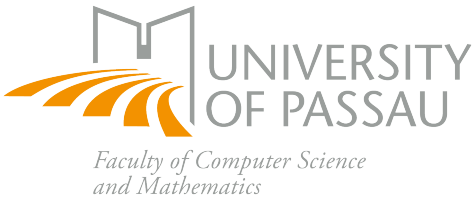

University of Passau

Faculty of Computer Science and Mathematics

Bachelor's Thesis

Evolutionary Computation and Swarm Intelligence in the Field of Music Recommender Systems

Author:

Dominik Dassow

Matriculation Number: 87158

Date: September 30, 2021

Advisor:

Prof. Dr. Gordon Fraser
Department of Software Engineering II
University of Passau

Dassow, Dominik:
Evolutionary Computation and Swarm Intelligence in the Field of Music Recommender Systems
Bachelor's Thesis, University of Passau, 2021.

Abstract

Thanks to the rise of music streaming services, such as Spotify, Apple Music, or Pandora, anybody can instantly access almost all the music in the world. This has led to increased demand in using recommendations to help users find relevant and interesting music tracks and playlists. Research on recommender systems (RSs) has already gained interest throughout the last decades and resulted in some valuable techniques for general recommendations. However, the field music recommender systems (MRSs) in particular is still only sparsely researched. One emerging approach is incorporating evolutionary computation (EC) and swarm intelligence (SI) techniques in the recommendation process. Notwithstanding, using these biology-inspired approaches explicitly in MRSs lacks research as well.

This thesis aims at partially filling this gap in research by examining some EC and SI techniques in the field MRSs. For that, I lay out some key learnings from existing research on EC and SI in general RSs, analyse possible application areas in MRSs, and compare different algorithms for the MRS task of music playlist continuation (MPC). This task has gained much attention during the *RecSys Challenge 2018* and was identified as one of the grand challenges in MRSs. The primary focus of this thesis lies in designing, implementing, and conducting a study that compares EC and SI algorithms in the context of the MPC task. The study results show an overall inferior solution quality of the examined algorithms compared to machine learning approaches. Nevertheless, the EC algorithm "SMS-EMOA" yields the best results among the other algorithms in the study.

The study results suggest that using EC and SI algorithms for calculating and optimising solutions in the MPC task is not very suitable. However, using EC and SI in other (music) recommendation application areas highlighted in this thesis still seems promising for future work.

Contents

1. Introduction

In the era of internet-based commerce and entertainment, one of the biggest challenges for users is handling the possible information overload [1]. Every online shop, every movie and music streaming service, every tourism platform, etc., expands its collection of items every day. For example, the Amazon.com marketplace consists of over 45 million different products[1], the Spotify music catalogue has over 70 million tracks[2], and the Apple iTunes Store offers more than 65,000 films[3]. And in the jungle of all these possibilities, millions of users - all of them with their own preferences - try to find the most interesting items.

As a result of joint efforts by the industry and the research, new tools have emerged that intend to help users find relevant items among large collections: recommender systems (RSs). At the heart of RSs is the idea of using data about users and data about items in order to match them. In 2006, Clive Humby coined the phrase "data is the new oil" [2], and especially in the context of RSs, he could not be more correct. However, suitable methods need to be determined and elaborated to use the data and provide users with valuable recommendations. Since the first mentioning of RSs in the mid-1990s [3], the industry has adopted many concepts and methods developed in research. The currently most sophisticated and used RSs base on collaborative filtering (CF) and content-based filtering (CBF).

Moreover, metaheuristics like evolutionary computation (EC) and swarm intelligence (SI) have gained interest in research within the last decades. EC and SI are traditionally considered part of computational intelligence [4] and cover a variety of biology-inspired optimisation techniques. Among the main methods in EC are evolutionary algorithms (EA, Bäck [5]) and genetic algorithms (GA, Holland [6]), which build upon Darwin's *"Survival of the Fittest"* theory and make use of selection, reproduction, and fitness functions. The field of SI orients towards the behaviour of animals and their communication techniques. This large set of analogy-based approaches, e.g., ant colony optimisation (ACO, Colorni et al. [7]), particle swarm

[1] https://www.amazon.com
[2] https://newsroom.spotify.com/company-info/
[3] https://itunes.apple.com/us/genre/movies/id33

optimisation (PSO, Kennedy and Eberhart [8]), or artificial bee colony (ABC, Pham et al. [9]), are used to solve different optimisation problems. Even though SI might have been seen as part of the EC paradigm in the past [4], SI has become a standalone field in research nowadays.

The research on EC and SI combined with RSs has mainly focused on general systems and less on specific application areas. Even though the key components are quite similar across all types of RSs, there is no "one-fits-all" system. Every application area has its differences and unique requirements. For example, a full-length movie might be watched one night but will probably not be rewatched for a long time, if at all. On the contrary, a music track is more of a short-lived item that could be streamed multiple times a day and added to different playlists. With this in mind, the current state of research on RSs based on EC and SI still seems too broad and lacks sector specialisation.

1.1 Motivation

On a personal level, music has always been enormously interesting, and seeing the music market growing every day is quite fascinating. Significantly since the rise of streaming services like Spotify, Apple Music, Pandora, etc., the way people consume music has drastically changed. This has led to increasing demand by the industry for music recommendation systems (MRSs) and a raised interest in research. Identified as one of the grand challenges in MRSs (Schedl et al. [10]), the task of music playlist continuation (MPC) got much attention in the context of the *RecSys Challenge 2018*[4]. Additionally, the approach evolutionary computation and swarm intelligence algorithms follow, i.e., using techniques that have emerged over millions of years in nature for modern computer science problems, is heavily intriguing. Hence, combining these two topics seems to be perfect for exploration in this thesis.

1.2 Research Questions

For quite some time, recommender systems managed to be valuable without using evolutionary computation and swarm intelligence, even though these techniques have experienced a massive gain in importance in computational intelligence. Nonetheless, researchers' interest in combining these two areas grows year by year, and the results show great potential. Even though notable progress has been made in the field of EC/SI and general RSs, only basic research in the area of music recommender systems can be found. Especially research concerning different EC/SI techniques in the context of the music playlist continuation task is missing.

In order to fill the said gap in research - at least partially - this thesis' contribution can be narrowed down to the following research questions:

[4] https://www.recsyschallenge.com/2018/

RQ1 What can be learnt from the advanced research on recommender systems based on evolutionary computation and swarm intelligence, and what can be adopted in music recommender systems?

RQ2 How can evolutionary computation and swarm intelligence methods be applied in music recommender systems, and how can they improve results?

RQ3 How does a genetic algorithm, an evolutionary algorithm, and an ant colony optimisation algorithm differ in suitability for the music playlist continuation task, and which one performs better?

1.3 Thesis Structure

I start by giving an introduction to the theoretical concepts of evolutionary computation and swarm intelligence as well as recommender systems in section 2. Together with related work in the field of combing these research areas, I can then answer the first two research questions. In section 3, I lay out the methods used in the study that compares the different EC and SI algorithms for the task of music playlist continuation, before, in section 4, detailing how the study is implemented. In section 5, I explain the general study conduct, evaluate the results, and answer the third research question. Finally, in section 6, I wrap up this thesis and discuss the most significant learnings.

2. Background

Before diving deep into the application of evolutionary computation and swarm intelligence algorithms to the task of music playlist continuation, the theoretical basics and underlying concepts are explained in this section. I start by giving an overview of the ideas behind EC and SI algorithms and how they generally work. Following that, the approach of multi-objective optimisation, which is essential in many real-world problems, including recommender systems, is detailed. Then, I introduce the fundamentals of RSs, and MRSs in specific, by laying out some common techniques, key challenges, and particular application areas. Finally, I review related work already done in the examined research fields before summarising the most important aspects for the subsequent study.

2.1 Evolutionary Computation

The field of evolutionary computation [11] belongs to the topic of computational intelligence [12], an area in computer science that deals with biology-inspired methods to solve real-world problems. Many of these problems are often so complex that it is not feasible to aim at finding *the best* solution. A common technique of approaching these problems anyway is by approximating a set of *possibly best* solutions with the help of metaheuristics [13]. Solving computational problems this way makes them so-called optimisation problems since the goal is not to find the provable, absolute best solution but to optimise the solutions as much as possible with the available resources. Metaheuristics, therefore, focus on intelligently sampling a subset of all possible solutions, which are otherwise infeasible to compute, thus resulting in solutions that cannot be guaranteed to be the most optimal.

As the name suggests, EC borrows its central idea from biological evolution and also strives for improvement over time. At its core lies the trial-and-error strategy that describes a process of repeatedly generating and testing solutions. In EC, the starting point is an initial population of candidate solutions that is iteratively updated and improved. Each new generation is created on the basis of the previous one, however, without the least desired candidates and with slight random changes. Biologically

speaking, this behaviour resembles the process of natural selection and mutation. The desirability of a candidate is expressed by its fitness, i.e., how valuable it is w.r.t. an objective. All in all, this process has the goal of steadily raising the overall fitness of the population and eventually finding the (possibly) most optimal solutions.

Over many years of research, different variants of this concept have been developed and proven to be valuable metaheuristics to solve optimisation problems. This thesis takes a deeper look at the family of evolutionary algorithms (EA) and the refined class of genetic algorithms (GA) in specific, which build upon the concepts of EAs.

2.1.1 Evolutionary Algorithms

The family of evolutionary algorithms all base on the same underlying idea [11]: Given a population of individuals and a resource-constrained environment, competition for those resources leads to natural selection (survival of the fittest). Over time, i.e., over many generations, this causes a rise in the population's fitness w.r.t. an objective. The fitness is expressed by a heuristic quality function that can be applied to a set of candidate solutions, which themselves are created from the individuals in the population. The fitness values then allow determining which candidates are chosen for the next generation, i.e., which should be rewarded or which should be neglected. Rewarding better candidates or neglecting worse ones depends on the EA variant; however, all variants share the idea that higher fitness values are better. New generations are created by applying recombination and/or mutation operations to a population. Recombination selects two or more candidates (the parents) and produces one or more new candidates (the children). Mutation applies to one candidate and generates one new candidate. Executing both operations leads to a set of new candidates (the offspring), which then get evaluated w.r.t. their fitness to compete with the parents. This process is repeated until a predefined termination condition is met, e.g., a certain number of generations is reached, or a candidate solution with a certain quality is found.

Algorithm 1 Evolutionary Algorithm [11]

```
Initialise a population with random candidate solutions
Evaluate each candidate
while not termination condition is reached do
    Select parents
    Recombine pairs of parents
    Mutate the resulting children candidates
    Evaluate the new offspring candidates
    Select individuals for next generation
```

Independent of the particular EA variant, three components have to be designed for any EA implementation: representation, recombination, and mutation. Whereas the representation of a candidate solution depends mainly on the problem to solve, the recombination and mutation design depends on the representation. All necessary details will be explained in the course of this thesis; a more thorough explanation can be found in [11].

2.1.2 Genetic Algorithms

The genetic algorithm [11; 14] is the most popular variant of evolutionary algorithms. It was first conceived by Holland [6] in 1975 and quickly gained much attention for solving optimisation problems efficiently while using a binary representation. During the following decades, this initial form (the 'simple' GA) evolved more versatile forms using additional features and incorporating other representations as well. However, all GAs follow the same workflow: given a population with N individuals, each generation first creates an intermediary mating population with the same size by selecting a number of parents based on their fitness. Then, a crossover operation is applied to all randomly chosen pairs of this mating population, each with a probability of p_c, resulting in a set of children. Applying the mutation operation to each child, every time with a probability of p_m, creates the offspring population, which then fully replaces the parents. Therefore, each new generation primarily contains new individuals and only a few, maybe none, survivors of the previous one, depending on the probabilities p_m and p_c.

Algorithm 2 Genetic Algorithm [15]

Initialise a population with random candidate solutions
Evaluate each candidate
while not termination condition is reached **do**
    Select mating population from parents (according to their fitness)
    Crossover random pairs in the mating population (with probability p_c)
    Mutate the resulting children candidates (with probability p_m)
    Evaluate the resulting offspring candidates
    Replace candidates with the offspring population

2.2 Swarm Intelligence

Swarm intelligence (SI) [4; 16] is another computational intelligence [12] technique and is oriented towards the collective behaviour of animals. With decentralised control and self-organisation at its core, SI techniques aim at using communication methods and interactions found in natural systems to solve real-world problems. These population-based systems, i.e., swarms, are made up of a number of simple agents that interact with each other and their environment. Examples from nature include colonies of ants, schools of fish, and flocks of birds. Like evolutionary computation, SI can be utilised as a metaheuristic to approximate a set of *possibly best* solutions for, i.a., optimisation problems.

SI systems are typically composed of many homogeneous individuals, i.e., agents, that interact based on simple behavioural rules and exchange information directly or via their environment. Another characteristic feature is the absence of a central coordinator or controller, hence making the agents in a swarm eminently self-organised. This level of independence allows SI systems to be highly scalable, parallel, and very much fault-tolerant.

Over the last decades, research in the field of SI has evolved numerous algorithms based on this concept. Among the most popular are ant colony optimisation (ACO,

Colorni et al. [7]) and particle swarm optimisation (PSO, Kennedy and Eberhart [8]). In the context of this thesis, the focus lies in ACO algorithms.

2.2.1 Ant Colony Optimisation Algorithms

Ant colony optimisation [7; 16] is oriented towards the collective behaviour of ants in their search for food. Ants initially explore the area surrounding their nest randomly and leave a chemical pheromone trail on the ground. This pheromone trail can be smelled by other ants, which then again influences the choice of path each ant explores. As soon as an ant finds a food source, it carries some of it back to the nest while laying new pheromones at the same time. The amount of pheromones usually depend on the quality and quantity of the located food. With the help of these pheromone trails, other ants in the colony are then able to find the shortest path between their nest and the food source. Due to natural evaporation of these trails, the influence of particular trails on the ants' decision-making decreases over time.

