PyMusiqMix

Constraints based Playlist generation using Genetic Algorithm

MAIN PROJECT REPORT

Submitted by

Nimitha Liz Sunny

Riya Mary Roly

Tom Joseph

Department of Computer Science and Engineering

Rajagiri School of Engineering and Technology

Rajagiri Valley, Kakkanad, Kochi, 682039

APRIL 2015

RAJAGIRI SCHOOL OF ENGINEERING AND TECHNOLOGY

RAJAGIRI VALLEY,KAKKANAD,KOCHI-39

CERTIFICATE

Certified that this is a Bonafide Record of the Main Project Report on ***Constraints based Playlist generation using Genetic Algorithm*** *done by* ***Nimitha Liz Sunny,Riya Mary Roly,Tom Joseph*** *with University Register Numbers* ***11012345 ,11012359 ,11012381*** *of branch* ***Computer Science and Engineering*** *during the semester Eight year 2015 at Rajagiri School of Engineering and Technology,Kakkand,Kochi.*

Project Guide *Project Coordinator* *Head of the Department*

Abstract

With a huge amount of music collections, it might be difficult for users to find favorite music. Therefore, a variety of emerging music services is proposed, such as recommendation and playlist generation. In apple iTunes or Windows Media player we see that the user has to manually add songs to the playlist. To get away from all the tedious work involved with this, lots of playlist generation techniques have come up(using variety of methods like similarity based approach, network approach, local search based approach, simulated annealing etc.).The most recent being the one using genetic algorithms.

Genetic algorithm is a very promising technique on which lots of research is being done. Genetic Algorithms are the heuristic search and optimization techniques that mimic the process of natural evolution. Genetic algorithms implement optimization strategies by simulating evolution in species through natural selection. Now in our problem, we need to generate a playlist (set of songs) according to the likes of the user. So we apply the same formula or technique to our problem. So initially we generate a random population. Each individual in a population represents a solution (in our case, a playlist).We then evaluate the fitness of each solution (playlist) by checking the number of constraints each song in the playlist satisfies. The more constraints the better solution. Now we apply the genetic operators: Selection, Crossover and Mutation. Then calculate fitness of new generation. And the process goes on.

After a few hundred or thousand generations we see that the best solution in the population will have a very high fitness value (i.e. it satisfies most of the user defined constraints).This process of GA may not sound very logical at first. But it works wonderfully and great results are generated. Given a set of constraints, we apply a genetic algorithm to generate a playlist which optimizes the number of matched constraints. We implement our prototype, and perform experiments to show the feasibility and effectiveness of prototype. So we intend to create a music player for creating a good playlist using GUI framework of Python named **PyMusiqMix**.

Acknowledgement

It is with deep sense of gratitude that we acknowledge all the helping hands that have contributed to the successful completion of main project. First of all, we are grateful to God Almighty, for having given us the skill and perseverance to complete this main project. Apart from our efforts, the success of report depends on the encouragement and guidelines of many others.

It is with deep sense of gratitude that we acknowledge all the helping hands that have contributed to the successful completion of main project. First of all, we are grateful to God Almighty, for having given us the skill and perseverance to complete this main project. Apart from our efforts, the success of report depends on the encouragement and guidelines of many others.

We express our sincere gratitude to Dr. A.Unnikrishnan, Principal, RSET and Mr. Ajith S, HOD, Department of Computer Science and Engineering, RSET for providing us with all the necessary facilities, which helped in completion of main project.

We express our gratitude to our project guide Mr. Febin P. Jacob, Assistant Professor, Department of Computer Science, RSET for his sincere and expert guidance and encouragement throughout the completion of main project

We express our gratitude to Mr. Jayarajan J N, Assistant Professor, Department of Computer Science and Engineering, for helping with the concerns and assistance.

We are extremely thankful to Mr. John Jose our project coordinator, for providing extra guidance and timely help in related matters for the successful completion of main project.

We also express our gratitude to all our friends for their help, guidance and encouragement throughout the completion of main project.

We are also thankful to all other faculties and staffs of the Department of Computer Science and Engineering for their kind help and encouragement extended to us throughout our work.

Contents

List of Figures

1 INTRODUCTION

1.1 Problem Statement

With a huge amount of music collections, it might be difficult for users to find favorite music. Therefore, a variety of emerging music services is proposed, such as recommendation and playlist generation.

In apple iTunes or Windows Media player we see that the user has to manually add songs to the playlist. Presently there are no good playlist generation software.

1.2 Project Scope / Objective

- By applying the genetic algorithm, an approach to generate playlists which satisfy constraints as much as possible in our music player for desktop application.
- The proposed solution for the music selection problem finds applications in online music distribution systems.
- Playlist creation can be aided by personalized song recommendations, ratings and reviews.
- Users having the same taste of music can share their playlists making the system more efficient.

1.3 Design and Implementation Constraints

- Result is dependent on the performance of algorithm
- It does not guarantee optimal solutions. It will provide you an acceptable solution or near optimal solution.
- The time for computation needs to be reduced

1.4 Assumptions and Dependencies

- The playlist generated cannot be as predicted
- Dependent upon the users profile
- Available internet connection for web interactions

2 LITERATURE SURVEY

2.1 Introduction To Genetic Algorithm

Genetic algorithms are useful for problems in which you know the desired result and you know how to guess at the initial state, but dont know how to get from the initial to final states. These problems often have many different parameters that can vary simultaneously which makes working through every combination of all the parameters computationally very slow and not feasible.

Genetic algorithms can evolve into a solution much like living organisms can evolve to become better fit for their environment. The basic idea is to start with a population of individuals and to allow that population to evolve to a more fit state.

We are working with a population of individuals each of whom have a fitness to exist in their current environment. This fitness is determined by the set of genes present on a single chromosome. The individuals in the population can mate and can share their genetic material through crossing-over.

Mutations can also occur at random changing any gene in any generation. When the environment changes, some individuals will be better fit to survive in the new environment. Those better-fit individuals will be more likely to mate than an individual that is poorly fit to survive in the new environment. Thus, the genes of the better-fit individual(s) are more likely to be passed on to offspring .

With each subsequent generation, the population as a whole will evolve to be more fit in the new environment because they will have a better combination of genes. There is a very nice discussion of this process, with more details, using a hypothetical organism called hooters.

Computationally, the process is very similar to the biological one. There are two critical steps that must be taken before a genetic algorithm can be run. First, one needs to define a series of parameters that might represent a solution to the problem. Second, one needs to define a fitness function that can quantify how close an individual is to being a perfect answer (perfect fitness). Once a population of individuals is generated, they are all scored for fitness.

The most-fit individuals will mate more often and some fraction of the least-fit individuals will be discarded .A mating between two individuals consists of randomly choosing a cross-over point for the two chromosomes and also performing one or more mutations (determined by a pre-determined mutation frequency).

The matings are used to generate new individuals who will replace those who died. This keeps the population at the same size in each generation. Once the population is regenerated, all of the individuals are scored again, and the process repeated. Thus, with each generation, the population becomes more fit, until an ideal solution is eventually obtained.

Genetic algorithm was used to examine different combinations of regulatory motifs and transcription factor binding sites (TFBS) with the goal of defining a set of sites that could explain the regulation of a gene set based on gene expression data. In this case, they developed a program called MotifCombinator to examine combinations of discovered motifs and known TFBS.

They seek the right combination of motifs to correlate with expression patterns of gene sets. Their chromosomes consist of a string of specific motifs or motif pairs arranged in a specific order. The fitness is related to how close the motif combination (chromosome) comes to matching gene expression patterns.

