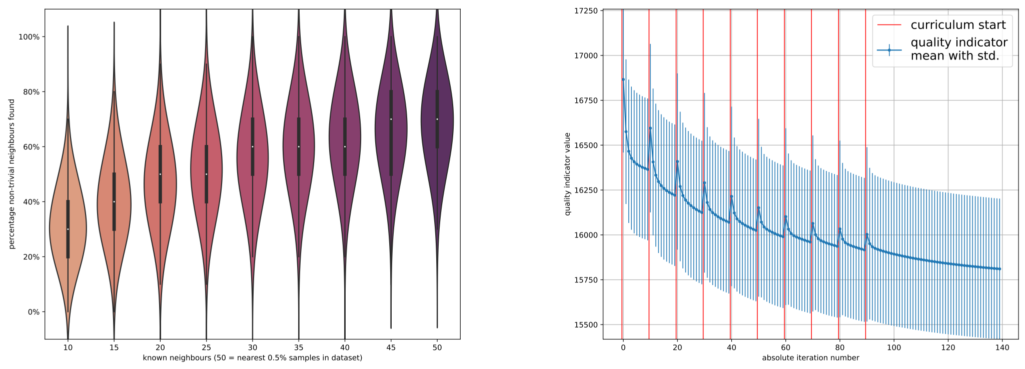


Figure C.4: *GoNN* on samples with uniformly distributed features.

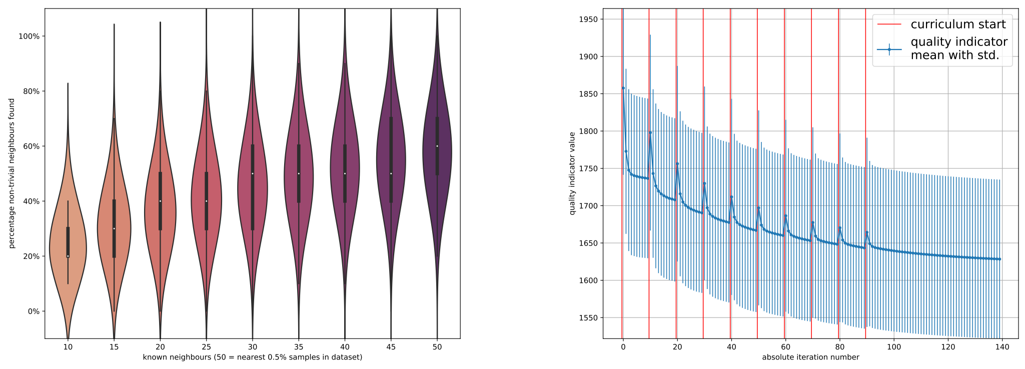


Figure C.5: *GoNN* on samples drawn from 3 dense and tight Gaussians.