Based on this indirect communication between the ants via pheromone trails, Dorigo et al. have developed the first ACO algorithm in 1996: Ant System [17]. In the years after that, more different variants based on that concept have evolved, mostly aiming at providing a suitable metaheuristic for solving optimisation problems. Generally speaking, the ants in a colony build solutions by traversing a fully connected construction graph, which contains a set of solution components (either the vertices or the edges). By moving from vertex to vertex, an ant constructs a path of components that resembles a (partial) solution. An ant's decision of which component it chooses next on its path depends, i.a., on the pheromone values associated with each component. Additionally, the amount of pheromone an ant deposits on a component depends on the quality of the solution the ant constructs. All in all, the underlying idea of all ACO algorithms can be summoned as follows:

Algorithm 3 Ant Colony Optimisation [18; 19]

```
Set parameters, initialise pheromone trails
while not termination condition is reached do
    ConstructAntSolutions
    DaemonActions
    UpdatePheromones
```

After initialisation, the algorithm repeats three phases: solution creation, daemon action application, and pheromone update. In the first step, every ant in a colony creates a number of solutions independently. An initially empty solution is filled by adding one component after another, as long as the partial solution does not violate any constraints defined for the problem. One common constraint, e.g., is that solutions must not contain duplicate components. The order in which the components are selected depends on *(i)* the pheromone value of each component, $\tau(c)$, and *(ii)* the heuristic information associated with each component, $\eta(c)$. The heuristic information is highly problem-specific and generally indicates the quality of a component. So-called daemon actions denote a set of problem- and variant-specific

actions that are executed after all solutions in a cycle have been created but before the pheromone trails are updated. A typical and widely used daemon action, e.g., is determining the best solutions found in the previous step. Finally, the pheromone values are updated, typically by increasing values associated with good solutions and decreasing these associated with bad ones. This behaviour is commonly archived by, first, applying an evaporation factor to all values and then raising the values of components contained in the good solutions. This cycle is repeated until a predefined termination condition is met, e.g., a certain number of cycles is reached, or a solution with a certain quality is found.

2.3 Multi-Objective Optimisation

In the last sections, I have detailed two computational intelligence techniques widely used for optimisation problems. I have briefly discussed how metaheuristics can be utilised to approximate a set of possibly best solutions and that they require a method of evaluating candidate solutions w.r.t. an objective. However, real-world problems are usually not limited to a single objective, but multiple, and they (or some of them) are typically contradicting [11]. This class of problems is generally referred to as *multi-objective problems* (MOPs), and solving them is known as *multi-objective optimisation* (MOO).

2.3.1 Non-Dominated Solutions

The current state-of-the-art approaches to solve multi-objective problems incorporate the concepts of dominance and Pareto optimality [11]. Dominance is defined as follows: given two solutions and a set of objective functions (which, without loss of generality, are assumed to be maximised), one solution dominates the other if all its objective values are at least as high as the other's, and at least one objective value is strictly higher than the other's. Formally expressed, solution A dominates B ($A \succeq B$), considering m objective functions, when:

$$A \succeq B \iff \forall\, i \in \{1, \ldots, m\} : A_i \geq B_i \quad \text{and} \quad \exists\, i \in \{1, \ldots, m\} : A_i > B_i \tag{2.1}$$

with A_i and B_i denoting the objective values, respectively, for the i^{th} objective function. For multi-objective optimisation problems with two or more conflicting objectives, no single solution dominates all others but a (possibly infinite) set of non-dominated solutions exists. This set is characterised by not containing any solutions that could be improved w.r.t. any objective without negatively affecting any other objective's value. The set of non-dominated solutions is also commonly referred to as *Pareto set*, *Pareto front*, or *Pareto optimal solutions*.

2.3.2 Hypervolume

The hypervolume indicator, originally proposed by Zitzler and Thiele [20], measures "the size of dominated space" [21]. In other words, it allows comparing different sets of non-dominated solutions by measuring the volume of these solutions in an m-dimensional space (where m denotes the number of objectives) w.r.t. a certain reference point.

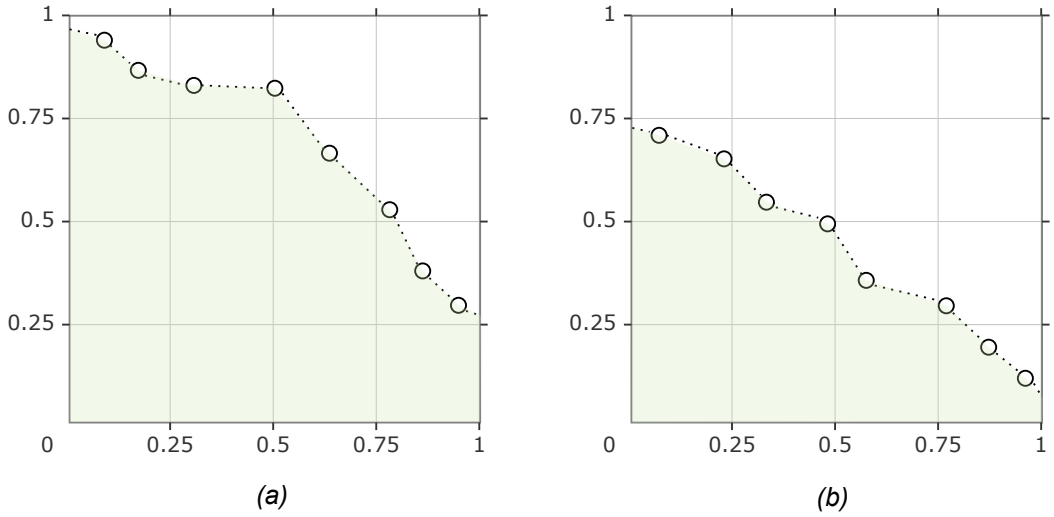

Figure 2.1: Examples of two solutions in two-dimensional objective planes. The circles denote single non-dominated solutions; the green area denotes the dominated space.

Figure 2.1 demonstrates how the hypervolume indicator works. Both plots show sets of non-dominated solutions for a two-objective problem (the x and y axes denote the fitness values for each objective). Let $(0, 0)$ be the reference point; then, it is evident that the left solutions' dominated space (area) is higher than the right solutions'. It can be derived that the solutions in *(a)* are better than those in *(b)*. The same technique can also be utilised to compare more than two objectives.

2.4 Recommender Systems

A recommender system [3; 22] is, generally speaking, an algorithm that aims at suggesting interesting items to users. The items to recommend strongly depend on the industry and could be, e.g., movies to watch, songs to listen to, or products to buy. The research field of RSs originated in the 1990s to filter relevant online news articles based on personal preferences [23; 24]. Nowadays, RSs are oftentimes indispensable for many companies, especially internet-based ones. The research and the industry sector almost form a symbiotic relationship: on the one hand, the industry thankfully accepts new developments in RSs to build even better (software) products. On the other hand, companies occasionally share parts of the ever-growing amount of data they (or rather their users) generate, which encourages even more research in this field. Examples of companies sharing data are the *Last.fm Million Song Dataset*[5], the *Netflix Prize*[6], and the *Spotify Million Playlist Dataset*[7]. This section takes a deeper look at how RSs generally work and which techniques are used before I will examine the field of music recommendation systems in the next section.

The underlying idea of RSs is to predict the 'rating' users would give to items. These predictions are typically based on three types of data:

[5] http://millionsongdataset.com/lastfm/
[6] https://www.netflixprize.com/
[7] https://recsys-challenge.spotify.com

- **User data**, e.g., a user's age or the country he/she lives in
- **Item data**, e.g., a movie's genre or its actors
- **User-Item data**, e.g., which movies a user likes or has already watched

The last type is commonly referred to as a user-item rating/interaction/preference matrix and is the most important component of RSs. Values in this matrix can be categorised into two types: *(1) explicit* [3] data, e.g., when a user rates a movie with 1 to 5 stars, and *(2) implicit* [25] data, e.g., when a user stops watching a movie after 10 minutes. The first two data types are mainly used to enhance predictions, especially in more modern RS approaches. With this data and the resulting predictions, an RS then can recommend other, ideally relevant and interesting items to users. In the following, I will explain some of the most popular techniques for making these predictions, namely *collaborative filtering* (CF), *content-based filtering* (CBF), and *hybrid filtering*.

2.4.1 Collaborative Filtering

A common approach when designing recommender systems is using collaborative filtering [3; 22; 26]. It is based on the idea that users, who have liked similar items in the past, will like similar items in the future. For that, the RS first determines a set of users who have the same or similar preferences as a certain user and then recommends items these other users prefer. For example, let Alice and Bob be two users who both have watched *Inception* and rated it with 5 stars. Moreover, the system knows that Bob has watched *The Matrix* and *The Wolf of Wall Street*, and also gave them a high rating. A CF algorithm that aims at recommending movies to Alice, would then find the similar user Bob (based on the shared high rating for *Inception*) and would recommend these two items to Alice.

To find similar users, CF-based algorithms typically use the concept of neighbourhoods, which contain users with similar preferences, so-called 'peers'. Depending on the algorithm, the users' profiles (age, location, etc.) might also influence the neighbourhoods. Similar to these user-based neighbourhoods, item-based neighbourhoods can be utilised to find similar items, i.e., items that share related features or are rated similarly by similar users. Both techniques are usually based on similarity measures like the *Pearson correlation coefficient* or the *Cosine* function. They use vector representations of the user-item rating matrix to calculate 'distances' between the users and between the items. After creating these neighbourhoods a priori, the RS is then able to *(1)* find the set of users similar to a particular user and *(2)* find items similar to the ones preferred by the similar users.

One of the main issues with CF-based RSs can be found in the so-called *new user problem* and the *new item problem*, which are part of the *cold-start problem*. When there is too little data about a (new) user's preferences, the system cannot reliably recommend relevant items. On the other hand, when new items are added to the catalogue, none of the users has rated that item; thus, it gets not (or at least less) recommended. Together with the *data sparsity problem*, which describes the disproportion between the vast numbers of users and items many RSs deal with and the relatively few ratings that are given, the *long-tail problem* is another common

issue in CF-based RSs. This 'long-tail' describes the steep logarithmically decreasing frequency of item ratings, resulting in a few items with a lot of ratings, any many items with only a few.

2.4.2 Content-based Filtering

The other widely used recommender systems approach is content-based filtering [22; 27]. The idea is that users are recommended items similar to the ones they have preferred in the past. The similarity of items is based on a 'content description' available for each item, which is often referred to as a set of keywords describing an item. For example, let Alice be a user who has watched *Inception* and rated it with 5 stars. A CBF algorithm would then look for similar movies and find, e.g., *The Matrix* based on the genre and *The Wolf of Wall Street* based on the same leading actor, Leonardo DiCaprio. These two items would then be recommended to Alice.

To find similar items by means of content keywords, the entire catalogue of items must have been scanned and indexed a priori. After that, techniques known from classical information retrieval, like the *term frequency/inverse document frequency* (TF-IDF) measure [28], allow weighting the items' keywords for relative importance. Finally, a user's preferences are compared with the weighted items in the catalogue in order to recommend those with high overlap.

Similar to CF-based algorithms, CBF-based algorithms have to deal with the *cold-start problem*, the *data sparsity problem*, and the *long-tail problem*, too. However, especially the *new item problem* can be elevated to a great extent since an item's content keywords are mostly static and initially available. Nevertheless, a key issue with CBF is the so-called overspecialisation [3], i.e., the lack of diversity. When recommendations are solely based on a user's past preferences, the recommended items are likely limited to items similar to those already seen or rated. Additionally, when recommended items are *too* similar to the already known, they might sometimes be less relevant (clearly depending on the industry).

2.4.3 Hybrid Filtering

Several studies, such as [29; 30; 31], have shown that combining collaborative and content-based filtering outperforms the pure approaches. This combination, i.e., hybridisation, can be achieved by different strategies [3], including:

- using CF and CBF separately and then combining the results
- adding CF methods and to CBF (or vice versa)
- constructing a unified model with both CF and CBF

A good example of hybrid filtering can be found in the recommender system of Netflix [32]. Their recommendations are based on comparing users' watch histories and their likes (i.e., CF) as well as finding movies that share features and keywords with these a user has already rated highly before (i.e., CBF).

2.5 Music Recommender Systems

The field of music recommendation, commonly referred to as *music recommender systems* [10], is a class of recommender systems focused on music items, i.e., music tracks. MRSs have experienced a massive gain in interest during the last years, both in research and in industry. Thanks to the rise of music streaming services, such as Spotify, Apple Music, or Pandora, the music industry evolved from an ownership-based to an access-based landscape [33]. Prior to these services, the main goal of MRSs was to sell single tracks and albums; however, nowadays, almost all music in the world is instantly accessible for users. The industry has therefore transitioned to recommend 'experiences' instead of just items to help users find relevant tracks and playlists in the overwhelming space of sometimes over 70 million available tracks.

In comparison to general RSs and even other RS application areas like movie recommendation, music recommendation poses a number of particular challenges unique to this field. Among them are the short duration of tracks (in contrast to movies), the high emotional and context factors music entails, and the sequential consumption of (possibly duplicate) tracks. In this section, I start by detailing all the major particularities MRSs have to deal with before presenting one of the grand challenges in modern MRSs and this thesis' main focus: music playlist continuation.

2.5.1 Particularities of Music Recommendation

Music recommendation yields a number of particular challenges other areas do not have to face, distinguishing MRSs from fields like movie, book, or product recommendation. A comprehensive list and analysis can be found in the paper Schedl et al. [10] published in 2018; thus, only the most influential particularities are revised here.

Duration of tracks. Unlike movies, which have a typical duration of 90 minutes, the duration of most music tracks ranges between 3 and 5 minutes. This kind of volatility makes a track more of a short-lived item and requires special consideration for recommending them.