Genetic algorithms, therefore, can be used to solve problems that are currently not solvable in any way. They are a kind of hill climbing algorithm that tries to get closer to a solution with each generation. However, because they involve random choices in setting up the initial population and in making each new generation, they might not ever get to the final solution. Also, that same randomness might lead to different solutions on each run for example, different RNA structures or motif combinations.

2.2 Origins of Genetic Algorithm

The genetic algorithm (GA) is an optimization and search technique based on the principles of genetics and natural selection. A GA allows a population composed of many individuals to evolve under specified selection rules to a state that maximizes the fitness (i.e., minimizes the cost function).

The method was developed by John Holland (1975) over the course of the 1960s and 1970s and finally popularized by one of his students, David

Goldberg, who was able to solve a difficult problem involving the control of gas-pipeline transmission for his dissertation (Goldberg, 1989). Hollands original work was summarized in his book.

He was the first to try to develop a theoretical basis for GAs through his schema theorem. The work of De Jong (1975) showed the usefulness of the GA for function optimization and made the first concerted effort to find optimized GA parameters. Goldberg has probably contributed the most fuel to the GA fire with his successful applications and excellent book (1989). Since then, many versions of evolutionary programming have been tried with varying degrees of success.

Some of the advantages of a GA include that it

- Optimizes with continuous or discrete variables,
- Doesnt require derivative information,
- Simultaneously searches from a wide sampling of the cost surface,
- Deals with a large number of variables,
- Is well suited for parallel computers,
- Optimizes variables with extremely complex cost surfaces (they can jump out of a local minimum),
- Provides a list of optimum variables, not just a single solution,
- May encode the variables so that the optimization is done with the encoded variables, and

- Works with numerically generated data, experimental data, or analytical functions.

These advantages are intriguing and produce stunning results when traditional optimization approaches fail miserably. Of course, the GA is not the best way to solve every problem. For instance, the traditional methods have been tuned to quickly find the solution of a well behaved convex analytical function of only a few variables. For such cases the calculus-based methods outperform the GA, quickly finding the minimum while the GA is still analyzing the costs of the initial population.

For these problems the optimizer should use the experience of the past and employ these quick methods. However, many realistic problems do not fall into this category. In addition, for problems that are not overly difficult, other methods may find the solution faster than the GA. The large population of solutions that gives the GA its power is also its bane when it comes to speed on a serial computerthe cost function of each of those solutions must be evaluated.

2.3 PATS: Realization and User Evaluation of an Automatic Playlist Generator

A means to ease selecting preferred music referred to as Personalized Automatic Track Selection (PATS) has been developed. PATS generates playlists that suit a particular context of use, that is, the real-world environment in which the music is heard.

To create playlists, it uses a dynamic clustering method in which songs are grouped based on their attribute similarity. The similarity measure selectively weighs attribute-values, as not all attribute-values are equally important in a context-of-use. An inductive learning algorithm is used to reveal the most important attribute-values for a context-of-use from preference feedback of the user.

In a controlled user experiment, the quality of PATS compiled and randomly assembled playlists for jazz music was assessed in two contexts-of-use. The quality of the randomly assembled playlists was used as base-line. The two contexts-of-use were listening to soft music and listening to lively music.

Playlist quality was measured by precision (songs that suit the context-of-use), coverage (songs that suit the context-of-use but that were not already contained in previous playlists) and a rating score.

Results showed that PATS playlists contained increasingly more preferred music (increasingly higher precision), covered more preferred music in the collection (higher coverage), and were rated higher than randomly assembled playlists[2].So far, music player functionality that has been designed for accessing and exploiting large personal music collections aims at providing fast and accurate ways to retrieve relevant music.

This type of access generally requires well-defined targets. Music listeners need to instantaneously associate artists and song titles (or even CD and track numbers) with music.

This is not an easy task to do, since titles and artists are not necessarily learnt together with the music. In our view, selecting music from a large personal music collection is better described as a search for poorly defined targets.

These targets are poorly defined since it is reasonable to assume that music listeners have no a-priori master list of preferred songs for every listening intention, lack precise knowledge about the music, and cannot easily express their music preference on-the-fly.

Rather, choice for music requires listening to brief musical passages to recognize the music before being able to express a preference for it. If we take music programming on current music (jukebox) players as an example, it allows playing a personally created temporal sequence of songs in one go, once the playlist or program has been created. The creation of a playlist, however, can be a timeconsuming choice task. It is hard to arrive at an optimal playlist as music has personal appeal to the listener and is judged on many

subjective criteria. Also, optimality requires a complete and thorough examination of all available music in a collection, which is impractical to do so. Lastly, music programming consists of multiple serial music choices that influence each other. choice criteria pertain to individual songs as well as already selected choices.

A means to ease and speed up this music selection process could be of much help to the music listener. PATS (Personalized Automatic Track Selection) is a feature for music players that automatically creates playlists for a particular listening occasion (or context-of-use) with minimal user intervention.

This paper presents the realization of PATS and the results of a controlled user experiment to assess its performance. PATS has been realized by a decentralized and dynamic cluster algorithm that continually groups songs using an attribute-value-based similarity measure.

A song refers to a recorded performance of an artist as can be found as a track on a CD. The clustering on similarity adheres to the listeners wish of coherent music in a playlist. Since it is likely that this coherence is based on particular attribute values of the songs, some attribute values contribute more than others in the computation of the similarity by the use of weights.

At the same time, the clustering allows groups of songs to dissolve to form new groups. This concept adheres to the listeners wish of varied music within a playlist and over time. Clusters are presented to the music listener as playlists from which the listener can remove songs that do not meet the expectations of what a playlist should contain.

An inductive learning algorithm based on decision trees is then employed that tries to reveal the attribute values that might explain the removal of songs. Weights of attribute values are adjusted accordingly, and the clustering continues with these new weights aiming at providing better future playlists.

2.4 New Recommendation Techniques for Multi-Criteria Rating Systems

While traditional single-rating recommender systems have been successful in a number of personalization applications, the research area of multi-criteria recommender systems has been largely untouched. In order to take full advantage of multi-criteria ratings in various applications, new recommendation techniques are required.

In this paper we propose two new approaches the similarity-based approach and the aggregation function-based approach to incorporating and leveraging multi-criteria rating information in recommender systems. We also discuss multiple variations of each proposed approach, and perform empirical analysis of these approaches using a real-world dataset. Our experimental results show that multi-criteria ratings can be successfully leveraged to improve recommendation accuracy, as compared to traditional single-rating recommendation techniques.

2.5 Evaluating Collaborative Filtering Recommender Systems

Recommender systems have been evaluated in many, often incomparable, ways. In this article, we review the key decisions in evaluating collaborative filtering recommender systems: the user tasks being evaluated, the types of analysis and datasets being used, the ways in which prediction quality is measured, the evaluation of prediction attributes other than quality, and the user-based evaluation of the system as a whole.

In addition to reviewing the evaluation strategies used by prior researchers, we present empirical results from the analysis of various accuracy metrics on one content domain where all the tested metrics collapsed roughly into three equivalence classes. Metrics within each equivalency class were strongly correlated, while metrics from different equivalency classes were uncorrelated

2.6 A Comparison Of Signal-Based Music Recommendation To Genre Labels, Collaborative Filtering, Musicological Analysis, Human Recommendation, And Random Baseline

The emergence of the Internet as todays primary medium of music distribution has brought about demands for fast and reliable ways to organize, access, and discover music online. To date, many applications designed to perform such tasks have risen to popularity.

Each relies on a specific form of music metadata to help consumers discover songs and artists that appeal to their tastes. Very few of these applications, however, analyze the signal waveforms of songs directly. This low-level representation can provide dimensions of information that are inaccessible by metadata alone.

To address this issue, we have implemented signal based measures of musical similarity that have been optimized based on their correlations with human judgments. Furthermore, multiple recommendation engines relying on these measures have been implemented. These systems recommend songs to volunteers based on other songs they find appealing.