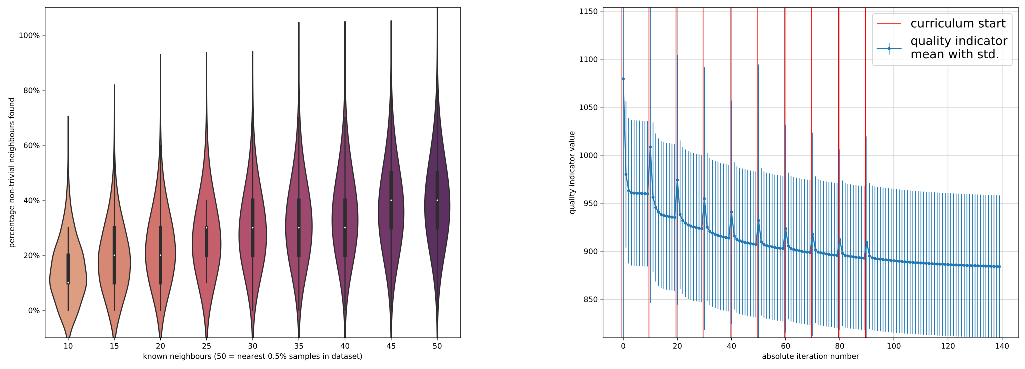


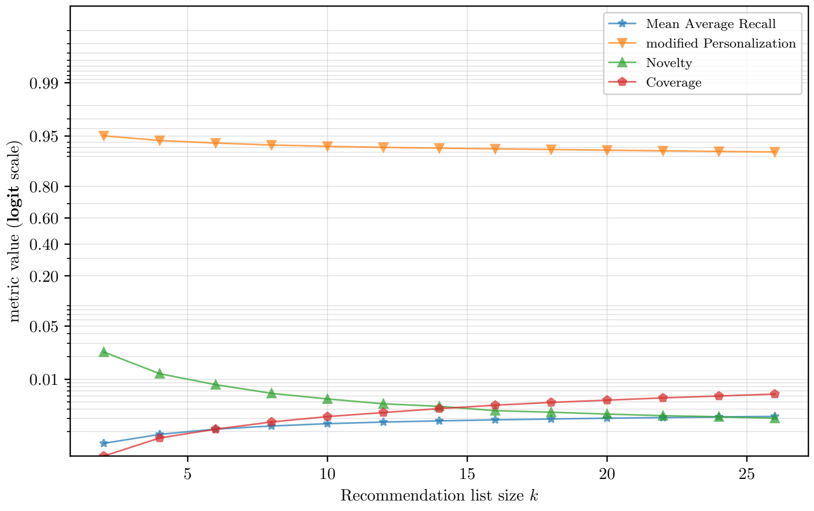
Figure C.6: *GoNN* on samples drawn from 10 dense and tight Gaussians.

```
Algorithm 3: Neighbour Search step: $search()$
Input: H, self$_i$, $search_i, archive_i, archive_d$
Result: updated $archive_i$ and $archive_d$ with nearer neighbours
for i← 0 to SEARCH_SIZE do
    if i < SEARCH_SIZE then
        s$_i = search_i[self_i, i] \mod C$;
    else
        x = (self$_i + ABSOLUTE\_ITERATION\_NUM) \mod C$;
        s$_i = search_i[x, i] \mod C$;
    end
    d = $dist(H[self_i, :], H[s_i, :])$
    // Now that we know the distance, find the insert location
    FLAG_new_neib = True;
    insert$_i = 0$
    while archive$_d[x, insert_i] <= dist$ & $insert_i < ARCHIVE\_SIZE$ do
        if archive$_i[x, insert_i] == search_i$ then
            FLAG_new_neib = False;
            break;
        insert$_i += 1$ end
        // Check if archive is full or neighbour already exists
        if $insert_i$ < ARCHIVE_SIZE ;
        & $FLAG\_new\_neib$ then
            if archive$_i[self_i, insert_i] == -1$ then
                // add in an empty space
                archive$_i[x, insert_index] = search_index$ $archive_d[x, insert_index] = dist$
                else
                    // insert item and move other items right
                    temp$_{i1}, temp_{d1} = archive_i[self_i, insert_i], archive_d[self_i, insert_i]$
                    archive$_i[x, insert_index] = s_i$
                    archive$_d[x, insert_index] = d$
                    itr = insert$_i + 1$
                    while itr < ARCHIVE_SIZE do
                        temp$_{i2}, temp_{d2} = archive_i[self_i, itr], archive_d[self_i, itr]$
                        archive$_i[self_i, itr] = temp_{i1}$
                        archive$_d[self_i, itr] = temp_{d1}$
                        temp$_{i1}, temp_{d1} = temp_{i2}, temp_{d2}$
                        itr += 1;
                    end
                end
            end
```

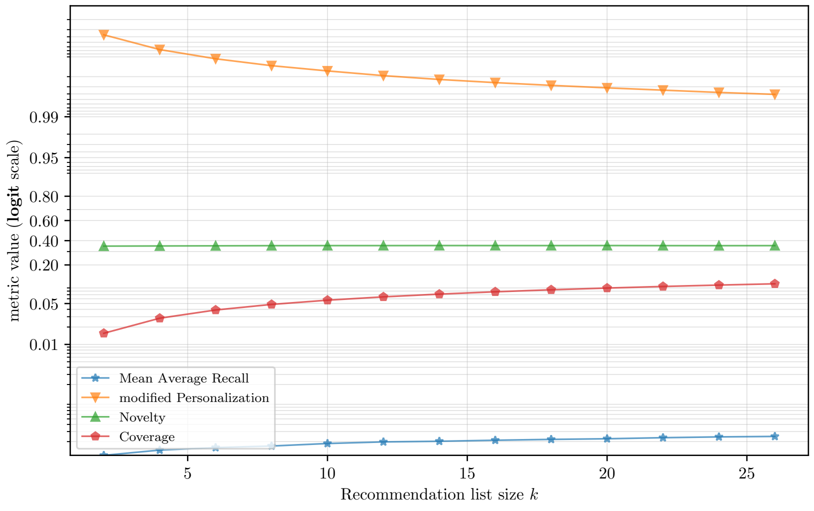
```
Algorithm 4: Update Step: $update()$
Input: $self_i$, $search_i$, $archive_i$
Result: updated $search_i$, with first half having neighbours of closest neighbours.
// number of neighbours of neighbours to copy
update_window = int(SEARCH_SIZE ÷2 ÷UPDATE_SUPPORT_NEIGHBOURS);
for i← 0 to UPDATE_SUPPORT_NEIGHBOURS do
    neib = archive$_i[self_i, i]$ for j ← 0 to update_window do
        search$_i[self_i, int(i \times update\_window + j)] = archive_i[neib, j]$
    end
end
```

D
Appendix: Individual Recommenders

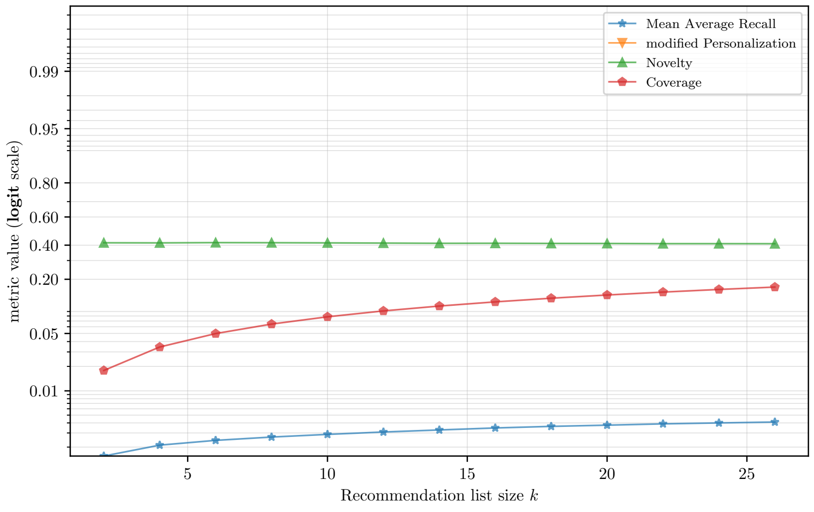
Following are the various RS's performance on $test_{RS}$ for chosen user-centric metrics at various various list sizes.