Magnitude of tracks. Most music streaming services give users access to tens of millions of tracks in their catalogue. In comparison, most movie and series streaming platforms 'only' span from thousands to tens of thousands of items. Hence, the issue of scalability is a lot more prominent in MRSs than it is in other domains.

Sequential consumption. Music tracks are typically consumed in sessions, i.e., users listen to multiple tracks sequentially, usually in one or more playlists. One of the main considerations in an MRS, therefore, is the arrangement of these playlists.

Recommendation of previously recommended tracks. In contrast to, e.g., movie recommendations, users of MRSs usually like track recommendations they have gotten before. Thus, one key challenge in music recommendation is determining when and how often tracks get recommended again.

Listening intent and context. Music consumption behaviour is highly dependent on a user's mood, activity, location, time, or other factors. The same user might prefer very different tracks (or playlists) after waking up in the morning and while

warming-up with friends to go to a party. Additionally, studies [34; 35] have shown that mood and emotion regulation is one of the most common listening intents users have. Taking both the listening intent and the context into account is probably the most difficult challenge MRSs face since these factors are highly dynamic and hard to measure, i.e., tough to express in data.

2.5.2 Music Playlist Continuation

The task of music playlist continuation, also commonly referred to as automatic playlist continuation [10], is an extension of the automatic music playlist generation (MPG, Bonnin et al. [36]) task. Basically, a playlist is defined as a sequence of tracks that are intended to be listened to together. Therefore, the task of MPG includes *(1)* choosing the tracks contained in that sequence and *(2)* arranging them in a particular order. On the other hand, the MPC task extends MPG by adding one or more tracks to an existing playlist in a way that matches this playlist's characteristics, thus extending a given sequence of seed tracks. Two main goals of this task are: continuing a user's listening session beyond the end of a finite-length playlist and supporting a user while creating playlists by suggesting other fitting tracks.

The most common approaches to MPC are based on both content-based and collaborative techniques. On the one hand, all available tracks in the catalogue have to be indexed to form some kind of background knowledge. On the other hand, deciding which tracks match a playlist's seed tracks is typically based on playlists created by other users. By combining these techniques, an MRS then aims at extending a given playlist with appropriate tracks in a sequence that is as "satisfying" [36] to a user as possible.

2.6 Related Work

The related work I discuss in this section is split into two parts: work that combines the concepts of evolutionary computation or swarm intelligence with the field of (music) recommendation and work that aims at solving the music playlist continuation task. However, it can be stated in advance that no other research has been found that *(a)* compares EC and SI approaches for music recommender systems or *(b)* applies EC and/or SI techniques to the specific task of MPC.

2.6.1 Using Evolutionary Computation or Swarm Intelligence in Recommender Systems

Many research papers that describe the use of evolutionary computation methods for (music) recommender systems or propose new methods were found in the state-of-the-art analysis by Horváth and Carvalho [37]. They discuss a broad array of relevant aspects, such as used recommendation techniques, datasets, and evaluation methods, and classify more than 65 publications in this field. Furthermore, Peska et al. [38] review and analyse more than 77 research papers concerning swarm intelligence techniques in RSs/MRSs. Based on these meta-analyses, some of the more relevant and related publications are presented in the following.

Bobadilla et al. [39] have created a method to improve the similarity function used in RSs based on collaborative filtering by utilising a genetic algorithm. They have observed quality and speed improvements compared to traditional similarity measures, such as the Pearson correlation and the Cosine similarity. Their analysis was focused on the area of movie recommendation. Alhijawi et al. [40; 41] have extended that approach, but their algorithm does not require any existing similarity measures to build on. They start with initially random similarity values, which has yielded even better results compared to Bobadilla et al. [39]. Wei et al. [42] recently published an algorithm (HP-MOEA) that combines both a genetic algorithm (the NSGA-II) and an evolutionary algorithm (the SMS-EMOA) for multi-objective optimisation problems. They have used two stages, first the NSGA-II and then the SMS-EMOA, which results in higher hypervolume values than other multi-objective EAs in the field of movie and book recommendation.

Mocholí et al. [43] proposed an algorithm for generating music playlists based on ant colony optimisation. The algorithm utilises metadata, such as a track's genre or artist, to build solutions w.r.t. sets of restrictions (e.g., genres or artists). Bedi and Sharma [44] have created an ACO algorithm based on trust between the ant colonies to improve the neighbourhood similarity function in a CF-based RS. This approach has shown positive effects for movie recommendations in terms of precision.

2.6.2 Solving the Music Playlist Continuation Task

The task of music playlist continuation is relatively new in music recommendation research and mainly got attention during the *RecSys Challenge 2018*. This challenge was particularly focused on MPC based on the *Spotify Million Playlist Dataset* (MPD); more details in section 3.1. The associated analysis [45] the challenge's organisers provide gives insight into the participants' approaches. I will not go into much detail regarding these approaches since that would go way beyond the scope of this thesis. However, interestingly, none of the participants used any form of evolutionary computation or swarm intelligence in their work.

Many top-performing participants used a two-stage architecture of a collaborative filtering algorithm and either a neural network or a 'learning to rank' model. In the first stage, matrix factorisation [46] was used as the dominant approach in CF, which, i.a., aims at predicting missing values in the playlist-track matrix given by the MPD. The second stage was usually focused on re-ranking the output of the first stage model to improve accuracy. While some participants built convolutional or recurrent neural networks [47; 48] to learn relationships of tracks in the playlists, others built tree-based 'learning to rank' models [49] based on, e.g., XGBoost [50] or GBDT [51], to re-rank tracks.

2.7 Summary

Over the course of this chapter, I have explained all the major concepts that are important for this thesis. I started by detailing the computational intelligence areas of evolutionary computation and swarm intelligence and clarified the underlying ideas of some commonly utilised algorithms, such as evolutionary algorithms, genetic algorithms, and ant colony optimisation algorithms. Further on, I have addressed

the class of multi-objective optimisation problems, how to incorporate metaheuristics to find a set of *possibly best*, non-dominated solutions, and how to compare these solutions with the aid of the hypervolume metric. Then, I have changed the focus to the research field of recommender systems and their three main approaches: collaborative filtering, content-based filtering, and hybrid filtering. I have stated the primary types of data RSs work with, i.e., user data, item data, and user-item preference data, and pointed out some key challenges an RS faces, including the cold-start problem, the long-tail problem, and data sparsity. Building upon the knowledge about general RSs, I have taken a closer look at the field of music recommendation afterwards. I have touched upon the most significant particularities MRSs entail compared to other areas of recommendation, like the short duration of tracks and the typical sequential consumption of tracks. Moreover, I have explained the specific MRS task of music playlist continuation, in which a set of seed tracks is supposed to be extended. Finally, I have taken a look at related research publications, both in the field of using EC/SI techniques in an RS/MRS and in the field of solving the MPC task. With all that laid out, I can now answer the first two research questions this thesis covers.

2.7.1 Learnings from Research

As stated in the introduction and shown in the related work section, the research on combining evolutionary computation or swarm intelligence with general recommender systems is already quite advanced. However, the most popular application area in that research is, by far, the field of movie recommendation. Even though music recommendation can build upon general RSs, and movies and music are comparable and similar to some extent, the research on MRSs is still sparse, especially in the field of incorporating EC and SI techniques. To answer **RQ1**, I will now lay out some key learnings and observations from that advanced research on RSs based on EC and SI, and what can be adopted in MRSs.

- MRSs typically have access to both content descriptions (e.g., track features) and user preferences (e.g., liked tracks or listening logs); thus, combining CF and CBF into a hybrid method is suitable.
- EC and SI techniques can primarily be utilised at two points in an RS: improving the similarity function of CF-based and CBF-based methods and optimising results found in a prior step.
- Finding similar items, i.e., tracks, with the help of neighbourhoods and traditional information retrieval methods is already very advanced and results in valuable recommendations. Hence, optimising these results by using EC and SI techniques even further is a very promising approach.
- Using the hypervolume metric as a baseline for optimising results is already the most widely used technique and probably the most auspicious.
- Integrating a user's listening intent and context into MRSs is very challenging with today's methods, thus often neglected or just barely touched. EC and SI techniques have not been found significantly better at tackling that.

- MRSs oftentimes have access to massive datasets (in comparison to other recommendation areas) regarding both the magnitude of tracks and the variety of track features. This allows a more profound result generation, however, might entail performance and scalability issues. Using EC and SI techniques might help narrowing down the search space to the essential aspects.

2.7.2 Application and Possible Improvements

To answer **RQ2**, I will now take a look at different ways of applying evolutionary computation and swarm intelligence techniques to music recommendations. The main approaches have already been mentioned in the learnings from research above; however, I will go into a little more detail here and further explain how they can improve the results and to what extent improvements can be expected at all.

Feature selection. The great amount of tracks and track features a music recommender system can utilise for making recommendations is not easy to handle, especially in terms of performance. A common approach to tackle this issue is to narrow down the search space, i.e., selecting the most significant features. Research concerning the use of EC and SI techniques for feature selection in machine learning [52] and hyperparameter tuning for neural networks [53] has already put forth promising approaches. In MRSs, (track) feature selection could be supported by EC and SI techniques, e.g., by repeatedly selecting some track features and testing which have the most influence. Any type of feature search space narrowing would certainly be beneficial in terms of performance.

Similarity function. One of the main components in any RS is the similarity function used to find similar users or items. Research on enhancing this similarity function with EC [39; 40] in, i.a., CF-based algorithms has already shown improvements compared to only using traditional similarity methods. Applying this approach to MRSs, which typically incorporate even more data on which to base similarity, would certainly result in more profound methods of finding similar tracks. However, without adequately selecting the 'right' features to consider in the similarity function, it might lead to false confidence in how similar tracks actually are. Moreover, some MRS-specific challenges, such as considering a user's listening intent and context, would not benefit at all from improving the similarity function.

Result calculation. As mentioned in section 2.6.2, many participants in the *RecSys Challenge 2018* approached the MPC task by using neural networks for the final result generation. This approach can be traced back to the commonly used method of building a comprehensive machine learning model based on the available training data, which can then be used to produce results for some test data (in this case, the playlists to continue). EC and SI techniques could be an alternative for this process so that results are directly calculated by EC and SI algorithms without using a neural network. In terms of the MPC task, this could be achieved by using the hypervolume metric as a baseline for optimising a set of solutions generated in a prior, similarity-based step. One main difference between these approaches would be the performance of 'requesting' results, i.e., generating tracks that continue given seed tracks. Neural networks have the advantage that they can be build once and deliver requested results relatively fast. Calculating results with EC and SI algorithms would

be more of an 'on-the-fly' process, which would definitely lead to a higher compute time for each request. However, this loss in performance might be worth it when the result quality is higher, as seen in [42].

For the rest of this thesis, the focus lies in the third approach, i.e., using evolutionary computation and swarm intelligence techniques for calculating and optimising results in the context of MPC. I chose this approach mainly due to the fact that there is no other research on this.

3. Study Design

This chapter gives a detailed description of this thesis' main research objective: comparing different evolutionary computation and swarm intelligence algorithms for music playlist continuation. I will start with introducing the *RecSys Challenge 2018*, which is the baseline of the whole study and essential for defining the conducted problem. The actual problem definition follows then and includes the starting situation and how to compute the best solutions. Then, I will explain the different objectives that the examined algorithms aim to optimise and the different algorithms themselves. Finally, the implementation of the study, its components, its evaluation methods, and how it is conducted will be described.

3.1 RecSys Challenge 2018

The *RecSys Challenge 2018* is an ongoing[8] challenge in the field of music recommendation. It is the most recent and largest contest for music recommender systems, and therefore an important reference point in current research and this thesis. As mentioned in section 2.6, a description and subsequent analysis are available in [45; 54]. This challenge, as this thesis, is focused on the MRS task of music playlist continuation and introduced the *Spotify Million Playlist Dataset* as its underlying dataset. The MPD consists of one million Spotify user playlists, each with title, track list, and additional metadata. For validation and evaluation in the context of the *RecSys Challenge 2018*, a separate challenge dataset was used that includes 10,000 additional but incomplete playlists, divided into ten different challenge playlist types (such as "title and first 100 tracks" or "no title and first 10 tracks"). The goal of the challenge was to extend every challenge playlist with an ordered list of 500 recommended tracks, resulting in the task of MPC.

The participants' submissions were evaluated based on three core metrics: R-precision, normalized discounted cumulative gain (NDCG) [55], and recommended songs clicks. *R-precision* measures the fraction of recommended relevant tracks among the number of withheld tracks on both the track and the (proportionate) artist level. *NDCG*

[8] https://www.aicrowd.com/challenges/spotify-million-playlist-dataset-challenge

assesses the quality in ranking the recommended tracks, i.e., takes the order of items into account (in contrast to the R-precision). *Recommended songs clicks* relates to the Spotify feature "Recommended Songs" and describes the number of refreshes needed before the first relevant track in a list of 10 tracks, respectively, is found. Each of these three metrics is calculated for all challenge playlists and accumulated overall and by playlist type. Higher R-precision and NDCG and lower recommended songs clicks scores are better.

3.2 Problem Definition

As a starting point of the study, a definition of the problem is required. Let P_t be the large set of training playlists and P_c the smaller set of incomplete challenge playlists that can be derived from the *RecSys Challenge 2018*. For all playlists in P_c, each having a title and containing X seed tracks, the goal is to find K additional tracks to recommend; in this case, $K = 500$. Each playlist $P_{c_i} \in P_c$ is considered as one independent instance of the music playlist continuation task, i.e., its own MPC problem. The general approach for finding the best solutions for every problem can be summoned as follows:

1. find the set of playlists P_{s_i} similar to the constituent playlist P_{c_i}
2. create the set of candidate tracks T_{c_i} containing the tracks of all playlists in P_{s_i}
3. determine the best permutations of T_{c_i} by applying the examined algorithms

In the first step, namely finding similar playlists, a standard neighbourhood-based algorithm calculates distances between P_{c_i} and all playlists in P_t. The totalled number of tracks L in the resulting set P_{s_i} must exceed K, i.e., the most similar playlists in P_t will be chosen so that $L > K$. The resulting set of L candidate tracks T_{c_i}, together with the X tracks in P_{c_i}, will be the base for the individual solutions in the subsequent optimisation. More specifically, the fitness functions of the algorithms evaluate solutions containing the X seed tracks fixed at their positions in P_{c_i} and the first $Y = L - X$ tracks in T_{c_i}, resulting in a total number of $K = Y$ evaluated tracks per solution. Since solutions are simply different orders of the L tracks in T_{c_i}, the MPC task is considered a permutation problem.