Blind experiments have been conducted in which volunteers rate the systems recommendations along with recommendations of leading online music discovery tools (Allmusic which uses genre labels, Pandora which uses musicological analysis, and Last.fm which uses collaborative filtering), random baseline recommendations, and personal recommendations by the first author. This paper shows that the signal-based engines perform about as well as popular, commercial, state-of-the-art systems.

2.7 Playlist Generation Using Start And End Songs

A new algorithm for automatic generation of playlists with an inherent sequential order is presented. Based on a start and end song it creates a smooth transition allowing users to discover new songs in a music collection. The approach is based on audio similarity and does not require any kind of meta data. It is evaluated using both objective genre labels and subjective listening tests.

This work is concerned with the creation of playlists with an inherent sequential order. Such a playlist consists of a start and an end song, both chosen by a user. The songs in between should form a smooth transition, with songs at the beginning sounding similar to the start song, songs at the end similar to the end song and songs in the middle similar to both start and end songs. Our approach is based solely on audio analysis and does not require any kind of metadata.

It could therefore easily replace or at least support manual creation of playlists. It allows to explore audio collections by simply choosing two songs and a desired length for the playlist. It also enables efficient discovery of new music if applied to collections of yet unknown songs by automatically creating a smooth transition between only two supporting songs.

Most existing approaches to playlist generation rely on the usage of one seed song or a group of seed songs. The playlist then consists of songs which are somehow similar to this seed. Some authors use different kinds of audio similarity to create the playlists.

Others work with some kind of metadata. Seed based creation of playlists has the problem of producing too uniform lists of songs if applied to large data bases with lots of similar music. If a data base does not contain enough similar music to a seed song there is the danger of playlist drift towards music that sounds very different.

Few authors report about generating playlists with an inherent sequential order. Most approaches are solely based on metadata and not audio similarity. Case Based Reasoning has been applied to create new playlists with inherent temporal structure based on patterns of subsequences of a collection of existing playlists.

Creation of playlists satisfying user constraints based on rich metadata has also been reported. These constraints may also concern the temporal order of the playlists (e.g. rising tempo, change of genre). This constraint based approach has been extended to include notions of audio similarity as yet another constraint (e.g. timbre continuity through a playlist).

Related approaches have been formulated based on simulated annealing and linear programming. Travelling Salesman algorithms applied to audio similarity have been used to generate a sequential order of all songs in a data base. Since all songs have to be part of the playlist, this is a quite different kind of organising principle for the playlist.

Direct interaction with a two dimensional mapping of music spaces based on audio similarity also allows creation of playlists with inherent sequential structure. Computations are based on the lower dimensional representations and not directly on the audio models of the songs themselves. A related approach also using a two dimensional display which is enriched with various kinds of meta data has also been presented.

2.8 Related Models

There are two kinds of approaches for generating playlists recommendation systems and constraint based approach.

2.8.1 Recommendation Systems

Recommended systems or recommendation systems are a subclass of information filtering system that seek to predict the 'rating' or preference' that user would give to an item (such as music, books, or movies) or social element (e.g. people or groups) they had not yet considered, using a model built from the characteristics of an item (content-based approaches), the user's social environment (collaborative filtering approaches) or hybrid systems. Recommender systems have become extremely common in recent years .Examples of such systems:

- When viewing a product on Amazon.com, the store will recommend additional items based on a matrix of what other shoppers bought along with the currently selected item.
- Netflix offers predictions of movies that a user might like to watch based on the user's previous ratings, watching habits, and characteristics such as the genre of the film.

- Collaborative Filtering

  Collaborative filtering methods are based on collecting and analyzing a large amount of information on users behaviors, activities or preferences and predicting what users will like based on their similarity to other users. A key advantage of the collaborative filtering approach is that it does not rely on machine analysable content and therefore it is capable of accurately recommending complex items such as movies without requiring an understanding" of the item itself.

  The recommender system compares the collected data to similar and dissimilar data collected from others and calculates a list of recommended items for the user. Collaborative filtering uses the ratings and the user preferences to make recommendations.

  This system tries to use such information to find other users who have a similar taste, and recommends those songs from the same user group.

The basic idea behind is that users with similar behavior manner shares their preference. For example, user A and user B belong to the same group. User A likes object 1 and object 2, while user B likes object 1 and object 3. Therefore, object 3 will be recommended to user A.

- Content-based Filtering

  Content-based filtering methods are based on information about and characteristics of the items that are going to be recommended. In other words, these algorithms try to recommend items that are similar to those that a user liked in the past or is examining in the present.

  In particular, various candidate items are compared with items previously rated by the user and the best-matching items are recommended. This approach has its roots in information retrieval and information filtering research.

2.8.2 Hybrid Systems

Recent research has demonstrated that a hybrid approach, combining collaborative filtering and content-based filtering could be more effective in some cases. Hybrid approaches can be implemented in several ways: by making content-based and collaborative based predictions separately and then combining them, by adding content-based capabilities to a collaborative-based approach and vice versa, or by unifying the approaches into one model.

2.8.3 Constraint-based Approach

In constraint based approach, constraints can be specified by users or derived by system. A playlist will be constructed to satisfy constraints as much as possible. The different constraint based approaches are:

- Network flow Approach

  In this system, a song is represented as a node and constraints as edges with cost and weight. The target of this system is to find a continuous path connecting the source and the sink (i.e., the first and the last song in the playlist) associated with a minimum cost.

  This system consists of two constraints: absolute and coherence constraints. Absolute constraints are the maximum and the minimum percentage of each attribute; they are represented by weights associated with edges in the network flow model.

  The coherence constraints enforce certain correlations between successive songs in the sequence such that they are similar. The coherence constraints are represented by costs associated with edges in the network flow model. After setting up the corresponding network model, the problem of finding path will be transformed into an integer linear program and solved by the technique of branch and bound. However, in the worst case, the time complexity would be in exponential time.

- Similarity based Approach

  Similarity based approach is another method for solving playlist generation problem. It is a special case of the constraint-based approach.

Here timbral similarities are used as the distances, and a playlist is created such that the

Consecutive songs are maximally similar by applying the Traveling Salesman Algorithm.A technique of combining audio signal-based music similarity and web-based musical artist similarity is an enhancement to the method to accelerate the task of automatic playlist generation.

2.9 Advantages Of Genetic Algorithm

The advantages are:-

- The playlist generation problem can easily be represented as chromosomal encoding.
- As the problem has multiple solutions, genetic algorithm solves it better than other conventional methods.
- Genetic algorithm is very easy to understand and it practically does not demand the knowledge of mathematics.
- Genetic algorithms can be easily transferred to existing simulations and modules.
- Genetic algorithms search parallel from a population of points. Therefore, it has the ability to avoid being trapped in local optimal solution like traditional methods, which search from a single point.
- Genetic algorithms use probabilistic selection rules, not deterministic ones.

- Genetic algorithms work on the Chromosome, which is encoded version of potential solutions parameters, rather the parameters themselves.
- Genetic algorithms use fitness score, which is obtained from objective functions, without other derivative or auxiliary information

2.10 Limitations of Genetic Algorithm

There are limitations of the use of a genetic algorithm compared to alternative optimization algorithms:

- Certain optimization problems (they are called variant problems) cannot be solved by means of genetic algorithms. This occurs due to poorly known fitness functions which generate bad chromosome blocks in spite of the fact that only good chromosome blocks cross-over.
- There is no absolute assurance that a genetic algorithm will find a global optimum. It happens very often when the populations have a lot of subjects.
- Like other artificial intelligence techniques, the genetic algorithm cannot assure constant optimization response times. Even more, the difference between the shortest and the longest optimization response time is much larger than with conventional gradient methods. This unfortunate genetic algorithm property limits the genetic algorithms' use in real time applications.
- Genetic algorithm applications in controls which are performed in real

time are limited because of random solutions and convergence, in other words this means that the entire population is improving, but this could not be said for an individual within this population. Therefore, it is unreasonable to use genetic algorithms for on-line controls in real systems without testing them first on a simulation model.