(a) $RS_{U \to I}$

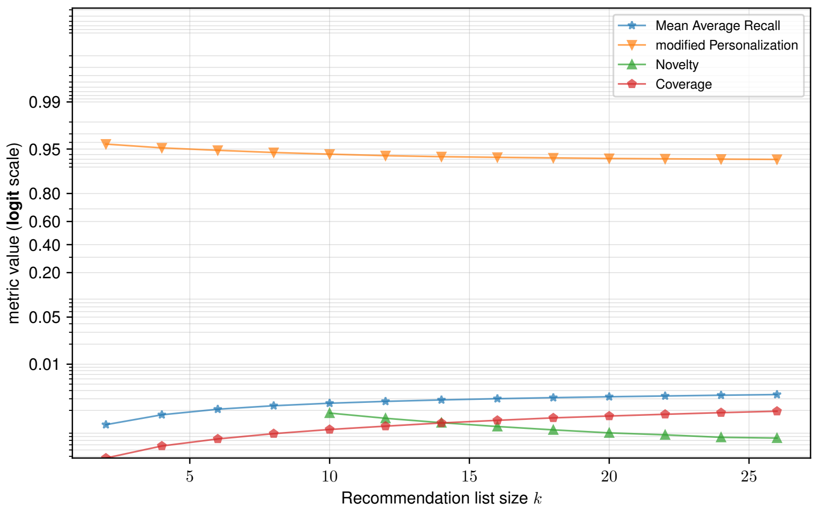


(b) $RS_{U \to A}$

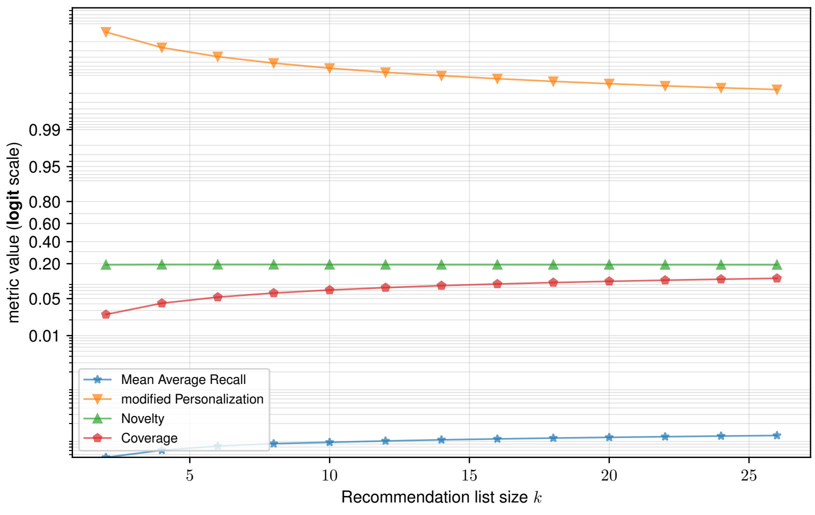


(c) RS_T

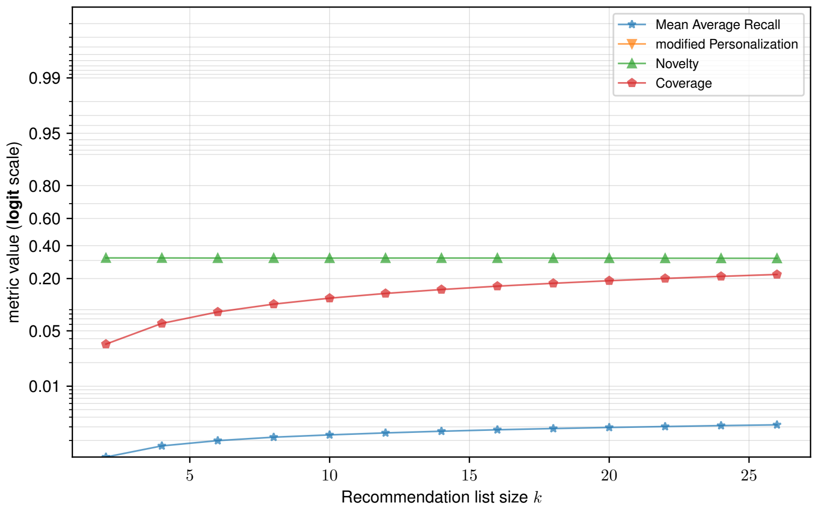
Figure D.1: Different recommenders' metrics achieved on $test_{RS}$, for year 2005