3.3 Objectives

While computing the best solutions, i.e., non-dominated solutions, for each challenge playlist $P_{c_i} \in P_c$, the algorithms need a way of evaluating these solutions. More specifically, they have to calculate the fitness values to reward good solutions, i.e., solutions that are better than others. Additionally, not all examined algorithms evaluate only complete solutions, i.e., different permutations of the candidates; a way of evaluating single candidates is required as well.

This thesis considers three objectives to evaluate solutions and candidates: accuracy, novelty, and diversity. For a long time, accuracy (i.e., precision and recall scores) was the primary objective RSs took into account [56]. Besides, so-called beyond-accuracy

measures (i.a., diversity, novelty, and serendipity) have gained much attention and proved valuable for recommendations [57]. Serendipity will be neglected in this thesis since it is rather fuzzy and subjective, i.e., hard to measure without explicit user feedback.

As a baseline for the optimisation, this thesis will be oriented towards the *Pareto-Efficient Hybridisation* approach proposed in [56]. This approach is mainly based on the novelty and diversity framework by Vargas and Castells [58] and additionally calculate accuracy based on precision and recall scores. The general approach of how the different objectives are calculated follows now.

3.3.1 Accuracy

Generally speaking, accuracy describes the overlap between one set of candidates and another [59] or the presence of one candidate in a set of candidates. Here, accuracy is defined by the number of hits, i.e., matches, between all tracks of a solution and each similar playlist in P_{s_i}. Additionally, the accuracy score is weighted by the similarity value of each matched playlist in P_{s_i}.

3.3.2 Novelty

The novelty of a set of candidates or one single candidate generally refers to how different it is w.r.t. what has been previously "seen" [58]. In the context of this thesis, novelty depends on how rarely a track appears in the similar playlists P_{s_i}. Each of the L candidate tracks appears between 1 and $|P_{s_i}|$ times, thus approximating the track popularity. Inverting this popularity score results in a numeric assessment of how novel a track is.

3.3.3 Diversity

Diversity generally describes how different the candidates in a set are w.r.t. each other [58]. Here, this difference is represented by the average pairwise dissimilarity between every two tracks in a solution. Dissimilarity is calculated by inverting the Cosine similarity between two tracks, which builds upon the distance between the tracks' feature vectors. These track features contain information about a track, like the artist, the album, and audio features such as energy and tempo; they will be detailed further in section 4.3.

3.4 Algorithms

This section introduces the examined algorithms for the study, gives an overview of how they generally work, and details which parameters they depend on.

3.4.1 NSGA-II

The *Non-dominated Sorting Genetic Algorithm II*, NSGA-II [14], is a widely used dominance-based multi-objective genetic algorithm. It incorporates fast non-dominated sorting, crowding distance assignment, and an elitism selection process.

Algorithm 4 NSGA-II pseudocode [14; 15]

Input: N, p_c, p_m
1: $t \leftarrow 0$
2: $P_t \leftarrow create\_population(N)$
3: $evaluate\_population(P_t)$
4: **while not** termination condition is reached **do**
5: $\quad Q_t \leftarrow selection(P_t, N)$
6: $\quad Q_t \leftarrow crossover(Q_t, p_c)$
7: $\quad Q_t \leftarrow mutation(Q_t, p_m)$
8: $\quad evaluate\_population(Q_t)$
9: $\quad R_t \leftarrow P_t \cup Q_t$
10: $\quad fast\_non\_dominated\_sorting(R_t)$
11: $\quad crowding\_distance\_assignment(R_t)$
12: $\quad P_{t+1} \leftarrow non\_dominated\_solutions(R_t)$
13: $\quad t \leftarrow t + 1$
14: **return** P_t

As presented in *Algorithm 4*, the NSGA-II is parameterised by the population size N, the crossover probability p_c, and the mutation probability p_m. Additionally, a termination condition has to be defined, which might depend on the number of generations, the total evaluation count, or the elapsed time. First, a random initial population with N individuals is created (*L2*) and evaluated for each of the objectives (*L3*). At the beginning of each generation t, a children population Q_t is selected that has the same chromosome (i.e., track) structure as its parent population P_t (*L5*). The selection function focuses on elite individuals of the population by using non-dominated ranking and crowding distance. In the next two steps, Q_t then undergoes the genetic crossover and mutation operations; these are closely oriented towards the genetic behaviour in nature. During the crossover (*L6*), the individuals and their chromosomes, respectively, get disarranged - every time with a probability of p_c. This mixed-up population then gets randomly (with a probability of p_m) mutated (*L7*), i.e., some chromosomes get swapped, their values get slightly changed, or another way. Then, the offspring population is evaluated for each objective (*L8*) and combined with the parent population (*L9*). The resulting population R_t is fast non-dominated sorted (*L10*), in which the ranking of an individual depends on the non-dominated front their chromosomes lay on. In addition, the crowding distances are assigned to all individuals of R_t (*L11*), which measures how close an individual is to its neighbours and aims at maintaining diversity of the Pareto fronts. Finally, the parent population for the next generation, P_{t+1}, is calculated by finding all non-dominated individuals of R_t (*L12*). When the termination condition is met, the algorithm returns the most recent population (*L14*).

3.4.2 SMS-EMOA

The *S-Metric Selection Evolutionary Multi-Objective Algorithm*, SMS-EMOA [60], aims explicitly at maximising the dominated hypervolume (S-metric) during an evolutionary process.

Algorithm 5 SMS-EMOA pseudocode [60]

Input: N, p_c, p_m
1: $t \leftarrow 0$
2: $P_t \leftarrow create\_population(N)$
3: $evaluate\_population(P_t)$
4: **while not** termination condition is reached **do**
5: $\quad Q_t \leftarrow selection(P_t, 2)$
6: $\quad q_t \leftarrow crossover(Q_t, p_c)$
7: $\quad q_t \leftarrow mutation(q_t, p_m)$
8: $\quad evaluate\_individual(q_t)$
9: $\quad R_t \leftarrow P_t \cup \{q_t\}$
10: $\quad fast\_non\_dominated\_sorting(R_t)$
11: $\quad P_{t+1} \leftarrow reduced\_solutions(R_t)$
12: $\quad t \leftarrow t + 1$
13: **return** P_t

The SMS-EMOA, as detailed in *Algorithm 5*, resembles the NSGA-II in many aspects and is parameterised in the same way: the population size N, the crossover probability p_c, and the mutation probability p_m. Generally, it also creates and evaluates an initially random population (*L2-L3*) and modifies this population in each generation until the termination condition is reached. The fundamental difference is that only one child is generated per generation, and it only becomes a member of the next population if replacing another individual leads to a higher hypervolume of the population. Generating the child, q_t, is achieved by selecting two parent individuals (*L5*) and applying the crossover operation (*L6*). After applying the mutation operation (*L7*) to the child and evaluating it (*L8*), the resulting population R_t combines the parent population and the offspring (*L9*). Then, exactly as in the NSGA-II, R_t is fast non-dominated sorted (*L10*); however, the parent population for the next generation, P_{t+1}, is not created by finding all non-dominated individuals of R_t but by removing the one individual that contributes the least value to the populations hypervolume. More specifically, the reduction function (*L11*) splits the population into differently ranked, non-dominated sets of individuals, i.e., fronts, and discards one individual from the worst-ranked front. Over the course of the generations, this guarantees the steady rise of the population's hypervolume. In the end, the algorithm returns the most recent population (*L13*).

3.4.3 m-ACO

The *Multi-Objective Ant Colony Optimisation* algorithm, m-ACO [61], is a generic ACO algorithm for multi-objective problems. The algorithm follows the MAX-MIN Ant System [62] scheme, extending the original Ant System algorithm [17] in two main aspects: rewarding only the best solutions during the pheromone update and keeping the amount of deposited pheromones in certain bounds. In contrast to the previously discussed evolutionary computation algorithms, m-ACO requires that the objectives' fitness functions return smaller values for better solutions; hence multiplying the fitness values by -1 is needed to maintain compatibility with the other algorithms. Furthermore, the m-ACO algorithm offers four different variants,

each differing in the number of ant colonies and pheromone trails used and in the implementation of choosing candidates and computing pheromones to deposit.

Algorithm 6 m-ACO pseudocode [61]

Input: N_c, N_p, N_t, N_a, V, α, β, ρ

$t \leftarrow 0$
$S \leftarrow \emptyset$
$P \leftarrow initialise\_pheromone\_trails(N_p, V)$
while $t < N_t$ **do**
    for each colony c **in** $1...N_c$ **do**
        $S_c \leftarrow \emptyset$
        for each ant a **in** $1...N_a$ **do**
            $s \leftarrow construct\_solution(c)$
            $S_c \leftarrow S_c \cup \{s\}$
        $S \leftarrow S \cup best\_solutions(S_c)$
    for each pheromone trail p **in** P **do**
        for each candidate v **in** p **do**
            $\tau^p(v) \leftarrow updated\_pheromone\_trail(\tau^p(v))$
    $t \leftarrow t + 1$
return S

function construct_solution(c)
    $s \leftarrow ()$
    $V' \leftarrow V$
    while $V' \neq \emptyset$ **do**
        $v \leftarrow choose\_with\_probability(V', p_s^c(v))$
        $s \leftarrow (s_0, \ldots, s_{|s|}, v)$
        $V' \leftarrow V' \setminus \{v\}$
    return s

function updated_pheromone_trail($\tau^p(v)$)
    $\tau^p(v) \leftarrow \tau^p(v) * (1 - \rho)$
    $\tau^p(v) \leftarrow \tau^p(v) + \Delta\tau^p(v)$
    $ensure\_bounds(\tau^p(v))$
    return $\tau^p(v)$

As seen in *Algorithm 6*, the m-ACO is parameterised by eight inputs, namely:

N_c	the number of ant colonies
N_p	the number of pheromone trails
N_t	the number of cycles before the algorithm terminates
N_a	the number of ants per colony
V	the set of candidates
α	the pheromone factors weight $*$
β	the heuristic factors weight $*$
ρ	the evaporation factor

$*$ not directly used in the pseudocode but in the solution construction

The first two parameters, N_c and N_p, are defined by the above-mentioned variant of the m-ACO algorithm. More precisely, given the number of objectives, m, the problem incorporates, the two parameters are defined as follows:

Variant	N_c	N_p
m-ACO$_1$	$m+1$	m
m-ACO$_2$	$m+1$	m
m-ACO$_3$	1	1
m-ACO$_4$	1	m

In this thesis, the focus lies in the first variant, m-ACO$_1$, since a preliminary study has shown it yields the best results compared to the other variants (more on that in section 5.1). The following description of the algorithm, therefore, applies mainly to the m-ACO$_1$ variant. As described in section 3.2, three objectives are considered; hence the parameters $N_c = 4$ and $N_p = 3$ are used. According to the m-ACO paper [61], there are three single-objective colonies, each with a dedicated pheromone trail and one extra multi-objective colony that aims at optimising all objectives. The extra colony uses a randomly chosen pheromone trail in each calculation and the sum of heuristic information associated with all objectives. Additionally, the set of candidates, V, equals the L candidate tracks defined in section 3.2. The other five parameters are configurable and will be detailed in section 3.4.3.

In the beginning, the three pheromone trails, P, are initialised (*L3*), each containing the maximum pheromone value $\tau_{\max}$ for all candidates $v \in V$ on the trail. During each cycle, every colony constructs a set of solutions, S_c, from which the best ones are selected by means of a measure that depends on the regarded problem (*L10*). Here, the three single-objective colonies select the best solutions based on their respective objective, and the multi-objective colony selects all non-dominated solutions. These sets of local best solutions of each colony are added to the set of global best solutions S (*L10*). After all colonies have constructed their solutions, the pheromone trails are updated (*L11-L13*). When all cycled are done, the algorithm returns the global best solutions (*L15*).

Constructing Solutions

Each ant in every colony constructs its own solution per cycle. Starting with an empty solution, s, (*L17*) and a full set of candidates, V', (*L18*), an ant repeatedly selects one candidate (*L20*), adds it to the solution (*L21*), and removes it from the candidate set (*L22*). The candidate selection is based on a probability distribution that is defined as follows. Let $v \in V'$ be a remaining candidate, let c be the current colony, and let s be the list of already chosen candidates; the probability of selecting v is:

$$p_s^c(v) = \frac{[\tau_s^c(v)]^\alpha \cdot [\eta_s^c(v)]^\beta}{\sum_{v_i \in V'} [\tau_s^c(v_i)]^\alpha \cdot [\eta_s^c(v_i)]^\beta} \tag{3.1}$$

where $\tau_s^c(v)$ and $\eta_s^c(v)$ are the pheromone and the heuristic factors, respectively, and α and β are the respective weights.

Updating Pheromone Trails

As usual in all ACO algorithms, updating the pheromone trails is a two-step process and happens once all ants have constructed their solutions. First, the pheromone values for each candidate is reduced by a constant factor (*L25*) to simulate evaporation, similar to the actual process in nature. Then, new pheromones are deposited on the best candidates (*L26*), depending on the colony. Finally, the updated pheromone values are revised in order to stay within the defined bounds $\tau_{\min}$ and $\tau_{\max}$.