3 HARDWARE AND SOFTWARE

This section gives the hardware and software specification.

3.1 Hardware Requirements

- RAM : 1 GB
- Processor : Dual core Processor
- Hard Disk : 80 GB

3.2 Software Requirements

- Platform : Windows
- IDE : Pycharm

  PyCharm is an Integrated Development Environment (IDE) used for programming in Python. It provides code analysis, a graphical debugger, an integrated unit tester, integration with version control systems (VCSes), and supports web development with Django. PyCharm is developed by the Czech company JetBrains.

  It is cross-platform working on Windows, Mac OS X and Linux. PyCharm has a Professional Edition, released under a proprietary license and a Community Edition released under the Apache License. PyCharm Community Edition is less extensive than the Professional Edition.

- Front end : PyQt

  PyQt is a Python binding of the cross-platform GUI toolkit Qt. It is one of Python's options for GUI programming.Like Qt, PyQt is free software. PyQt is implemented as a Python plug-in. PyQt is developed by the British firm Riverbank Computing.

  It is available under similar terms to Qt versions older than 4.5; this means a variety of licenses including GNU General Public License (GPL) and commercial license, but not the GNU Lesser General Public License (LGPL). PyQt supports Microsoft Windows as well as various flavours of Unix, including Linux and OS X.

- Back end : Python

  Python is a widely used general-purpose, high-level programming language. Its design philosophy emphasizes code readability, and its syntax allows programmers to express concepts in fewer lines of code than would be possible in languages such as C++ or Java. The language provides constructs intended to enable clear programs on both a small and large scale.

  Python supports multiple programming paradigms, including object-oriented, imperative and functional programming or procedural styles. It features a dynamic type system and automatic memory management and has a large and comprehensive standard library.

  Python interpreters are available for installation on many operating systems, allowing Python code execution on a wide variety of systems.Using third-party tools, such as Py2exe or Pyinstaller, Python code can be packaged into stand-alone executable programs for some of the most popular operating systems, allowing for the distribution of Python-based software for use on those environments without requiring the installation of a Python interpreter.

4 SYSTEM ANALYSIS AND DESIGN

4.1 Existing system

Large music collections afford the listener flexibility in the form of choice, which enables the listener to choose the appropriate piece of music to enhance or complement their listening scenario on demand. However, bundled with such a large music collection is the daunting task of manually searching through each entry in the collection to find the appropriate song required by the listener.

This often leaves the listener frustrated when trying to select songs from a large music collection.With a huge amount of music collections, users may be difficult to find favorite music. Therefore, a variety of emerging music services are proposed, such as recommendation and playlist generation. For example, using Windows Media Player and iTunes, we can make custom playlists by manually adding songs. However, it is a very tedious work for users to maintain those playlists in such a manner.

So,inorder to solve this problem there is a need to define a formal model of playlist generation awith the help of constraints s, including parameter specified constraints and user-defined constraints. By applying the genetic algorithm, we provide an approach to generate playlists which satisfy constraints as much as possible.

4.2 Requirement Analysis

Requirements analysis in systems engineering and software engineering, encompasses those tasks that go into determining the needs or conditions to meet for a new or altered product or project, taking account of the possibly conflicting requirements of the various stakeholders, analyzing, documenting, validating and managing software or system requirements.

Requirements analysis is critical to the success of a systems or software project.The requirements should be documented, actionable, measurable, testable, traceable, related to identified business needs or opportunities, and defined to a level of detail sufficient for system design.

Requirements considering the project are

- System Requirements like Windows,Speakers,Python and mp3 songs.
- Constraint module for handling the various constraints.
- id3reader for reading metadata from the songs.
- Phonon for playing songs.
- PyQt for developing the Graphical User Interface(GUI).

4.3 Proposed System

In this model, the playlist generation problem is considered as a combinatorial problem.Genetic algorithm is used to solve the playlist generation problem.

4.3.1 Architectural and Structural Design

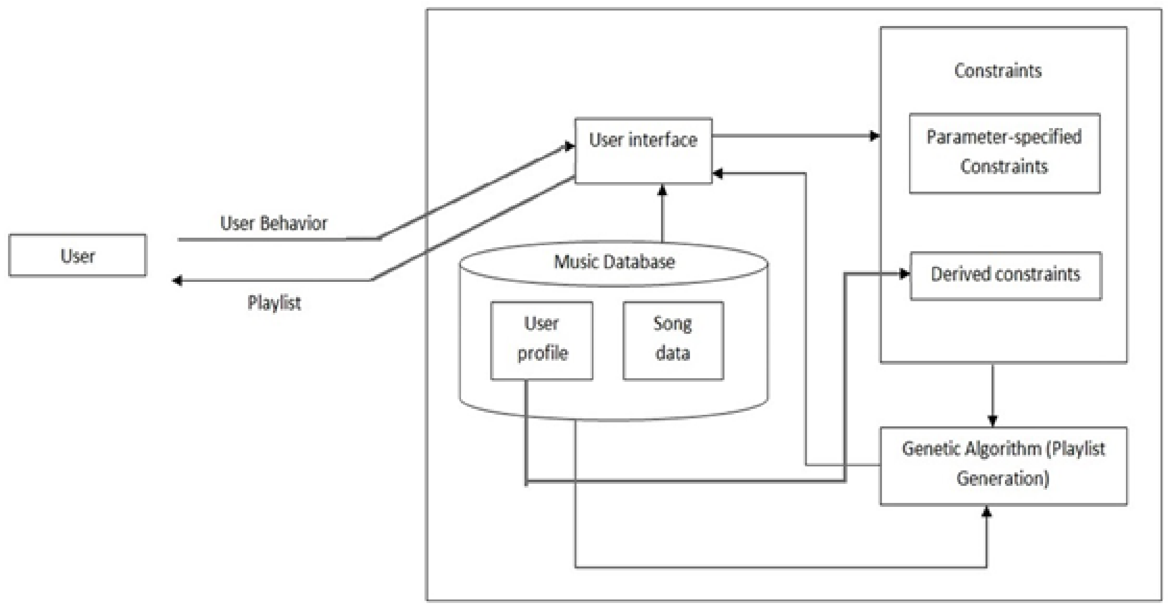


Figure 1: System Architecture

- Music database: The music database stores songs and user profiles.
- User interface module: It is the coordinator of the system, and collects user behaviour, dispatches user commands and organizes returned results.

- Playlist generation module: It is the core algorithm. By applying genetic algorithm, playlists will be generated to meet constraints as much as possible.

- Constraints module: User constraints will be maintained in this module. Three types of constraints are used: derived constraints, parameter-specified constraints, and user-defined constraints.

- Derived constraints: The derived constraints will be automatically learned and derived through the user behaviour, the user profile, and/or the user interaction.

- Parameter-specified constraints: A scheme for experienced users to specify their constraints. In the system, Users express the parameter-specified constraints by filling the attributes.

A. Derived constraints If naive users are not able to specify constraints,system will construct the corresponding constraints based on user behavior.