(a) $RS_{U \to I}$

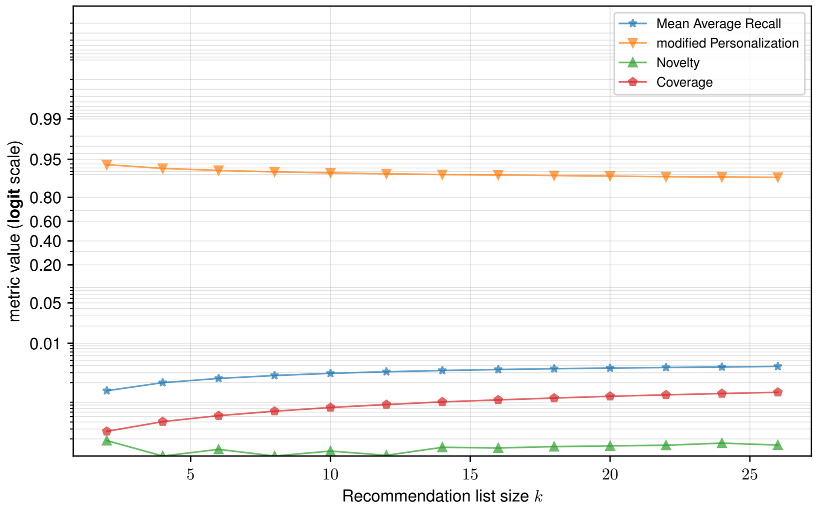


(b) $RS_{U \to A}$

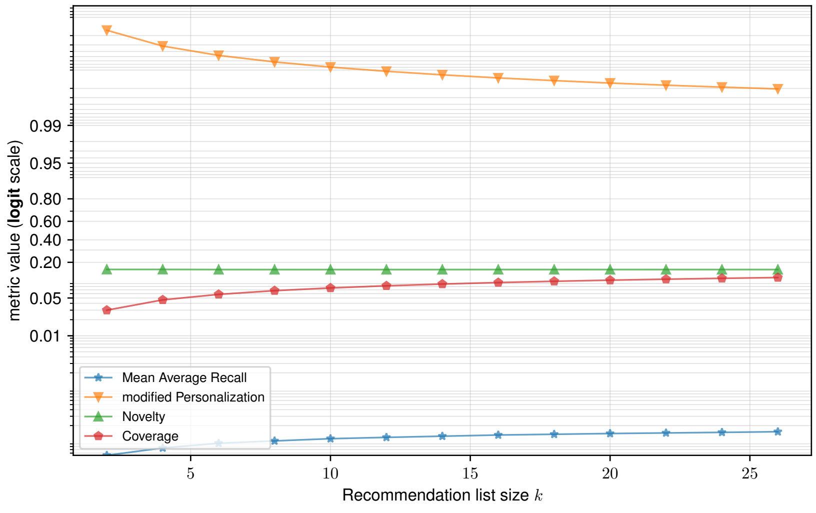


(c) RS_T

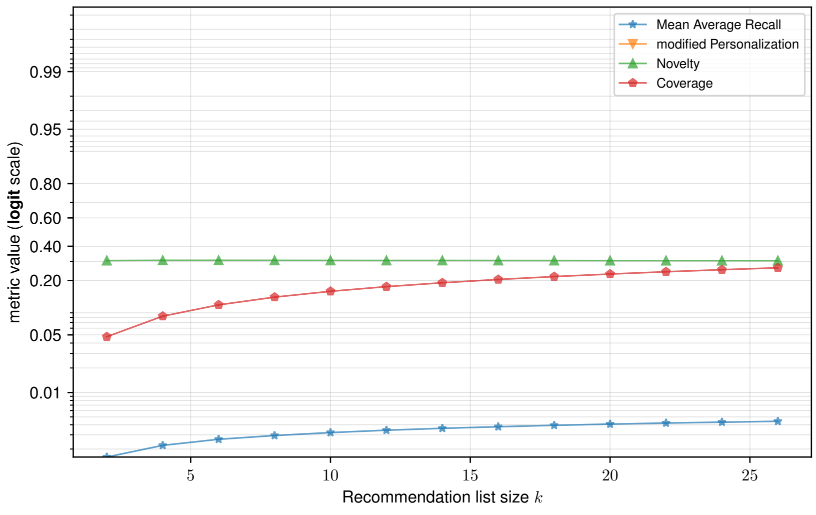
Figure D.2: Different recommenders' metrics achieved on $test_{RS}$, for year 2008



(a) $RS_{U \to I}$



(b) $RS_{U \to A}$



(c) RS_T

Figure D.3: Different recommenders' metrics achieved on $test_{RS}$, for year 2012

Bibliography

[1] Ning Yang, Min Wook Kang, Paul Schonfeld, and Manoj K. Jha. “Multi-objective highway alignment optimization incorporating preference information”. In: *Transportation Research Part C: Emerging Technologies* 40 (2014), pp. 36–48. ISSN: 0968090X. DOI: 10.1016/j.trc.2013.12.010.

[2] Xingjuan Cai, Zhaoming Hu, Peng Zhao, Wen Sheng Zhang, and Jinjun Chen. “A hybrid recommendation system with many-objective evolutionary algorithm”. In: *Expert Systems with Applications* 159 (Nov. 2020). ISSN: 09574174. DOI: 10.1016/j.eswa.2020.113648.

[3] Shanfeng Wang, Maoguo Gong, Haoliang Li, and Junwei Yang. “Multi-objective optimization for long tail recommendation”. In: *Knowledge-Based Systems* 104 (July 2016), pp. 145–155. ISSN: 09507051. DOI: 10.1016/j.knosys.2016.04.018.

[4] Eli Pariser. *The filter bubble: What the Internet is hiding from you*. Penguin UK, 2011.

[5] Markus Schedl, Sebastian Stober, Emilia Gómez, Nicola Orio, and Cynthia C S Liem. “User-Aware Music Retrieval”. In: 3 (2012). DOI: 10.4230/DFU.Vol3.11041.135.

[6] Markus Schedl, Arthur Flexer, and Julián Urbano. “The neglected user in music information retrieval research”. In: *Journal of Intelligent Information Systems* 41 (3 Dec. 2013), pp. 523–539. ISSN: 09259902. DOI: 10.1007/s10844-013-0247-6.