For all single-objective colonies, pheromones are deposited on the candidates of the local best solutions found by the i^{th} colony. The quality of a solution, i.e., its fitness, is evaluated by the corresponding function f_i for each objective i. Let s^i_{local} be the local best solution that minimises f_i for the current cycle, and let s^i_{global} be the global best solution that minimises f_i over all cycles since the beginning. The amount of pheromones to deposit on a candidate v on the i^{th} pheromone trail is defined as follows:

$$\Delta\tau^i(v) = \begin{cases} \dfrac{1}{1 + f_i(s^i_{local}) - f_i(s^i_{global})} & \text{if } s^i_{local} \text{ contains } v \\ 0 & \text{otherwise} \end{cases} \tag{3.2}$$

The fourth colony, which regards all objectives, determines a set of local best solutions: one per objective. It deposits pheromones on the trails corresponding to the objectives with the same formula.

3.4.4 NOOP

The *No-Operation* algorithm (NOOP) is a very simple one and basically does no type of optimisation and only returns a set of random solutions. Its sole purpose is to create a reference point for the subsequent analysis of the optimisation quality the other algorithms provide.

4. Implementation

In this chapter, I circumstantiate the implementation of the previously explained study design. I start by giving an overview of the technology stack used, and how the problem, a solution, and the data is encoded. After that, I describe which components are used to aid the result calculation and optimisation. Finally, I go into detail on how the objective functions and algorithms are represented in code.

4.1 Technology Stack

This section gives an overview of the technologies used in the thesis' project. Along with the general setup, the utilised 3rd-party libraries and their benefit for the implementation will be detailed.

Setup

When choosing a suitable programming language for this project, a broad list of candidate languages commonly used in data-related computations can be found. Python and Java are the languages on the shortlist since both offer a good amount of external libraries specialised in recommendation tasks. Due to personal preferences and experiences, I selected Java to be the basis of the whole implementation. The build tool used in the project is Gradle. It allows a simple way to handle dependencies and building the Java project.

The project needs to parse, process, and store a lot of data in an efficient way. Since the computation can be split into a series of tasks, each producing intermediate results, there is no need for a classical database like (No)-SQL-based systems. Instead, a combination of a file-based system (i.e., CSV files) to store intermediate data and loading the needed data into the RAM is used.

Libraries

One of the most comprehensive frameworks for multi-objective optimisation with metaheuristics is *jMetal*[9] by Antonio J. Nebro. It offers a broad range of useful

[9] https://github.com/jMetal/jMetal

interfaces and classes that can be combined to run and evaluate the elaborated algorithms. The other heavily used external library is *RankSys*[10] by Saúl Vargas, which is the Java implementation of the novelty and diversity framework by Vargas and Castells [58]. It provides a range of interfaces and algorithms I use for finding similar playlists and similar tracks. Additionally, I also use some helper classes provided in Google's popular *Guava*[11] Java utility library. Moreover, some more helper libraries are used, which will be mentioned later in this section at their specific use cases.

4.2 Problem and Solution Encoding

As an actual starting point of this thesis' study implementation, this section gives an overview of how the music playlist continuation problems and their solutions are encoded. To recall: each challenge playlist, P_{c_i}, in the *RecSys Challenge 2018* is considered as one independent problem to solve. Hence, each problem takes the X given seed tracks and a list of similar playlists, P_{s_i}, as input. Solutions of each problem are different permutations of the L candidate tracks, T_{c_i}, contained in P_{s_i}. Both encodings are based on corresponding interfaces provided by the *jMetal* library.

Two types had to be considered for the problem and solution encoding: full permutations and growing permutations. They only differ in the way solutions are created, more on that down below. Once a solution has been fully created, it gets evaluated w.r.t. the objectives defined for all problems. The evaluation is a two-step process: *(1)* an intermediate list with the seed tracks and the candidate tracks in the solution is created, and *(2)* the fitness values of this list are calculated for each objective. Using this intermediate list makes sure that the evaluation is context-based, i.e., that the candidate tracks are not evaluated independently, but in combination with the seed tracks they continue. For that, the `FixedBaseList` data structure is used. This list starts with a position-aware list of elements, in this case, the X seed tracks at their corresponding positions in P_{c_i}. Then, the candidate tracks of the solution are filled in the remaining positions in the list. Storing the fitness values for a solution is handled by the *jMetal* implementation.

Type I: Full Permutation

The evolutionary computation algorithms, NSGA-II and SMS-EMOA, require full permutations, meaning solutions already contain all candidate tracks from the beginning. *jMetal* provides the suitable `PermutationSolution` class, in which all candidate tracks are simply shuffled on creation.

Type II: Growing Permutation

The m-ACO algorithm needs a different implementation because, as mentioned in section 3.4.3, the ants must be able to create solutions iteratively (starting with an empty solution). Hence, a custom `GrowingSolution` interface was added to adapt to these particularities. However, the implementing class ensures that every candidate

[10] https://github.com/RankSys/RankSys
[11] https://github.com/google/guava

track can only be added once to each solution to maintain compatibility with the concept of permutations. Furthermore, the computation of the m-ACO heuristic factor, $\eta_s^c(v)$, requires a way of evaluating a single candidate, here v, w.r.t. the specific objective defined by the colony c.

Note: In ACO algorithms, the heuristic factor of a candidate v usually also depends on the partial solution s, which contains the already selected candidates on the current trail. However, implementing this behaviour would have resulted in infeasible compute times of the m-ACO algorithm. The reason lies with the (permutation) problem itself and the high number of candidate tracks ($L > 500$), which would have meant that the number of fitness evaluations goes way beyond 500! (factorial). By neglecting the partial solutions, the heuristic factor only has to be determined once per candidate track. This, on the other hand, resulted in also neglecting the diversity objective for the heuristic factor of single candidates; more on that in section 4.5.3.

4.3 Data Model

The data model for this project is oriented towards the data provided by the *Spotify Million Playlists Dataset* and the associated challenge playlists of the *RecSys Challenge 2018*. After stripping all non-relevant and redundant data, the resulting model looks as follows:

Tracks are identified by a unique ID and have an artist ID, an album ID, and a set of audio features. These audio features are obtained by the Spotify API[12] and will be detailed in the next section. From this data, a set of track features can be obtained for each track.

Track features are 3-tuples described by a dimension, an identifier, and a value. The dimension is either `ARTIST`, `ALBUM`, or `AUDIO`. The identifier is a dimension-related attribute, for example containing an artist ID or the name of the audio feature. The value is a numerical expression of the track feature and is discrete for the first two dimensions and continuous for the `AUDIO` dimension. Example track feature entries can be found in Appendix A.1.

Playlists have a unique ID, a list of tracks and corresponding positions, and a set of playlist features. All playlists in the dataset have complete lists of tracks, meaning all track positions are known. The challenge playlists may be incomplete and might miss tracks at certain positions.

Playlist features, similar to track features, are also 3-tuples described by a dimension, an identifier, and a value. However, playlist features are computed as the sum (for discrete dimensions) or the average (for the continuous `AUDIO` dimension) of all its tracks, i.e., its track features. Additionally, a fourth `TRACK` dimension contains all the track IDs of the playlist. Example playlist feature entries can be found in Appendix A.2.

SimilarTracksLists are subtypes of playlists and only contain a list of tracks and their overall similarity to a (challenge) playlist. The positions of the tracks are not relevant, and the similarity score ranges from 0% to 100%.

[12] https://developer.spotify.com/documentation/web-api/reference/#endpoint-get-several-audio-features

4.4 Components

A couple of semi-independent components are used in the implementation of the study. More precisely, the following descriptions refer to required tasks that have to work before the algorithms can actually compute results. I start with explaining the general import of the utilised data for the study before presenting my approach for finding similar playlists and tracks and how the algorithms use these.

4.4.1 Data Import

The project uses two data sources: the training playlists (MPD) and challenge playlists from the *RecSys Challenge 2018* and additional track information from the Spotify API. The playlists can be downloaded from the corresponding challenge website[13] and are available in the form of JSON files. While the 10,000 challenge playlists come in one single file (108 MB in size), the 1,000,000 training playlists are split into 1,000 chunk files, each containing 1,000 playlists (5.39 GB in total size). Spotify offers several API endpoints (all returning JSON data) to obtain information about tracks; the *Track Audio Features* API[14] provides the most useful data and includes the following audio features for each track: *acousticness*, *danceability*, *energy*, *instrumentalness*, *key*, *liveness*, *loudness*, *mode*, *speechiness*, *tempo*, *time signature*, and *valence*.

Importing all the data is a three-step process: first, parse all playlists and tracks, then download the audio features for all tracks, and finally store all playlists, playlist features, tracks, and track features. Parsing the playlists and tracks is heavily supported by the Java JSON library *Jackson*[15], which allows straightforward deserialisation of JSON into plain Java objects. The set of unique tracks, i.e., Spotify track IDs, contains a total number of 2,262,292 items for which the audio features are then downloaded. I used the *Spotify Web API Java* wrapper[16] for handling the authentication and the HTTP requests. However, even though an endpoint to get the audio features for several tracks at once exists, it only allows requesting 100 tracks at a time; hence the set of track IDs is split into chunks, each containing 100 IDs. Finally, all playlists, tracks, and their features are stored in CSV files, as described in section 4.1.

4.4.2 Similar Playlists

As defined in section 4.2, each challenge playlist $P_{c_i} \in P_c$ needs a set of similar playlists, P_{s_i}, to create the set of candidate tracks T_{c_i}. More specifically, P_{s_i} consists of a number of `SimilarTracksLists` containing more than K unique tracks in total. All lists can be precomputed and stored before the algorithms run, which results in a faster and more consistent compute time.

[13] https://www.aicrowd.com/challenges/spotify-million-playlist-dataset-challenge

[14] https://developer.spotify.com/documentation/web-api/reference/#endpoint-get-several-audio-features

[15] https://github.com/FasterXML/jackson

[16] https://github.com/thelinmichael/spotify-web-api-java

The implementation is mainly based on concepts the *RankSys* library provides. First, all playlists and playlist features are loaded into so-called `Indexes`, respectively, before linking them to so-called `PreferenceData`. This `PreferenceData` structure builds on the idea that a number of "users", here playlists, have "preferences" for a number of "items", here playlist features. The strength of preference must be numerical and corresponds to the above-mentioned value of a `PlaylistFeature`. Then, a so-called `Neighborhood` is created, which uses the `PreferenceData` and a measure of similarity, in this case, a `UserSimilarity`. *RankSys* provides different `UserSimilarity` implementations based on well-known metrics like the Cosine and the Jaccard similarity. Since an extensive comparison of different similarity measures would go beyond the scope of this thesis, I decided to use the default symmetric Cosine similarity. The `Neighborhood` then allows obtaining a set of similar playlists for each challenge playlist.

Note: One limitation of this approach was that challenge playlists with only a title and no known tracks do not have any playlist features. This problem could be solved by extracting additional playlist features from the playlist title, e.g., by using a natural language processor, but was neglected due to a lack of time. Instead, one tenth of the challenge playlist, namely these without tracks, only uses randomly generated similar playlists, which most likely leads to poorer results in this category. However, since the main focus of this thesis lies in comparing evolutionary computation and swarm intelligence algorithms, this limitation seems bearable.

4.4.3 Similar Tracks

The diversity objective requires a way of calculating the similarity of two given tracks. As mentioned, this similarity can be determined by measuring the distance between the tracks' features. The *RankSys* library also provides useful concepts to calculate this distance between tracks.

First, all track IDs and the related features are loaded into `Indexes`, respectively, which are then linked to a so-called `FeatureData` structure. Similar to the above-mentioned `PreferenceData`, the `FeatureData` structure holds a numerical value for each `Track-TrackFeature` tuple. This data can then be used in a so-called `ItemDistanceModel`, which does exactly what its name promises: returning a distance between two track items, i.e., two track IDs. Here, *RankSys* again provides different implementations using either the Cosine or the Jaccard similarity. Since there were no measurable differences between these, I decided upon using the Cosine similarity for reasons of consistency.

4.5 Objectives

In section 3.3, I have defined which objectives are used for the music playlist continuation problem and how they generally work. This section gives some more details on the implementation and what had to be taken into account for the evaluation of complete solutions and single candidates.

4.5.1 Accuracy

The fitness of a solution w.r.t. to accuracy is the sum of all its candidates fitness values. Accuracy of one candidate track is determined by the number of `SimilarTracksLists` containing the track and factoring in the similarity score of each of these matched lists. The number of occurrences is additionally normalised by the total number of similar tracks lists. For the evaluation of complete solutions, the position of a candidate track in the solution is also factored in. The earlier a track appears in the solution, the higher is the positional factor.

Algorithm 7 Accuracy objective pseudocode

Input: S, T_{sim}

```
1:  fitness ← 0
2:  for i ← 1 to |S| do
3:      track ← S_i
4:      fitness ← fitness + accuracy(track) · |S| / i
5:  return fitness

6:  function accuracy(track)
7:      value ← 0
8:      for j ← 1 to |T_sim| do
9:          if track ∈ T_sim_j then
10:             value ← value + |T_sim| / j · ⟨T_sim_j⟩
11:     return value
```

Algorithm 7 shows the implementation of the accuracy objective, with S being a solution to evaluate and T_{sim} being the similar tracks lists. The angle brackets in *L10* reflect the similarity value associated with the given `SimilarTracksList`. The accuracy function (*L6-L11*) is also used to evaluate single tracks independently.

4.5.2 Novelty

The fitness of a solution w.r.t. to novelty is, as in the accuracy objective, the sum of all its candidates' fitness values. Calculating a candidate track's popularity is also done by counting the `SimilarTracksLists` containing that track but without regarding the similarity scores. After normalising the popularity score w.r.t. the total number of similar tracks lists, it is inverted to reflect the novelty. As in the accuracy objective, the position of each candidate track within a complete solution also contributes to the novelty.