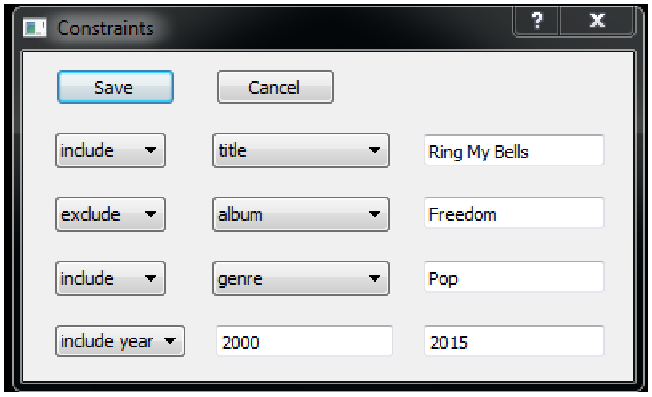
- GCo(1,—P—, 10, sad, 1,—P—)

  At least one sad mood song in the listened list.

- UI(6, 11,sad)

  The mood of the sixth song should be sad.

- BS(7, 5, 13)


Constraints

Save | Cancel

include	title	Ring My Bells
exclude	album	Freedom
include	genre	Pop
include year	2000	2015

Figure 2: Constraints

The release year of the seventh song should be earlier than the fifth one.

For each candidate constraint, say a constraint CA, CA will be evaluated against the listened list. If the number of matched cases is more than a predefined threshold, CA will be considered as a result of derived constraint.

B. Parameter-specified constraints We provide a scheme for experienced users to specify their constraints. In our system, we define Unary Operations to express the parameter-specified constraints. In our implementation, we design a form action of window application in which users are able to manipulate operators and assign parameters to describe constraints.

C. Algorithm for playlist generation First, the genetic representation of the solution domain along with the fitness functionand stop criterion or the termination condition.

Representation: In our system, each individual from population is a candidate playlist, and the corresponding chromosome of individual is a sequence of songIDs in the candidate playlist. When applying GA, the chromosome is usually coded as a bit-string. Therefore, we transform a songID into a bitstring for further operations. As a result, we use n x lg —M— bits to encode an individual, where n is the length of playlist, and —M— is the size of database M.

Termination condition: Two stop criterias are present. The first one is the maximum run of generations, in case of not reaching convergence. The second one is that all of individuals satisfy the fitness threshold.

4.4 Module Division

This section gives brief description about the module division in the project.

Module 1-TOM JOSEPH

- Algorithm Combining With Gui - Combining the gui features to the algorithm to produce a unified product
- GUI Integration With Web To use the web for web scrapping the ratings of the song

Module 2-RIYA MARY ROLY

- Algorithm-design Of Functions and Data Structure The core of algorithm are the three functions selection, crossover and mutation.

Module 3-NIMITHA LIZ SUNNY

- Gui Layout design- Basic Layout can be created Using PyQt and adopting widgets in its framework.

4.5 DFDs / UML Diagrams

4.5.1 Data Flow Diagram

A data flow diagram (DFD) is a graphical representation of the ”flow” of data through an information system, modelling its process aspects. A DFD is often used as a preliminary step to create an overview of the system, which can later be elaborated. DFDs can also be used for the visualization of data processing (structured design).

A DFD shows what kind of information will be input to and output from the system,where the data will come from and go to, and where the data will be stored. It does not show information about the timing of processes, or information about whether processes will operate in sequence or in parallel (which is shown on a flowchart).

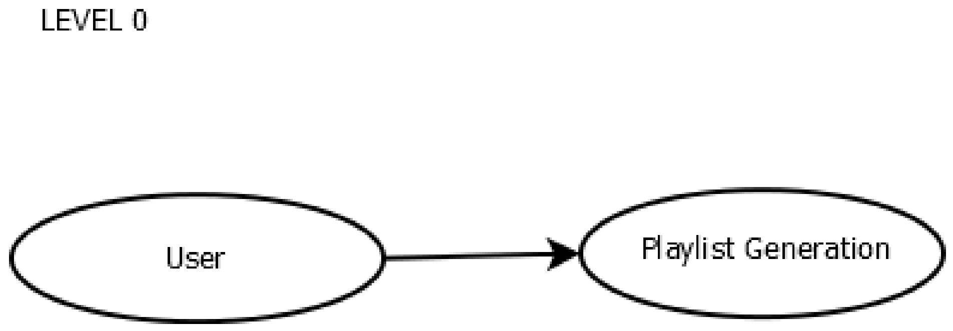


Figure 3: Data Flow Diagram level 0

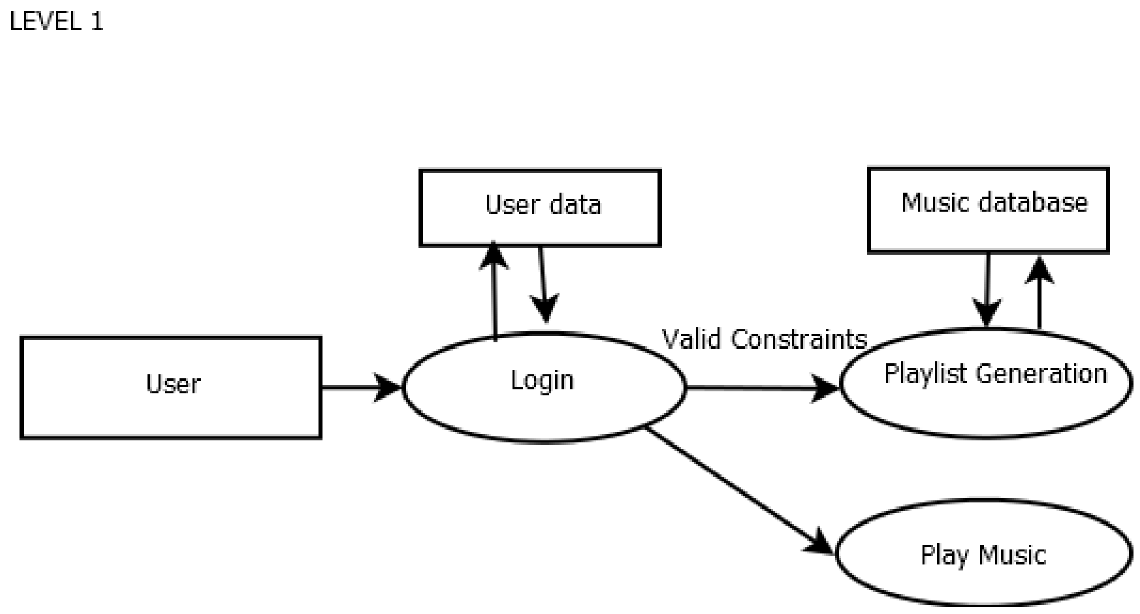


Figure 4: Data Flow Diagram level 1

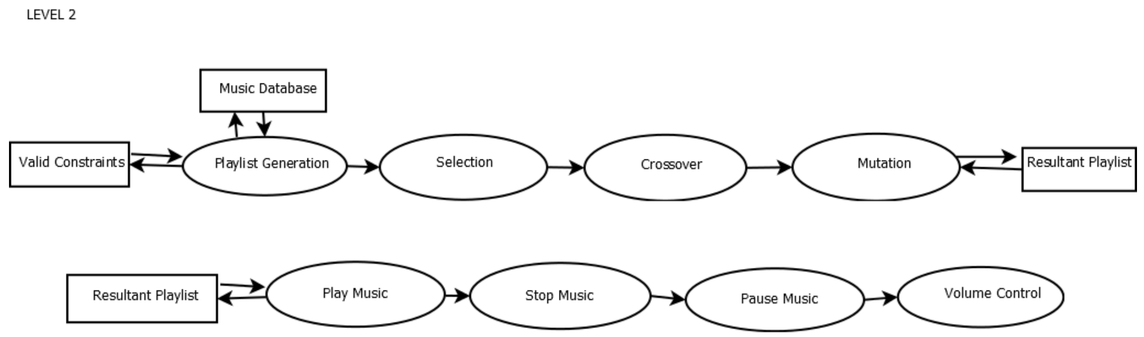


Figure 5: Data Flow Diagram level 2

4.5.2 Use Case Diagram

A use case diagram at its simplest is a representation of a user's interaction with the system that shows the relationship between the user and the different use cases in which the user is involved. A use case diagram can identify the different types of users of a system and the different use cases and will often be accompanied by other types of diagrams as well.