[7] Anja Nylund Hagen. *The Playlist Experience: Personal Playlists in Music Streaming Services*. Oct. 2015. DOI: 10.1080/03007766.2015.1021174.

[8] Sergey Volokhin and Eugene Agichtein. “Towards intent-aware contextual music recommendation: Initial experiments”. In: Association for Computing Machinery, Inc, June 2018, pp. 1045–1048. ISBN: 9781450356572. DOI: 10.1145/3209978.3210154.

[9] Sergey Volokhin and Eugene Agichtein. “Understanding music listening intents during daily activities with implications for contextual music recommendation”. In: vol. 2018-March. Association for Computing Machinery, Inc, Feb. 2018, pp. 313–316. ISBN: 9781450349253. DOI: 10.1145/3176349.3176885.

[10] Jiawei Chen, Hande Dong, Xiang Wang, Fuli Feng, Meng Wang, and Xiangnan He. “Bias and Debias in Recommender System: A Survey and Future Directions”. In: (Oct. 2020). URL: http://arxiv.org/abs/2010.03240.

[11] Axel Bruns. “Filter bubble”. In: *Internet Policy Review* 8.4 (2019).

[12] Kaisa Miettinen. *Nonlinear multiobjective optimization*. Vol. 12. Springer Science & Business Media, 2012.

[13] Anastasios Sextos and Panagiotis Mergos. *Multi-objective optimum selection of ground motion records with genetic algorithms*. 2018. URL: https://www.researchgate.net/publication/329870692.

[14] C-L Hwang and Abu Syed Md Masud. *Multiple objective decision making—methods and applications: a state-of-the-art survey*. Vol. 164. Springer Science & Business Media, 2012.

[15] George Mavrotas. “Effective implementation of the ε-constraint method in multi-objective mathematical programming problems”. In: *Applied mathematics and computation* 213.2 (2009), pp. 455–465.

[16] George Mavrotas and Danae Diakoulaki. “Multi-criteria branch and bound: A vector maximization algorithm for mixed 0-1 multiple objective linear programming”. In: *Applied mathematics and computation* 171.1 (2005), pp. 53–71.

[17] Indraneel Das and John E Dennis. “Normal-boundary intersection: A new method for generating the Pareto surface in nonlinear multicriteria optimization problems”. In: *SIAM journal on optimization* 8.3 (1998), pp. 631–657.

[18] James K Guest. “Imposing maximum length scale in topology optimization”. In: *Structural and Multidisciplinary Optimization* 37.5 (2009), pp. 463–473.

[19] Daniel Mueller-Gritschneder, Helmut Graeb, and Ulf Schlichtmann. “A successive approach to compute the bounded Pareto front of practical multiobjective optimization problems”. In: *SIAM Journal on Optimization* 20.2 (2009), pp. 915–934.

[20] Pradnya A. Vikhar. “Evolutionary algorithms: A critical review and its future prospects”. In: Institute of Electrical and Electronics Engineers Inc., June 2017, pp. 261–265. ISBN: 9781509004676. DOI: 10.1109/ICGTSPICC.2016.7955308.

[21] Bingrui Geng, Lingling Li, Licheng Jiao, Maoguo Gong, Qing Cai, and Yue Wu. “NNIA-RS: A multi-objective optimization based recommender system”. In: *Physica A: Statistical Mechanics and its Applications* 424 (2015), pp. 383–397.

[22] Bradley N Miller, Istvan Albert, Shyong K Lam, Joseph A Konstan, and John Riedl. “Movielens unplugged: experiences with an occasionally connected recommender system”. In: *Proceedings of the 8th international conference on Intelligent user interfaces*. 2003, pp. 263–266.

[23] James Bennett, Stan Lanning, et al. “The netflix prize”. In: *Proceedings of KDD cup and workshop*. Vol. 2007. Citeseer. 2007, p. 35.

[24] Ken Goldberg, Theresa Roeder, Dhruv Gupta, and Chris Perkins. “Eigentaste: A constant time collaborative filtering algorithm”. In: *information retrieval* 4.2 (2001), pp. 133–151.

[25] Chonghuan Xu. “A big-data oriented recommendation method based on multi-objective optimization”. In: *Knowledge-Based Systems* 177 (Aug. 2019), pp. 11–21. ISSN: 09507051. DOI: 10.1016/j.knosys.2019.03.032.

[26] Cai-Nicolas Ziegler. *Book-Crossing dataset*. URL: http://www2.informatik.uni-freiburg.de/~cziegler/BX/.

[27] Ken Goldberg, Theresa Roeder, Dhruv Gupta, and Chris Perkins. “Eigentaste: A constant time collaborative filtering algorithm”. In: *information retrieval* 4.2 (2001), pp. 133–151.

[28] Qingfu Zhang and Hui Li. “MOEA/D: A multiobjective evolutionary algorithm based on decomposition”. In: *IEEE Transactions on Evolutionary Computation* 11 (6 Dec. 2007), pp. 712–731. ISSN: 1089778X. DOI: 10.1109/TEVC.2007.892759.

[29] Nour El Islem Karabadji, Samia Beldjoudi, Hassina Seridi, Sabeur Aridhi, and Wajdi Dhifli. “Improving memory-based user collaborative filtering with evolutionary multi-objective optimization”. In: *Expert Systems with Applications* 98 (May 2018). <br/>, pp. 153–165. ISSN: 09574174. DOI: 10.1016/j.eswa.2018.01.015.