Algorithm 8 Novelty objective pseudocode

Input: S, T_{sim}

1: $fitness \leftarrow 0$
2: **for** $i \leftarrow 1$ **to** $|S|$ **do**
3: $\quad track \leftarrow S_i$
4: $\quad fitness \leftarrow fitness + accuracy(track) \cdot \frac{|S|}{i}$
5: **return** $fitness$

6: **function** novelty($track$)
7: $\quad matches \leftarrow 0$
8: $\quad$ **for each** T_{sim_j} **in** T_{sim} **do**
9: $\quad\quad$ **if** $track \in T_{sim_j}$ **then**
10: $\quad\quad\quad matches \leftarrow matches + 1$
11: $\quad$ **if** $matches = 0$ **then**
12: $\quad\quad$ **return** 0
13: $\quad$ **else**
14: $\quad\quad popularity \leftarrow \frac{matches}{|T_{sim}|}$
15: $\quad\quad$ **return** $\frac{1}{popularity}$

Algorithm 8 shows the implementation of the novelty objective, with S being a solution to evaluate and T_{sim} being the similar tracks lists. The novelty function (*L6-L15*) is also used to evaluate single tracks independently.

4.5.3 Diversity

The fitness value in the diversity objective can, by definition, only be calculated for complete solutions, i.e., a set of candidates, and not for single candidates. The pairwise dissimilarity is determined by summing the distance between all candidate tracks and their successors in the solution, respectively. This implicitly factors in the position of a candidate track, since these at the beginning of a solution contribute more to the fitness, thus have a higher weight. Measuring the distances between two tracks, as explained in section 4.4, returns a numeric value in the range of 0.0 (exactly the same, i.e., no diversity) and 1.0 (no overlap, i.e., highest diversity).

Algorithm 9 Diversity objective pseudocode

Input: S

1: $fitness \leftarrow 0$
2: **for** $i \leftarrow 1$ **to** $|S|$ **do**
3: $\quad$ **for** $j \leftarrow i + 1$ **to** $|S|$ **do**
4: $\quad\quad track_1 \leftarrow S_i$, $track_2 \leftarrow S_j$
5: $\quad\quad fitness \leftarrow fitness + distance(track_1, track_2)$
6: **return** $fitness$

Algorithm 9 shows the implementation of the diversity objective, with S being a solution to evaluate and *distance* being the function that returns a numerical distance between two tracks.

4.6 Algorithms

In section 3.4, I have defined the algorithms used in this study and how they work in detail. This section explains hows the algorithms are implemented and on which parameters they depend.

4.6.1 NSGA-II

Implementing the NSGA-II from scratch was not necessary since this algorithm is part of the *jMetal* library. Along with different variants of the NSGA-II, e.g., the *Steady-State NSGA-II* [63] or the *DNSGA-II* [64], the library also provides an implementation for the standard NSGA-II developed by Deb et al. [14]. I use the standard variant in this thesis because it is by far the most popular one.

As defined in *Algorithm 4*, the NSGA-II requires the following parameters: the population size N, the crossover probability p_c, and the mutation probability p_m. Additionally, the maximum number of evaluations, $T_{\max}$, is used to specify the termination condition. The following values are used for the parameters:

N	$T_{\max}$	p_c	p_m
100	25000	$\{0.85, 0.9, 0.95\}$	$\{0.005, 0.01, 0.015\}$

Table 4.1: NSGA-II parameters

The fixed parameters, N and $T_{\max}$, correspond exactly to the proposed values by Deb et al. For the probability parameters, sets of 3 different values, respectively, were selected to fine-tune in the preliminary study.

The crossover operator uses the *Partially Mapped Crossover* (PMX) method developed by Goldberg [65] in 1985, which is often used for permutation problems and is also provided by *jMetal*. Instead of the widely used crossover probability $p_c = 0.8$, I chose a set of slightly higher values for the parameter tuning because Singh et al. [66] use a value of 0.9 in a more recent paper refining the PMX crossover operator.

jMetal also provides a mutation operator to use for permutation problems: *Permutation Swap Mutation*. It simply swaps two candidates of a solution at random positions with the specified probability p_m in each mutation operation. The probability is usually calculated by $p_m = \frac{1}{N_v}$ (with N_v being the number of candidates, i.e., $N_v = K = 500$), which would result in $p_m = 0.002$. However, since 500 is a very high number of candidates, the resulting p_m would be very low and might cause premature convergences towards local optimums, and is therefore raised a little. Furthermore, a recent study by Garbaruk et al. [67] has shown that even much higher mutation probabilities, e.g., $p_m = 0.3$, can lead to better results (clearly depending on the problem).

For the selection operator, the *Binary Tournament Selection* method is used because Goldberg and Deb found it to be the preferred method in their genetic algorithm selection scheme analysis [68]. A tournament selection is basically a way of selecting the best solution out of a random n-ary subset of all solutions in the population; in this case, with $n = 2$. *jMetal* again provides the implementation for this

operator, which additionally requires a comparator to find the better solution during each tournament. To comply with the definition of the NSGA-II, the `RankingAndCrowdingDistanceComparator` is used.

4.6.2 SMS-EMOA

The implementation of the SMS-EMOA is again very similar to the NSGA-II. In fact, since the *jMetal* library covers the SMS-EMOA as well, all the specifications mentioned above for the parameters and genetic operators are the same for this algorithm. To maintain direct comparability, the constant and variable parameters are also equal:

N	$T_{\max}$	p_c	p_m
100	25000	$\{0.85, 0.9, 0.95\}$	$\{0.005, 0.01, 0.015\}$

Table 4.2: SMS-EMOA parameters

Note: Due to the high similarity to the NSGA-II, a minor copy-and-paste mistake was made in the implementation of the SMS-EMOA. Instead of using a random selection operator (as defined by the algorithm), the same tournament selection operator as in the NSGA-II implementation is used. I mention this for the sake of completeness, but this should not have any negative effects on the results.

4.6.3 m-ACO

The *jMetal* library does not provide an implementation for any ACO algorithm; hence I needed to implement the m-ACO algorithm from scratch. I followed the scheme provided by Alaya et al. [61], which lays out the details of the algorithm and its variants extensively. The general approach has been explained in section 3.4.3, so I only focus on the particularities of the implementation here.

As mentioned, the m-ACO algorithm can be used in four different variants, and the following values are used for the parameters in the preliminary study. They are closely oriented towards the values used in the evaluation by Alaya et al. in [61].

Variant	N_c	N_p	N_t	N_a	α	β	ρ
m-ACO$_1$	4	3	30	100	1.0	$\{2.0, 4.0\}$	$\{0.1, 0.01\}$
m-ACO$_2$	4	3	10	100	1.0	$\{2.0, 4.0\}$	$\{0.1, 0.01\}$
m-ACO$_3$	1	1	10	300	1.0	$\{2.0, 4.0\}$	$\{0.1, 0.01\}$
m-ACO$_4$	1	3	10	500	1.0	$\{2.0, 4.0\}$	$\{0.1, 0.01\}$

Table 4.3: m-ACO parameters

The additional input parameter V (the set of candidates) always equals the L candidate tracks in each music playlist continuation problem, i.e., each challenge playlist.

As described in section 4.2, the m-ACO cannot simply use the full permutation interfaces *jMetal* offers (compared to the NSGA-II and the SMS-EMOA). Hence, this

algorithm utilises the custom `GrowingProblem` and `GrowingSolution` interfaces. Firstly, the said problem interface allows evaluating single candidates w.r.t. a particular objective, i.e., calculating fitness values for a single candidate. Secondly, it enables to determine if a candidate should be rewarded in a certain solution, what is important for the pheromone update in *Equation 3.2*; here, that depends on the position of a candidate in a solution. Since only the first K of the L candidate tracks are influential for the MPC problem, only these are rewarded.

Along with the `MACO` algorithm class, the implementation is based on the following data structures: `Colony`, `Ant`, and `PheromoneTrail`. The algorithm class maintains a set of colonies, which then again each contain a set of ants and a set of pheromone trails. Both the number of colonies and pheromone trails depend on the N_c and N_p parameters, respectively, which are determined by the variant and the number of objectives, m, the problem regards. A `Colony` has three different implementations:

1. `SingleObjectiveColony`: A colony that aims at optimising one of the objectives while using one dedicated `PheromoneTrail`. Utilised for the m objectives in m-ACO$_1$ and m-ACO$_2$.
2. `MultiObjectiveColonyWithSinglePheromoneTrail`: A colony that aims at optimising all objectives while using one `PheromoneTrail`. Utilised in m-ACO$_3$.
3. `MultiObjectiveColonyWithMultiplePheromoneTrails`: A colony that aims at optimising all objectives while using m `PheromoneTrail` instances. Utilised for the extra colony in m-ACO$_1$ and m-ACO$_2$, and in m-ACO$_4$.

The third implementation is additionally parametrised by a pheromone aggregation type, which determines how the pheromone factor of the trails is calculated. For the m-ACO$_1$ and m-ACO$_4$, it is defined as random, and for the m-ACO$_2$, it is summed.

At the beginning of each m-ACO algorithm run (one for each challenge playlist), the heuristic factors used by each colony are initialised. This only needs to happen once because the heuristic factor, i.e., the fitness values, do not change throughout the algorithm run. Then, the classical ACO algorithm cycle is repeated N_t times before returning the global best solutions found. The implementation of this cycle resembles the detailed explanation in *Algorithm 6* to a great extend; solely the *choose_with_probability* function is worth a deeper look. Given a list of remaining candidates to choose from, each with a certain probability, the most suitable implementation for selecting the next candidate can be found in the so-called *Roulette Wheel Selection* method [69]. It is a concept often used in genetic algorithms and is based on the method of discrete probability distribution. For that, I use the `EnumeratedDistribution` class provided by the *Guava* library.

4.6.4 NOOP

The NOOP algorithm simply creates a population of N random solutions, evaluates them once, and returns all non-dominated ones.

5. Evaluation

In the last two sections, I have described how the algorithm study was designed and implemented. Now, I detail how the study was conducted, present the results, and finally, evaluate the results w.r.t. the third research question: which of the evolutionary computation and swarm intelligence algorithms performs best for the music playlist continuation task?

5.1 Study Conduct

Conducting the study was a three-step process: first, all required data was imported and preprocessed, then I have used a preliminary study to find the best algorithm configurations (parameter tuning), and finally, the full study on all challenge playlists was conducted. In this section, I describe all three steps and how the study results were generated.

All the steps ran on a *Google Cloud Compute VM*[17] in the Netherlands, more specific on an *Intel Xeon Scalable Processor (Cascade Lake)* with a 3.4 GHz all-core turbo frequency, 80 vCPUs, and 80 GB of RAM. Additionally, I have used a set of useful helper classes *jMetal* provides to run experiments.

Preparation

In the very first step, the data provided in the *RecSys Challenge 2018* was imported, as described in section 4.4.1. After that, the playlists and tracks were enhanced with the Spotify data, resulting in four data stores: playlist, playlist feature, track, and track feature. Finally, the `SimilarTracksLists` for each challenge playlist were generated and stored, using the techniques described in section 4.4.2.

Note: There was a bug in the challenge playlists import, which I only have discovered after the full study ran. Due to that bug, the challenge playlists were missing seed tracks (some only a few, some most of them). Unfortunately, this had a negative impact on all the subsequent calculations, especially on finding similar playlists.

[17] https://cloud.google.com/compute/docs/cpu-platforms

Preliminary Study

Since every optimisation problem is different, the algorithms' configuration usually influences the quality of results. Hence, the goal of the preliminary study is to find the best parameters for each examined algorithm before running the full study on all challenge playlists. It is a typical step in many machine learning studies and is commonly referred to as *(hyper-)parameter tuning* [53].

Appendix B.1 shows all configurations tested in the preliminary study, as described in section 4.6. The hypervolume value is the average of six challenge playlists across different challenge types and three independent runs, respectively. The NOOP algorithm was also tested and scored a hypervolume value of 0.4439. With this data, the best performing configurations for the NSGA-II and the SMS-EMOA and the overall best performing m-ACO variant/configuration were selected for the full study, namely:

- **NSGA-II**: $N = 100$, $T_{\max} = 25000$, $p_c = 0.95$, $p_c = 0.005$
- **SMS-EMOA**: $N = 100$, $T_{\max} = 25000$, $p_c = 0.95$, $p_c = 0.01$
- **m-ACO$_1$**: $N_c = 4$, $N_p = 3$, $N_t = 30$, $N_a = 100$, $\alpha = 1.0$, $\beta = 2.0$, $\rho = 0.1$

Full Study

These three algorithms and the NOOP algorithm were then used to run the full study on all 10,000 challenge playlists, each with one independent run (more would have been too much computation time for this thesis). They ran in parallel, utilising all available VM CPU cores, for approximately 15 days. To avoid unexpected interruptions, the algorithm runner was designed to be 'fail-resistant' and restarted randomly failed runs automatically; however, this was only needed a few times.

Result Generation

While the algorithms ran, the result tracks and fitness values were stored for each algorithm-playlist combination. After that, the Pareto reference fronts were calculated for each challenge playlist. For that, all non-dominated solutions the four algorithms have generated for each challenge playlist were combined, and one reference front per challenge playlist was created, each containing the lowest-ranked solution w.r.t. the fitness values. The hypervolume value of each algorithm in every challenge playlist was then calculated w.r.t. the corresponding reference front. Finally, the non-dominated solutions of the algorithm with the highest hypervolume were extracted and stored for each challenge playlist. Since these solutions are all equally good by definition, one of these solutions was selected randomly. In the end, one solution file per algorithm was created, each containing 500 tracks for all 10,000 challenge playlists, respectively. These files were then uploaded to the *Spotify Million Playlist Dataset Challenge* website[18] to obtain the R-precision, NDCG, and recommended songs clicks scores.