- Use Case Description

  The user inputs the username and password along with the constraints or parameters for generating the playlist.

- Flow of events

  The inputs along with the constraints are given to genetic algorithm which finds an optimal solution and gives to GUI for displaying the playlist.

- Pre-conditions

  Inputs for playlist i.e. parametric constraints

- Post-conditions

  The user will be recommended a playlist.

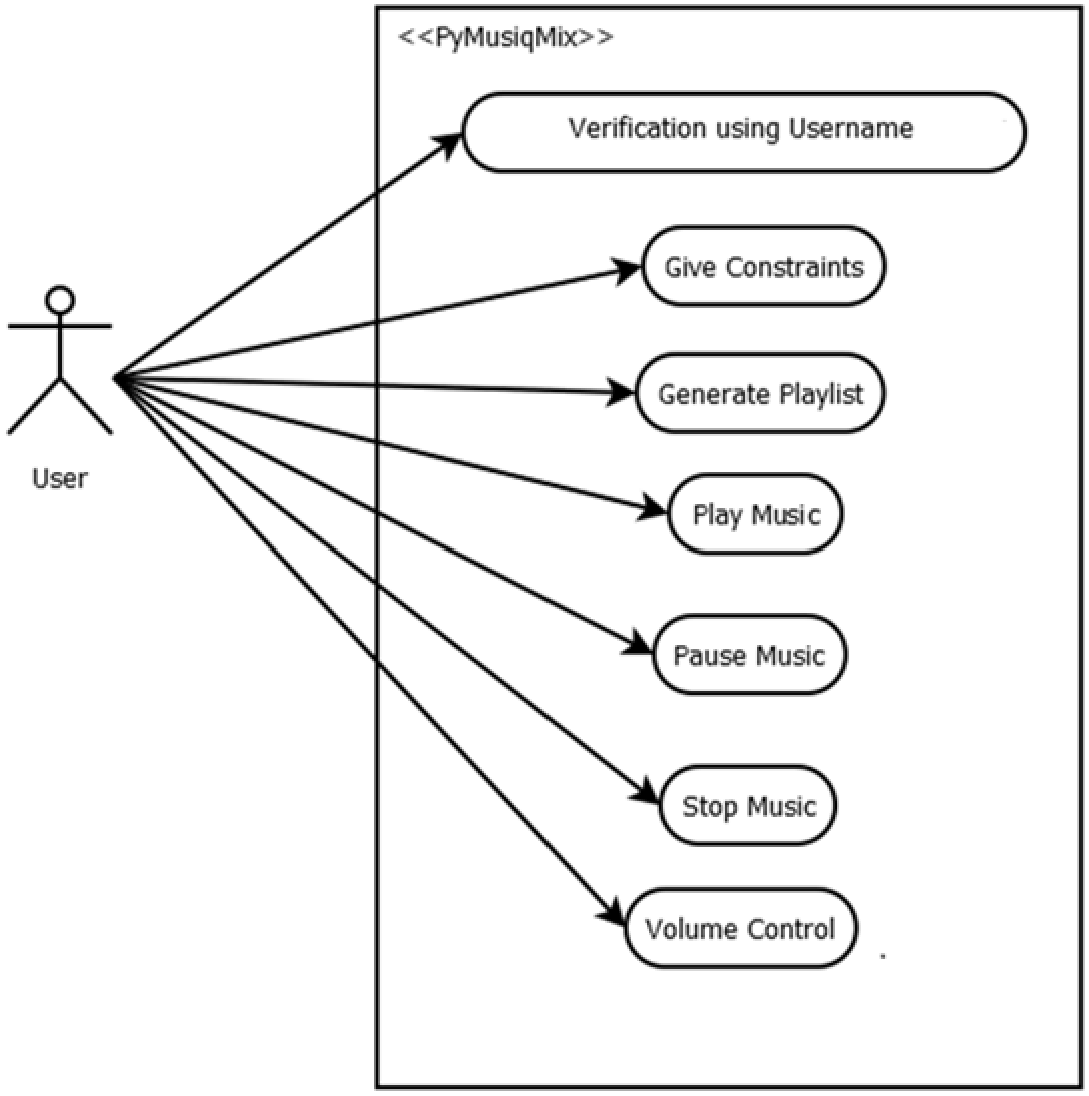


Figure 6: Use Case Diagram

4.5.3 Activity Diagram

Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency. Activity diagrams show the overall flow of control. Activity diagrams are constructed from a limited number of shapes, connected with arrows. The most important shape types:

- rounded rectangles represent actions;
- diamonds represent decisions;
- bars represent the start (split) or end (join) of concurrent activities;
- a black circle represents the start (initial state) of the workflow;
- An encircled black circle represents the end (final state).

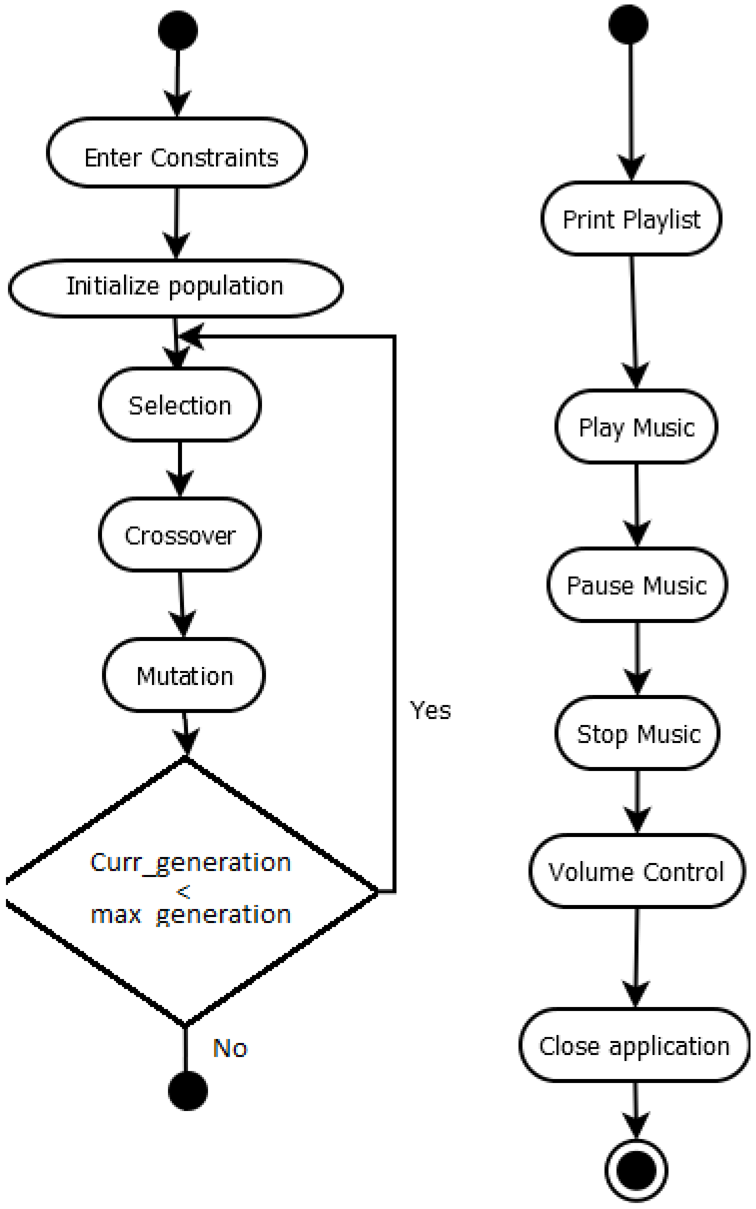


Figure 7: Activity Diagram

4.5.4 Sequence Diagram

A Sequence diagram is an interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message Sequence Chart. A sequence diagram shows object interactions arranged in time sequence. It depicts the objects and classes involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario.