[30] Matei Zaharia, Reynold S Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, Shivaram Venkataraman, Michael J Franklin, et al. “Apache spark: a unified engine for big data processing”. In: *Communications of the ACM* 59.11 (2016), pp. 56–65.

[31] Thierry Bertin-Mahieux, Daniel PW Ellis, Brian Whitman, and Paul Lamere. “The million song dataset”. In: (2011).

[32] Gabriel Vigliensoni and Ichiro Fujinaga. “The Music Listening Histories Dataset.” In: *ISMIR*. 2017, pp. 96–102.

[33] Brian Brost, Rishabh Mehrotra, and Tristan Jehan. “The music streaming sessions dataset”. In: *The World Wide Web Conference*. 2019, pp. 2594–2600.

[34] Oscar Celma. *Music Recommendation and Discovery in the Long Tail*. Springer, 2010.

[35] Markus Schedl. “Leveraging microblogs for spatiotemporal music information retrieval”. In: *European Conference on Information Retrieval*. Springer. 2013, pp. 796–799.

[36] David Hauger, Markus Schedl, Andrej Košir, and Marko Tkalcic. “The million musical tweets dataset: What can we learn from microblogs”. In: *Proc. ISMIR*. 2013, pp. 189–194.

[37] Eva Zangerle, Martin Pichl, Wolfgang Gassler, and Günther Specht. “# nowplaying music dataset: Extracting listening behavior from twitter”. In: *Proceedings of the first international workshop on internet-scale multimedia management*. 2014, pp. 21–26.

[38] Markus Schedl. “The lfm-1b dataset for music retrieval and recommendation”. In: *Proceedings of the 2016 ACM on International Conference on Multimedia Retrieval*. 2016, pp. 103–110.

[39] Aaron Swartz. “Musicbrainz: A semantic web service”. In: *IEEE Intelligent Systems* 17.1 (2002), pp. 76–77.

[40] Diogo Fernandes and Jorge Bernardino. “Graph Databases Comparison: AllegroGraph, ArangoDB, InfiniteGraph, Neo4J, and OrientDB.” In: *DATA*. 2018, pp. 373–380.

[41] Zhichao Cao, Siying Dong, Sagar Vemuri, and David HC Du. “Characterizing, modeling, and benchmarking rocksdb key-value workloads at facebook”. In: *18th {USENIX} Conference on File and Storage Technologies ({FAST} 20)*. 2020, pp. 209–223.

[42] Siying Dong, Mark Callaghan, Leonidas Galanis, Dhruba Borthakur, Tony Savor, and Michael Strum. “Optimizing Space Amplification in RocksDB.” In: *CIDR*. Vol. 3. 2017, p. 3.

[43] Aaron Clauset, Cosma Rohilla Shalizi, and Mark EJ Newman. “Power-law distributions in empirical data”. In: *SIAM review* 51.4 (2009), pp. 661–703.

[44] *DataFrameStatFunctions*. URL: `https://spark.apache.org/docs/2.4.7/api/scala/index.html#org.apache.spark.sql.DataFrameStatFunctions`.

[45] Michael Greenwald and Sanjeev Khanna. “Space-Efficient Online Computation of Quantile Summaries”. In: SIGMOD ’01. Santa Barbara, California, USA: Association for Computing Machinery, 2001, pp. 58–66. ISBN: 1581133324. DOI: `10.1145/375663.375670`. URL: `https://doi.org/10.1145/375663.375670`.

[46] Roger Zhe Li, Julián Urbano, and Alan Hanjalic. “Leave No User Behind: Towards Improving the Utility of Recommender Systems for Non-mainstream Users”. In: (Feb. 2021). DOI: `10.1145/3437963.344176910.1145/3437963.344176910.1145/3437963.3441769`. URL: `http://arxiv.org/abs/2102.01744%20http://dx.doi.org/10.1145/3437963.3441769%2010.1145/3437963.3441769%2010.1145/3437963.3441769`.

[47] Yong Zheng, Mayur Agnani, and Mili Singh. “Identification of grey sheep users by histogram intersection in recommender systems”. In: *International Conference on Advanced Data Mining and Applications*. Springer. 2017, pp. 148–161.

[48] Xiaoyuan Su and Taghi M Khoshgoftaar. “A survey of collaborative filtering techniques”. In: *Advances in artificial intelligence* 2009 (2009).

[49] Gábor Takács, István Pilászy, Bottyán Németh, and Domonkos Tikk. “Investigation of various matrix factorization methods for large recommender systems”. In: *2008 IEEE International Conference on Data Mining Workshops*. IEEE. 2008, pp. 553–562.

[50] Yunhong Zhou, Dennis Wilkinson, Robert Schreiber, and Rong Pan. “Large-scale parallel collaborative filtering for the netflix prize”. In: *International conference on algorithmic applications in management*. Springer. 2008, pp. 337–348.

[51] *Alternating Least Squares (ALS) matrix factorization*. URL: `https://spark.apache.org/docs/2.4.7/api/scala/index.html#org.apache.spark.ml.recommendation.ALSs`.

[52] Yifan Hu, Yehuda Koren, and Chris Volinsky. “Collaborative filtering for implicit feedback datasets”. In: *2008 Eighth IEEE International Conference on Data Mining*. Ieee. 2008, pp. 263–272.

[53] Pearl Pu, Li Chen, and Rong Hu. “Evaluating recommender systems from the user’s perspective: Survey of the state of the art”. In: *User Modeling and User-Adapted Interaction* 22 (4-5 Oct. 2012), pp. 317–355. ISSN: 09241868. DOI: `10.1007/s11257-011-9115-7`.