[18] https://www.aicrowd.com/challenges/spotify-million-playlist-dataset-challenge

5.2 Study Results

Algorithm	Hypervolume	Time	R-Precision	NDCG	Clicks
NOOP	0.1648	16s	0.0669	0.1725	8.7040
m-ACO	0.2896	2330s	0.0780	0.1703	8.8715
NSGA-II	0.6893	3264s	0.0745	0.1809	8.1791
SMS-EMOA	0.7399	4435s	0.0774	0.1830	8.0649

Table 5.1: Results of the full study

Table 5.1 shows the average performance of each algorithm for all challenge playlists w.r.t. all evaluation metrics: hypervolume (between 0.0 and 1.0), the time an algorithm required to compute solutions for one challenge playlist, R-Precision, NDCG, and recommended songs clicks. Higher R-precision and NDCG and lower recommended songs clicks scores are better.

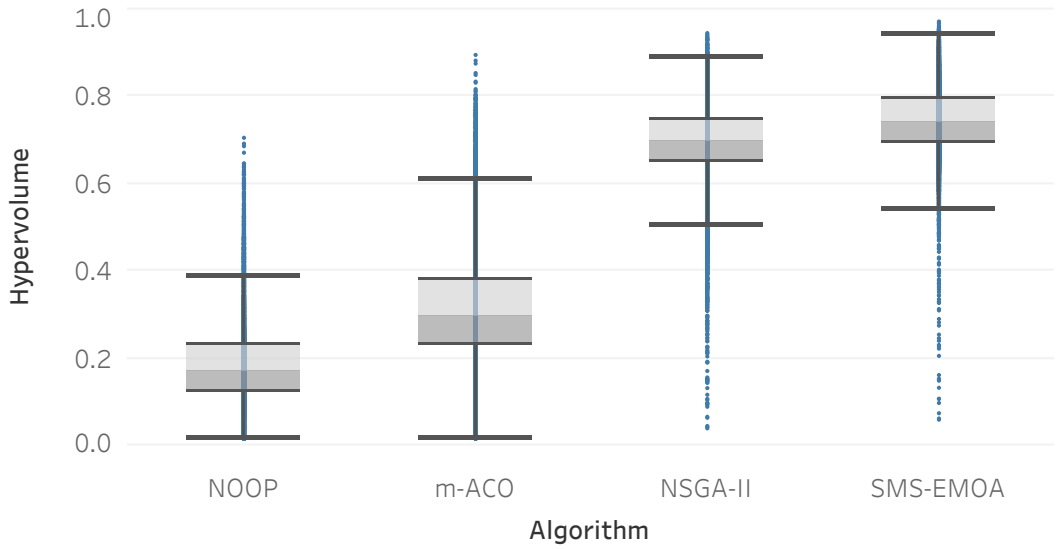

Figure 5.1: Box plot of hypervolume values for each algorithm

Figure 5.1 shows the hypervolume values plotted for each of the four algorithms. It includes the median, the lower and upper quartile (25th and 75th Percentile, respectively), the interquartile range, and the lower and upper whiskers. Each hypervolume data point was calculated w.r.t. the corresponding Pareto reference front.

5.3 Comparing the Algorithms

To answer **RQ3**, I will now compare the examined algorithms w.r.t. their suitability for the music playlist continuation task. First, I will take a look at the performance of the algorithms compared to each other and then compared to the results achieved by other participants of the *RecSys Challenge 2018*.

First of all, the average hypervolume values shown in figure 5.1 reveal that all actual algorithms generate better optimisation results than the random NOOP algorithm. Moreover, there is quite a big gap between the m-ACO algorithm and the NSGA-II and the SMS-EMOA. The 'winner' in terms of hypervolume is the SMS-EMOA, possibly because this algorithm explicitly optimises results based on the hypervolume metric. Interestingly, all four algorithms produce quite many outlier values. Based on these results, one can conclude that the evolutionary computation algorithms return higher-quality results than the m-ACO swarm intelligence algorithm. One reason for that might be the way the m-ACO algorithm (and ACO algorithms in general) creates solutions, as explained in section 4.2. To circumvent massive performance issues this solution creation would have entailed, the diversity objective was neglected, and a poorer quality of results was foreseeable.

When looking at the algorithms' performance in table 5.1, two main observations can be made directly: *(1)* all algorithms (except the NOOP algorithm) take a really long time to find solutions, and *(2)* the differences in hypervolume quality are not reflected in the other metrics, i.e., R-Precision, NDCG, and recommended songs clicks. With an average compute time for one challenge playlist between 38 minutes for the m-ACO algorithm to over 73 minutes for the SMS-EMOA, one can certainly say these are miserable, especially for the MPC task. For real-world MPC applications, probably only a compute time of a few seconds per playlist to continue would be acceptable[19]. Moreover, even though all SI and EC algorithms require an extraordinary amount of time to compute, the results in terms of R-Precision, NDCG, and recommended songs clicks are not significantly better than the NOOP algorithm's; the m-ACO algorithm and the NSGA-II even produce partly worse results than random.

Rank	***Team***	**R-Precision**	**NDCG**	**Clicks**
1	vl6	0.2234	0.3939	1.7845
2	Creamy Fireflies	0.2197	0.3846	1.9252
3	KAENEN	0.2090	0.3746	2.0482

Table 5.2: Results of the top-performing participants in the RecSys Challenge 2018 (creative track) [45]

The comparison with other participants of the *RecSys Challenge 2018* (table 5.2) also reveals an inferior quality of the examined EC and SI algorithms for MPC. It is not meaningful to linearly compare these values with the results of the examined algorithms; however, the significant discrepancy is unmissable and relatively high.

[19]This assumption is based on the performance of Spotify Enhance: a feature that extends an existing playlist with personalised track recommendations within a few seconds. (https://newsroom.spotify.com/2021-09-09/get-perfect-song-recommendations-in-the-playlists-you-create-with-enhance/)

In sum, one must say that all three examined algorithms are **not suitable** for optimising solutions in the context of MPC. They produce nearly random results in terms of the reference metrics given by the *RecSys Challenge 2018* and additionally require an extremely high amount of time to compute. However, compared to each other and in terms of hypervolume, the SMS-EMOA yields the best results.

6. Conclusion

In this chapter, I discuss the findings of the previous chapter as well as some learnings from other research on using evolutionary computation and swarm intelligence techniques in music recommendation systems. Additionally, I point out some aspects and improvements for future work.

Generally speaking, the examined EC and SI techniques used for calculating and optimising results in music playlist continuation are not very promising. With quite high compute times and rather poor solution quality, this approach is far from suitable. However, since the algorithms solely aim at optimising solutions and depend on priorly generated candidate tracks, the bug that caused only partial import of the challenge playlists' seed tracks (see section 5.1) is one certain reason for the poor results. Fixing this issue in future work might reveal a more promising solution quality. Furthermore, this thesis was mainly focused on applying EC and SI methods in the result calculation of MRSs. Other application areas mentioned in section 2.7, i.e., feature selection and similarity function enhancements, are possibly more suitable for EC and SI techniques. Related work, e.g., using genetic algorithms to enhance neighbourhood similarity in collaborative filtering recommender systems (Bobadilla et al. [39] and Alhijawi et al. [40]), has put forth valuable approaches for EC and SI in RSs and MRSs.

Moreover, using only one SI technique in this thesis, i.e., the ant colony optimisation algorithm, is definitely not representable for the rest of the SI field, especially when considering the exposed fact that ACO algorithms are not particularly suitable for permutation problems (see section 4.2). Exploring other SI algorithms in future work, e.g., particle swarm optimisation, might yield better results for MPC and other RS/MRS tasks, as shown by Ujjin and Bentley [70].

Another possible starting point for future work is incorporating the different challenge playlist types into the optimisation. Since it would have gone beyond this thesis' time frame, none of the types, such as "title only", "title and first 5 tracks", or "title and 100 random tracks", contributed to the solution optimisation. This had certainly a negative impact on the result quality, most of all on the results of the

”title only” challenge playlists, for which only random candidate tracks were selected (see section 4.4.2). Using any form of playlist title processing and utilising them at all appears to be a valuable component for MRSs and the MPC task in particular.

In conclusion, the research done in this thesis does not yield any groundbreaking findings that change the way music recommendation works from now on. Additionally, some mistakes were made, which probably led to some degree of diminished informative value about the analysed suitability. However, even though using evolutionary computation and swarm intelligence solely for calculating and optimising solutions seems not as suitable as using alternative machine learning methods, applying these versatile techniques in other (music) recommender system application areas might be.

Appendix

A. Example Track and Playlist Feature Entries

```
73zJaDmjbkSZoIJSMs3lyA,ARTIST#6UWifcscEdbjPgmbevBxZV,1.0
73zJaDmjbkSZoIJSMs3lyA,ALBUM#0c2k2ldJq0CNU1QpMFtufi,1.0
73zJaDmjbkSZoIJSMs3lyA,AUDIO#DANCEABILITY,0.527
73zJaDmjbkSZoIJSMs3lyA,AUDIO#LOUDNESS,-6.204
73zJaDmjbkSZoIJSMs3lyA,AUDIO#TEMPO,126.043
73zJaDmjbkSZoIJSMs3lyA,AUDIO#ACOUSTICNESS,0.0138
73zJaDmjbkSZoIJSMs3lyA,AUDIO#INSTRUMENTALNESS,0.0018
73zJaDmjbkSZoIJSMs3lyA,AUDIO#SPEECHINESS,0.0345
73zJaDmjbkSZoIJSMs3lyA,AUDIO#VALENCE,0.502
73zJaDmjbkSZoIJSMs3lyA,AUDIO#LIVENESS,0.107
73zJaDmjbkSZoIJSMs3lyA,AUDIO#ENERGY,0.772
```

Figure A.1: Example track feature entries stored for the track with the id *73zJaDmjbkSZoIJSMs3lyA*

```
336,TRACK#1TfqLAPs4K3s2rJMoCokcS,1.0
336,TRACK#6ScSkz4CAXfLGEbiMz2Qrk,1.0
336,TRACK#11y8GSP2ASv8S9n0FiiDva,1.0
336,TRACK#3UT4gyENyYB6vQpifOlQwL,1.0
336,TRACK#2H3ZUSE54pST4ubRd5FzFR,1.0
336,ARTIST#6YHEMoNPbcheiWS2haGzkn,2.0
336,ARTIST#3koiLjNrgRTNbOwViDipeA,1.0
336,ARTIST#4L1z1IcfK7lbqx8izGHaw5,1.0
336,ARTIST#0NKDgy9j66h3DLnN8qu1bB,1.0
336,ALBUM#13TOuCZy3QU2Tt1llKPUqc,1.0
336,ALBUM#5jNDWA19BJbE24x1UUJGRY,1.0
336,ALBUM#76UFgbtfCFi3mGjckycfvX,1.0
336,ALBUM#6sbZYwwQB15bt5TgkPFAdb,1.0
336,ALBUM#4wQ6v4Q5YidrpC85KuBlTL,1.0
336,AUDIO#DANCEABILITY,0.5606
336,AUDIO#SPEECHINESS,0.0303
336,AUDIO#ENERGY,0.5244
336,AUDIO#ACOUSTICNESS,0.318
336,AUDIO#LOUDNESS,-9.3016
336,AUDIO#VALENCE,0.548
336,AUDIO#LIVENESS,0.1128
336,AUDIO#TEMPO,133.514
336,AUDIO#INSTRUMENTALNESS,0
```

Figure A.2: Example playlist feature entries stored for the playlist with the id *336*, containing five tracks with two tracks sharing the same artist

B. Preliminary Study Configurations

NSGA-II	N	$T_{\max}$	p_c	p_m	*Hypervolume*
	100	25000	0.85	0.005	0.6299
	100	25000	0.85	0.01	0.6336
	100	25000	0.85	0.015	0.6339
	100	25000	0.9	0.005	0.6322
	100	25000	0.9	0.01	0.6357
	100	25000	0.9	0.015	0.6316
	100	25000	0.95	0.005	**0.6387**
	100	25000	0.95	0.01	0.6375
	100	25000	0.95	0.015	0.6367
SMS-EMOA	N	$T_{\max}$	p_c	p_m	*Hypervolume*
	100	25000	0.85	0.005	0.6312
	100	25000	0.85	0.01	0.6344
	100	25000	0.85	0.015	0.6365
	100	25000	0.9	0.005	0.6379
	100	25000	0.9	0.01	0.6342
	100	25000	0.9	0.015	0.6350
	100	25000	0.95	0.005	0.6391
	100	25000	0.95	0.01	**0.6425**
	100	25000	0.95	0.015	0.6372
m-ACO$_1$	N_c	N_p	N_t	N_a	α	β	ρ	*Hypervolume*
	4	3	30	100	1.0	2.0	0.01	0.5033
	4	3	30	100	1.0	2.0	0.1	**0.5058**
	4	3	30	100	1.0	4.0	0.01	0.5036
	4	3	30	100	1.0	4.0	0.1	0.5017
m-ACO$_2$	N_c	N_p	N_t	N_a	α	β	ρ	*Hypervolume*
	4	3	10	100	1.0	2.0	0.01	0.5006
	4	3	10	100	1.0	2.0	0.1	**0.5022**
	4	3	10	100	1.0	4.0	0.01	0.4953
	4	3	10	100	1.0	4.0	0.1	0.5014
m-ACO$_3$	N_c	N_p	N_t	N_a	α	β	ρ	*Hypervolume*
	1	1	10	300	1.0	2.0	0.01	0.4418
	1	1	10	300	1.0	2.0	0.1	**0.4443**
	1	1	10	300	1.0	4.0	0.01	0.4148
	1	1	10	300	1.0	4.0	0.1	0.4117
m-ACO$_4$	N_c	N_p	N_t	N_a	α	β	ρ	*Hypervolume*
	1	3	10	500	1.0	2.0	0.01	0.4647
	1	3	10	500	1.0	2.0	0.1	**0.4819**
	1	3	10	500	1.0	4.0	0.01	0.4514
	1	3	10	500	1.0	4.0	0.1	0.4654