A sequence diagram shows, as parallel vertical lines (lifelines), different processes or objects that live simultaneously, and, as horizontal arrows, the messages exchanged between them, in the order in which they occur. This allows the specification of simple runtime scenarios in a graphical manner.

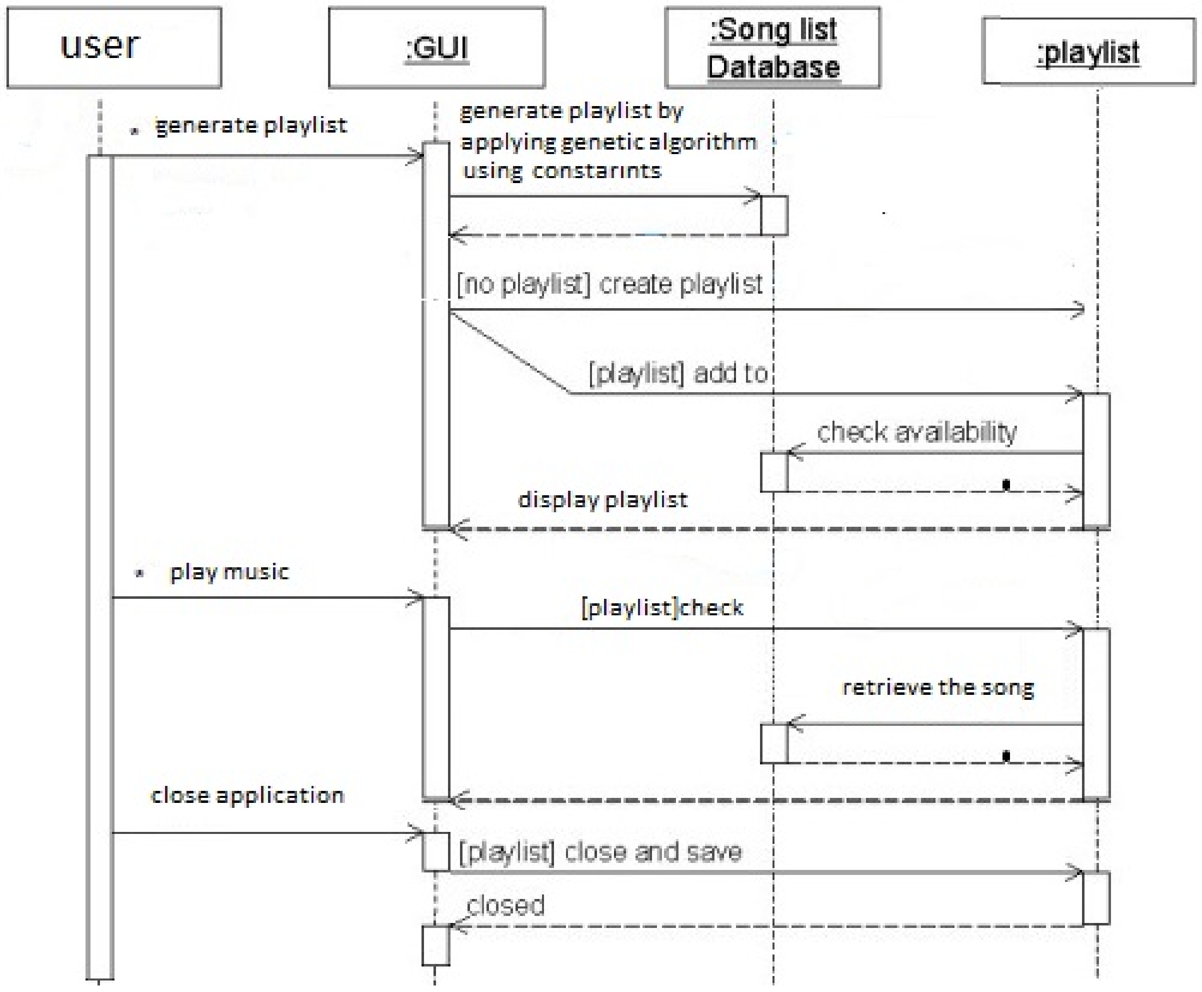


Figure 8: Sequence Diagram

4.6 Pseudocode/Algorithm

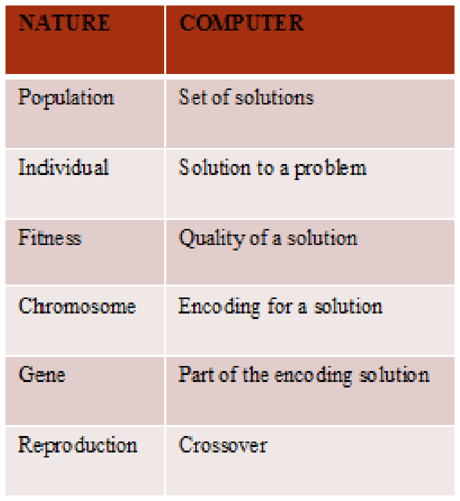
The algorithm used for the generation of playlist is Genetic algorithm. Genetic Algorithms are global search and optimization techniques modeled from natural selection, genetic and evolution. It mimics natural evolution process for producing better results. A typical genetic algorithm requires:

1. A genetic representation of the solution domain (chromosomes)

2. A fitness function to evaluate the solution domain.

New off springs are generated /evolved from the chromosomes using operators like selection, crossover and mutation.A playlist is defined as a finite sequence of songs. Each song is represented by a vector of attributes. The types of attributes can be textual, numerical and categorical.

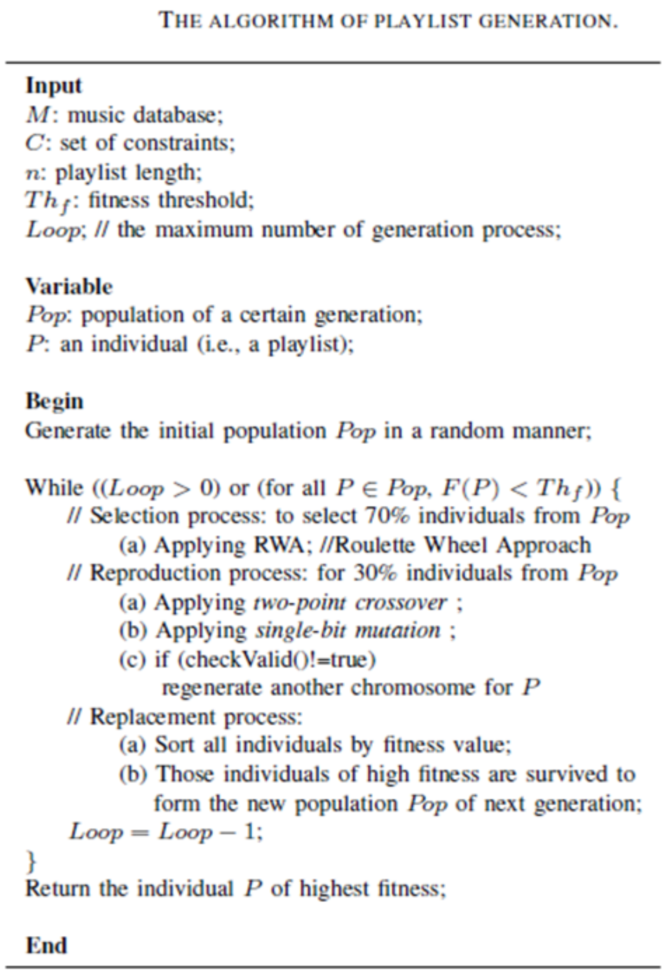
Nature to Computer Mapping



NATURE	COMPUTER
Population	Set of solutions
Individual	Solution to a problem
Fitness	Quality of a solution
Chromosome	Encoding for a solution
Gene	Part of the encoding solution
Reproduction	Crossover

Figure 9: Nature to Computer Mapping

The following is the genetic algorithm used for the generation of playlist.