[54] Mouzhi Ge, Carla Delgado-Battenfeld, and Dietmar Jannach. *Beyond accuracy: evaluating recommender systems by coverage and serendipity*. 2010, pp. 257–260.

[55] Tao Zhoua, Zoltán Kuscsik, Jian Guo Liu, Matúš Medo, Joseph Rushton Wakeling, and Yi Cheng Zhang. “Solving the apparent diversity-accuracy dilemma of recommender systems”. In: *Proceedings of the National Academy of Sciences of the United States of America* 107 (10 Mar. 2010), pp. 4511–4515. ISSN: 00278424. DOI: `10.1073/pnas.1000488107`.

[56] Saúl Vargas, Sandoval Supervisor, and Pablo Castells Azpilicueta. *Novelty and Diversity Enhancement and Evaluation in Recommender Systems*. Aggregate DIversity<br/><br/>. 2012.

[57] Siu Kwan Lam, Antoine Pitrou, and Stanley Seibert. “Numba: A LLVM-Based Python JIT Compiler”. In: *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*. LLVM ’15. Austin, Texas: Association for Computing Machinery, 2015. ISBN: 9781450340052. DOI: 10.1145/2833157.2833162. URL: https://doi.org/10.1145/2833157.2833162.

[58] Eckart Zitzler. *Evolutionary algorithms for multiobjective optimization: Methods and applications*. Vol. 63. Citeseer, 1999.

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

[60] Xin-She Yang. “Chapter 2 - Analysis of Algorithms”. In: *Nature-Inspired Optimization Algorithms*. Ed. by Xin-She Yang. Oxford: Elsevier, 2014, pp. 23–44. ISBN: 978-0-12-416743-8. DOI: https://doi.org/10.1016/B978-0-12-416743-8.00002-6. URL: https://www.sciencedirect.com/science/article/pii/B9780124167438000026.

[61] Kalyanmoy Deb, Ram Bhushan Agrawal, et al. “Simulated binary crossover for continuous search space”. In: *Complex systems* 9.2 (1995), pp. 115–148.

[62] Kalyanmoy Deb and Himanshu Jain. “An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, Part I: Solving problems with box constraints”. In: *IEEE Transactions on Evolutionary Computation* 18 (4 2014). NSGA-III, pp. 577–601. ISSN: 1089778X. DOI: 10.1109/TEVC.2013.2281535.

[63] Eckart ; Zitzler, Marco ; Laumanns, Lothar Thiele, Eckart Zitzler, and Marco Laumanns. “SPEA2: Improving the strength pareto evolutionary algorithm”. In: (2001). DOI: 10.3929/ethz-a-004284029. URL: https://doi.org/10.3929/ethz-a-004284029.

[64] Kalyanmoy Deb and Samir Agrawal. “A niched-penalty approach for constraint handling in genetic algorithms”. In: *Artificial Neural Nets and Genetic Algorithms*. Springer. 1999, pp. 235–243.

[65] Kalyanmoy Deb and Debayan Deb. “Analysing mutation schemes for real-parameter genetic algorithms”. In: *International Journal of Artificial Intelligence and Soft Computing* 4.1 (2014), pp. 1–28.

[66] Antonio Benitez-Hidalgo, Antonio J Nebro, Jose Garcia-Nieto, Izaskun Oregi, and Javier Del Ser. “jMetalPy: A Python framework for multi-objective optimization with metaheuristics”. In: *Swarm and Evolutionary Computation* 51 (2019), p. 100598.

[67] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. “A fast and elitist multiobjective genetic algorithm: NSGA-II”. In: *IEEE transactions on evolutionary computation* 6.2 (2002), pp. 182–197.

[68] Sushant S. Garud, Iftekhar A. Karimi, and Markus Kraft. “Smart Adaptive Sampling for Surrogate Modelling”. In: *26th European Symposium on Computer Aided Process Engineering*. Ed. by Zdravko Kravanja and Miloš Bogataj. Vol. 38. Computer Aided Chemical Engineering. Elsevier, 2016, pp. 631–636. DOI: https://doi.org/10.1016/B978-0-444-63428-3.50110-7. URL: https://www.sciencedirect.com/science/article/pii/B9780444634283501107.

[69] Vira Chankong and Yacov Y Haimes. *Multiobjective decision making: theory and methodology*. Courier Dover Publications, 2008.

[70] Saku Kukkonen and Jouni Lampinen. *GDE3: The third Evolution Step of Generalized Differential Evolution*. GDE3.

[71] KV Price, RM Storn, and JA Lampinen. “Differential Evolution-A Practical Approach to Global Optimization,□ Springer-Verlag”. In: *Berlin* (2005).

[72] Rainer Storn and Kenneth Price. “Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces”. In: *Journal of global optimization* 11.4 (1997), pp. 341–359.

[73] Eckart Zitzler and Lothar Thiele. “Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach”. In: *IEEE transactions on Evolutionary Computation* 3.4 (1999), pp. 257–271.

[74] Bernard W Silverman. “Density Estimation for Statistics and Data Analysis”. In: (1986).

[75] Lyndon While. “A New Analysis of the LebMeasure Algorithm for Calculating Hypervolume”. In: *Evolutionary Multi-Criterion Optimization*. Ed. by Carlos A. Coello Coello, Arturo Hernández Aguirre, and Eckart Zitzler. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 326–340. ISBN: 978-3-540-31880-4.