Table B.1: Results of the preliminary study

Bibliography

[1] Dirk Bollen, Bart P. Knijnenburg, Martijn C. Willemsen, and Mark P. Graus. Understanding choice overload in recommender systems. In *RecSys '10*, 2010. (cited on Page 1)

[2] Charles Arthur for The Guardian. *Tech giants may be huge, but nothing matches big data*. https://www.theguardian.com/technology/2013/aug/23/tech-giants-data. 2013-08-23. (cited on Page 1)

[3] Gediminas Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering*, 17:734–749, 2005. (cited on Page 1, 9, 10, and 11)

[4] X. Yang and Xingshi He. Swarm intelligence and evolutionary computation: Overview and analysis. In *Recent Advances in Swarm Intelligence and Evolutionary Computation*, 2015. (cited on Page 1, 2, and 6)

[5] T. Bäck. Evolutionary algorithms in theory and practice - evolution strategies, evolutionary programming, genetic algorithms. 1996. (cited on Page 1)

[6] J. Holland. Adaptation in natural and artificial systems. 1975. (cited on Page 1 and 6)

[7] A. Colorni, M. Dorigo, V. Maniezzo, F. Varela, and P. Bourgine. Distributed optimization by ant colonies. 1992. (cited on Page 1 and 7)

[8] J. Kennedy and R. Eberhart. Particle swarm optimization. *Proceedings of ICNN'95 - International Conference on Neural Networks*, 4:1942–1948, 1995. (cited on Page 2 and 7)

[9] D. Pham, A. Ghanbarzadeh, E. Koç, S. Otri, S. Rahim, and M. Zaidi. The bees algorithm - a novel tool for complex optimisation problems. 2006. (cited on Page 2)

[10] M. Schedl, Hamed Zamani, C. Chen, Yashar Deldjoo, and M. Elahi. Current challenges and visions in music recommender systems research. *International Journal of Multimedia Information Retrieval*, 7:95–116, 2018. (cited on Page 2, 12, and 13)

[11] A. Eiben and James E. Smith. Introduction to evolutionary computing. In *Natural Computing Series*, 2003. (cited on Page 4, 5, 6, and 8)

[12] N. Siddique and H. Adeli. Computational intelligence: Synergies of fuzzy logic, neural networks and evolutionary computing. 2013. (cited on Page 4 and 6)

[13] C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Comput. Surv.*, 35:268–308, 2003. (cited on Page 4)

[14] K. Deb, S. Agrawal, Amrit Pratap, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Trans. Evol. Comput.*, 6:182–197, 2002. (cited on Page 6, 20, 21, and 33)

[15] Kalyanmoy Deb, Samir Agrawal, Amrit Pratap, and T. Meyarivan. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimisation: Nsga-ii. In *PPSN*, 2000. (cited on Page 6 and 21)

[16] M. Dorigo and M. Birattari. Swarm intelligence. *Scholarpedia*, 2:1462, 2007. (cited on Page 6 and 7)

[17] M. Dorigo, V. Maniezzo, and A. Colorni. Ant system: optimization by a colony of cooperating agents. *IEEE transactions on systems, man, and cybernetics. Part B, Cybernetics : a publication of the IEEE Systems, Man, and Cybernetics Society*, 26 1:29–41, 1996. (cited on Page 7 and 22)

[18] M. Dorigo, M. Birattari, and T. Stützle. Ant colony optimization: artificial ants as a computational intelligence technique. *IEEE Computational Intelligence Magazine*, 1:28–39, 2006. (cited on Page 7)

[19] M. Dorigo. Ant colony optimization. *Scholarpedia*, 2(3):1461, 2007. (cited on Page 7)

[20] E. Zitzler and L. Thiele. Multiobjective optimization using evolutionary algorithms - a comparative case study. In *PPSN*, 1998. (cited on Page 8)

[21] E. Zitzler. Evolutionary algorithms for multiobjective optimization: methods and applications. 1999. (cited on Page 8)

[22] C. Aggarwal. Recommender systems: The textbook. 2016. (cited on Page 9, 10, and 11)

[23] David Goldberg, D. Nichols, B. Oki, and D. Terry. Using collaborative filtering to weave an information tapestry. *Commun. ACM*, 35:61–70, 1992. (cited on Page 9)

[24] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl. Grouplens: an open architecture for collaborative filtering of netnews. In *CSCW '94*, 1994. (cited on Page 9)

[25] D. Oard and Jinmook Kim. Implicit feedback for recommender systems. 1998. (cited on Page 10)

[26] M. Nilashi, Karamollah Bagherifard, O. Ibrahim, H. Alizadeh, L. Nojeem, and Nazanin Roozegar. Collaborative filtering recommender systems. *Research Journal of Applied Sciences, Engineering and Technology*, 5:4168–4182, 2013. (cited on Page 10)

[27] P. Aggarwal, V. Tomar, and Aditya Kathuria. Comparing content based and collaborative filtering in recommender systems. 2017. (cited on Page 11)

[28] G. Salton. Automatic text processing: The transformation, analysis, and retrieval of information by computer. 1989. (cited on Page 11)

[29] Prem Melville, R. Mooney, and R. Nagarajan. Content-boosted collaborative filtering for improved recommendations. In *AAAI/IAAI*, 2002. (cited on Page 11)

[30] R. Burke. Hybrid recommender systems: Survey and experiments. *User Modeling and User-Adapted Interaction*, 12:331–370, 2004. (cited on Page 11)

[31] M. Pazzani. A framework for collaborative, content-based and demographic filtering. *Artificial Intelligence Review*, 13:393–408, 2004. (cited on Page 11)

[32] C. Gomez-Uribe and Neil Hunt. The netflix recommender system: Algorithms, business value, and innovation. *ACM Trans. Manag. Inf. Syst.*, 6:13:1–13:19, 2016. (cited on Page 11)

[33] M. Schedl, Peter Knees, and F. Gouyon. New paths in music recommender systems research. *Proceedings of the Eleventh ACM Conference on Recommender Systems*, 2017. (cited on Page 12)

[34] Adam J. Lonsdale and A. North. Why do we listen to music? a uses and gratifications analysis. *British journal of psychology*, 102.1:108–34, 2011. (cited on Page 13)

[35] Thomas Schäfer, P. Sedlmeier, Christine Städtler, and D. Huron. The psychological functions of music listening. *Frontiers in Psychology*, 4, 2013. (cited on Page 13)

[36] G. Bonnin and D. Jannach. Automated generation of music playlists: Survey and experiments. *ACM Computing Surveys (CSUR)*, 47:1–35, 2014. (cited on Page 13)

[37] Tomás Horváth and A. Carvalho. Evolutionary computing in recommender systems: a review of recent research. *Natural Computing*, 16:441–462, 2016. (cited on Page 13)

[38] Ladislav Peska, Tsegaye Misikir Tashu, and Tomás Horváth. Swarm intelligence techniques in recommender systems - a review of recent research. *Swarm Evol. Comput.*, 48:201–219, 2019. (cited on Page 13)

[39] J. Bobadilla, F. Ortega, A. Hernando, and Javier Alcalá. Improving collaborative filtering recommender system results and performance using genetic algorithms. *Knowl. Based Syst.*, 24:1310–1316, 2011. (cited on Page 14, 16, and 41)

[40] Bushra Alhijawi and Y. Kilani. Using genetic algorithms for measuring the similarity values between users in collaborative filtering recommender systems. *2016 IEEE/ACIS 15th International Conference on Computer and Information Science (ICIS)*, pages 1–6, 2016. (cited on Page 14, 16, and 41)

[41] Bushra Alhijawi. *The Use of the Genetic Algorithms in the Recommender Systems*. PhD thesis, 2017. (cited on Page 14)

[42] Guoshuai Wei, Quanwang Wu, and Mengchu Zhou. A hybrid probabilistic multiobjective evolutionary algorithm for commercial recommendation systems. *IEEE Transactions on Computational Social Systems*, 8:589–598, 2021. (cited on Page 14 and 17)

[43] J. Mocholí, Victor Martinez, J. Martínez, and A. Catalá. A multicriteria ant colony algorithm for generating music playlists. *Expert Syst. Appl.*, 39:2270–2278, 2012. (cited on Page 14)

[44] Punam Bedi and R. Sharma. Trust based recommender system using ant colony for trust computation. *Expert Syst. Appl.*, 39:1183–1190, 2012. (cited on Page 14)

[45] Hamed Zamani, M. Schedl, P. Lamere, and C. Chen. An analysis of approaches taken in the acm recsys challenge 2018 for automatic music playlist continuation. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 10:1–21, 2019. (cited on Page 14, 18, and 39)

[46] Yehuda Koren, Robert M. Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42, 2009. (cited on Page 14)

[47] Oludare Isaac Abiodun, Aman Bin Jantan, Abiodun Esther Omolara, Kemi Victoria Dada, Nachaat Mohamed, and Humaira Arshad. State-of-the-art in artificial neural network applications: A survey. *Heliyon*, 4, 2018. (cited on Page 14)

[48] Aäron van den Oord, Sander Dieleman, and Benjamin Schrauwen. Deep content-based music recommendation. In *NIPS*, 2013. (cited on Page 14)

[49] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishikesh B. Aradhye, Glen Anderson, Gregory S. Corrado, Wei Chai, Mustafa Ispir, Rohan Anil, Zakaria Haque, Lichan Hong, Vihan Jain, Xiaobing Liu, and Hemal Shah. Wide & deep learning for recommender systems. *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*, 2016. (cited on Page 14)

[50] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016. (cited on Page 14)

[51] Jerome H. Friedman. Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29:1189–1232, 2001. (cited on Page 14)

[52] Bing Xue, Mengjie Zhang, Will Neil Browne, and Xin Yao. A survey on evolutionary computation approaches to feature selection. *IEEE Transactions on Evolutionary Computation*, 20:606–626, 2016. (cited on Page 16)

[53] Erik Bochinski, Tobias Senst, and Thomas Sikora. Hyper-parameter optimization for convolutional neural network committees based on evolutionary algorithms. *2017 IEEE International Conference on Image Processing (ICIP)*, pages 3924–3928, 2017. (cited on Page 16 and 37)

[54] C. Chen, P. Lamere, M. Schedl, and Hamed Zamani. Recsys challenge 2018: automatic music playlist continuation. *Proceedings of the 12th ACM Conference on Recommender Systems*, page 527–528, 2018. (cited on Page 18)

[55] K. Järvelin and Jaana Kekäläinen. Cumulated gain-based evaluation of ir techniques. *ACM Trans. Inf. Syst.*, 20:422–446, 2002. (cited on Page 18)

[56] Marco Tulio Ribeiro, Anísio Lacerda, Adriano Veloso, and N. Ziviani. Pareto-efficient hybridization for multi-objective recommender systems. In *RecSys '12*, 2012. (cited on Page 19 and 20)

[57] M. Kaminskas and D. Bridge. Diversity, serendipity, novelty, and coverage: A survey and empirical analysis of beyond-accuracy objectives in recommender systems. *ACM Trans. Interact. Intell. Syst.*, 7:2:1–2:42, 2016. (cited on Page 20)

[58] S. Vargas and P. Castells. Rank and relevance in novelty and diversity metrics for recommender systems. In *RecSys '11*, 2011. (cited on Page 20 and 27)

[59] Alexander Dekhtyar. Information retrieval. *Canadian Medical Association journal*, 99 14:722–3, 2018. (cited on Page 20)

[60] N. Beume, B. Naujoks, and M. Emmerich. Sms-emoa: Multiobjective selection based on dominated hypervolume. *Eur. J. Oper. Res.*, 181:1653–1669, 2007. (cited on Page 21 and 22)

[61] I. Alaya, Christine Solnon, and K. Ghédira. Ant colony optimization for multi-objective optimization problems. *19th IEEE International Conference on Tools with Artificial Intelligence(ICTAI 2007)*, 1:450–457, 2007. (cited on Page 22, 23, 24, and 34)

[62] T. Stützle and H. Hoos. Max-min ant system. *Future Gener. Comput. Syst.*, 16:889–914, 2000. (cited on Page 22)

[63] M. Buzdalov, I. Yakupov, and A. Stankevich. Fast implementation of the steady-state nsga-ii algorithm for two dimensions based on incremental non-dominated sorting. *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, 2015. (cited on Page 33)

[64] Xinye Cai, Haoran Sun, and Zhun Fan. A diversity indicator based on reference vectors for many-objective optimization. *Inf. Sci.*, 430:467–486, 2018. (cited on Page 33)

[65] D. Goldberg. Alleles, loci and the traveling salesman problem. 1985. (cited on Page 33)

[66] Vijendra Singh and Simran Choudhary. Genetic algorithm for traveling salesman problem: Using modified partially-mapped crossover operator. *2009 International Multimedia, Signal Processing and Communication Technologies*, pages 20–23, 2009. (cited on Page 33)

[67] Julia Garbaruk and D. Logofatu. Convergence behaviour of population size and mutation rate for nsga-ii in the context of the traveling thief problem. In *ICCCI*, 2020. (cited on Page 33)

[68] D. Goldberg and K. Deb. A comparative analysis of selection schemes used in genetic algorithms. In *FOGA*, 1990. (cited on Page 33)

[69] A. Lipowski and Dorota Lipowska. Roulette-wheel selection via stochastic acceptance. *ArXiv*, abs/1109.3627, 2011. (cited on Page 35)

[70] S. Ujjin and P. Bentley. Particle swarm optimization recommender system. *Proceedings of the 2003 IEEE Swarm Intelligence Symposium. SIS'03 (Cat. No.03EX706)*, pages 124–131, 2003. (cited on Page 41)

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Textpassagen, die wörtlich oder dem Sinn nach auf anderen Quellen beruhen, sind als solche kenntlich gemacht. Weiterhin wurde die Arbeit bisher weder in einem anderen Prüfungsverfahren vorgelegt noch anderweitig veröffentlicht.

Passau, 30. September 2021