[76] Carlos M Fonseca, Luís Paquete, and Manuel López-Ibánez. “An improved dimension-sweep algorithm for the hypervolume indicator”. In: *2006 IEEE international conference on evolutionary computation*. IEEE. 2006, pp. 1157–1163.

[77] Oliver Schuetze, Xavier Equivel, Adriana Lara, and Carlos A Coello Coello. “Some comments on GD and IGD and relations to the Hausdorff distance”. In: *Proceedings of the 12th annual conference companion on Genetic and evolutionary computation*. 2010, pp. 1971–1974.

[78] David Allen Van Veldhuizen. *Multiobjective evolutionary algorithms: classifications, analyses, and new innovations*. Tech. rep. 1999.

[79] Carlos A Coello Coello and Nareli Cruz Cortés. “Solving multiobjective optimization problems using an artificial immune system”. In: *Genetic programming and evolvable machines* 6.2 (2005), pp. 163–190.

[80] Eckart Zitzler, Lothar Thiele, Marco Laumanns, Carlos M Fonseca, and Viviane Grunert Da Fonseca. “Performance assessment of multiobjective optimizers: An analysis and review”. In: *IEEE Transactions on evolutionary computation* 7.2 (2003), pp. 117–132.

[81] Sergio Garcia and Cong T Trinh. “Comparison of multi-objective evolutionary algorithms to solve the modular cell design problem for novel biocatalysis”. In: *Processes* 7.6 (2019), p. 361. DOI: 10.3390/pr7060361.

[82] Raj Jain. *The art of computer systems performance analysis: techniques for experimental design, measurement, simulation, and modeling*. john wiley & sons, 1990.

[83] Gavin C Cawley and Nicola L C Talbot. *On Over-fitting in Model Selection and Subsequent Selection Bias in Performance Evaluation*. 2010, pp. 2079–2107.

[84] Indraneel Das and John E Dennis. “Normal-boundary intersection: A new method for generating the Pareto surface in nonlinear multicriteria optimization problems”. In: *SIAM journal on optimization* 8.3 (1998), pp. 631–657.

[85] Henry B Mann and Donald R Whitney. “On a test of whether one of two random variables is stochastically larger than the other”. In: *The annals of mathematical statistics* (1947), pp. 50–60.

[86] Fiona McElduff, Mario Cortina-Borja, Shun-Kai Chan, and Angie Wade. “When t-tests or Wilcoxon-Mann-Whitney tests won’t do”. In: *Advances in physiology education* 34.3 (2010), pp. 128–133.

[87] Carlo E Bonferroni. “Il calcolo delle assicurazioni su gruppi di teste”. In: *Studi in onore del professore salvatore ortu carboni* (1935), pp. 13–60.

[88] Carlos García-Martínez, Pablo D Gutiérrez, Daniel Molina, Manuel Lozano, and Francisco Herrera. “Since CEC 2005 competition on real-parameter optimisation: a decade of research, progress and comparative analysis’s weakness”. In: *Soft Computing* 21.19 (2017), pp. 5573–5583.

[89] Harold D Delaney and András Vargha. “Comparing several robust tests of stochastic equality with ordinally scaled variables and small to moderate sized samples.” In: *Psychological Methods* 7.4 (2002), p. 485.

[90] Alastair Porter, Dmitry Bogdanov, Robert Kaye, Roman Tsukanov, and Xavier Serra. “Acousticbrainz: a community platform for gathering music information obtained from audio”. In: *Müller M, Wiering F, editors. ISMIR 2015. 16th International Society for Music Information Retrieval Conference; 2015 Oct 26-30; Málaga, Spain. Canada: ISMIR; 2015.* International Society for Music Information Retrieval (ISMIR). 2015.

[91] Dmitry Bogdanov, Nicolas Wack, Emilia Gómez Gutiérrez, Sankalp Gulati, Herrera Boyer, Oscar Mayor, Gerard Roma Trepat, Justin Salamon, José Ricardo Zapata González, Xavier Serra, et al. “Essentia: An audio analysis library for music information retrieval”. In: *Britto A, Gouyon F, Dixon S, editors. 14th Conference of the International Society for Music Information Retrieval (ISMIR); 2013 Nov 4-8; Curitiba, Brazil.[place unknown]: ISMIR; 2013. p. 493-8.* International Society for Music Information Retrieval (ISMIR). 2013.

[92] Cynthia CS Liem and Chris Mostert. *Can't Trust the Feeling? How Open Data Reveals Unexpected Behavior of High-Level Music Descriptors*. 2020.

[93] C Mostert. *More than a feeling? Reliability and robustness of high-level music classifiers*. URL: http://repository.tudelft.nl/..

[94] Lena Oden. "Lessons learned from comparing C-CUDA and Python-Numba for GPU-Computing". In: *2020 28th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*. 2020, pp. 216–223. DOI: 10.1109/PDP50117.2020.00041.

[95] Mohammad Rastegari, Shobeir Fakhraei, Jonghyun Choi, David Jacobs, and Larry S. Davis. *Comparing apples to apples in the evaluation of binary coding methods*. 2014. arXiv: 1405.1005 [cs.CV].

[96] Agner Fog. *Instruction tables: Lists of instruction latencies, throughputs and micro-operation breakdowns for Intel, AMD and VIA CPUs*. Mar. 2021.

[97] Henry S Warren. *Hacker's delight*. Pearson Education, 2013.

[98] Jeffrey L Elman. "Learning and development in neural networks: The importance of starting small". In: *Cognition* 48.1 (1993), pp. 71–99.