THE ALGORITHM OF PLAYLIST GENERATION.

```
Input
$M$: music database;
$C$: set of constraints;
$n$: playlist length;
$Th_f$: fitness threshold;
$Loop$; // the maximum number of generation process;

Variable
$Pop$: population of a certain generation;
$P$: an individual (i.e., a playlist);

Begin
Generate the initial population $Pop$ in a random manner;

While (($Loop > 0$) or (for all $P \in Pop$, $F(P) < Th_f$)) {
    // Selection process: to select 70% individuals from $Pop$
        (a) Applying RWA; //Roulette Wheel Approach
    // Reproduction process: for 30% individuals from $Pop$
        (a) Applying *two-point crossover* ;
        (b) Applying *single-bit mutation* ;
        (c) if (checkValid()!=true)
              regenerate another chromosome for $P$
    // Replacement process:
        (a) Sort all individuals by fitness value;
        (b) Those individuals of high fitness are survived to
            form the new population $Pop$ of next generation;
    $Loop = Loop - 1$;
}
Return the individual $P$ of highest fitness;

End
```

Figure 10: Algorithm for Playlist Generation

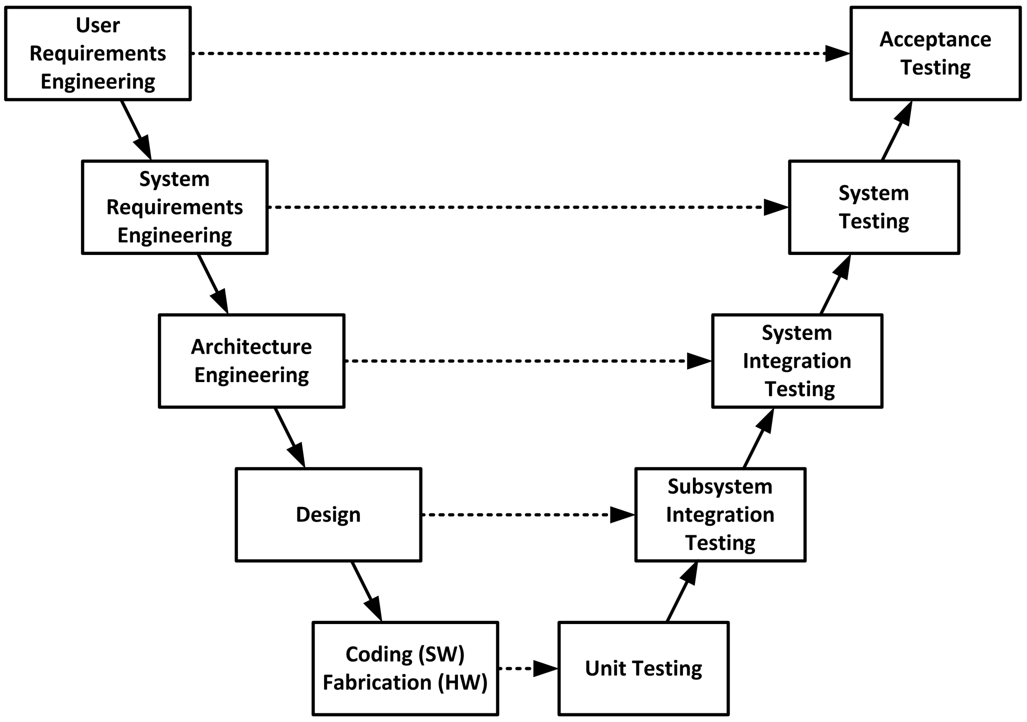
5 IMPLEMENTATION

The various modules was recognized for the project and different module was created.These module was allotted among the members.The genetic algorithm was broken down into module functions like selection,crossover,mutation and validation.The coding for genetic algorithm was done.Genetic algorithm was tested.The various needs for GUI was identified and user Interface was designed.The coding for GUI application was done.GUI was tested by given test cases.The algorithm and GUI was integrated.The system as a whole was tested and bugs was corrected.

6 TESTING STRATEGIES

Testing is the penultimate step of software development. An elaborate testing of data is prepared and the system is using test data. While doing testing, errors are noted and correction is made. The users are trained to operate the developed system .

Both hardware and software securities are made to run the developed system successfully.System testing is aimed at ensuring the system works accurately before live operation commences. Testing is vital to the system.



User Requirements Engineering
Acceptance Testing
System Requirements Engineering
System Testing
Architecture Engineering
System Integration Testing
Design
Subsystem Integration Testing
Coding (SW) Fabrication (HW)
Unit Testing

Figure 11: Testing Life cycle

System testing makes a logical assumption that if all parts of the system are correct, the goal will be successfully achieved. The candidate system is subjected to a verity of tests Online Response, Volume, Stress Recovery & Security and Usable tests. A series of testing are performed for the proposed system before the system is ready for user acceptance testing. Nothing is complete without testing, as it is vital success of the system.

The entire testing process can be divided into 3 phases:-

- Unit Testing

  In computer programming, unit testing is a software testing method by which individual units of source code, sets of one or more computer program modules together with associated control data, usage procedures, and operating procedures, are tested to determine whether they are fit for use.

  Intuitively, one can view a unit as the smallest testable part of an application.In procedural programming, a unit could be an entire module, but it is more commonly an individual function or procedure. In object-oriented programming, a unit is often an entire interface, such as a class, but could be an individual method.

  Unit tests are short code fragments created by programmers or occasionally by white box testers during the development process. It is also known as component testing.

- Integration Testing

  Integration testing (sometimes called integration and testing, abbreviated I&T) is the phase in software testing in which individual software modules are combined and tested as a group. It occurs after unit testing and before validation testing.

Integration testing takes as its input modules that have been unit tested, groups them in larger aggregates, applies tests defined in an integration test plan to those aggregates, and delivers as its output the integrated system ready for system testing

- Final System testing

  System testing of software or hardware is testing conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements. System testing falls within the scope of black box testing, and as such, should require no knowledge of the inner design of the code or logic.

  As a rule, system testing takes, as its input, all of the "integrated" software components that have passed integration testing and also the software system itself integrated with any applicable hardware system(s). The purpose of integration testing is to detect any inconsistencies between the software units that are integrated together (called assemblages) or between any of the assemblages and the hardware. System testing is a more limited type of testing; it seeks to detect defects both within the "inter-assemblages" and also within the system as a whole.

Case no	Featured tested	Test procedure
1.	Main module	The user selects and runs the program.
2.	Displaying options	Various options are enlisted.
3.	Login module	The user is allowed to login to the application by entering the username
4.	User module	The software is customized for a particular user.
5.	Add more files	Different song files can be added by the user.These serve as seed songs
6.	Constraints	The user is able to select the required constraints and add constraints.
7.	Generation of playlist	The playlist is generated according to the constraints entered.

7 RESULTS

The results of the work are given as screenshots. The screenshots in this section depict the various test cases.

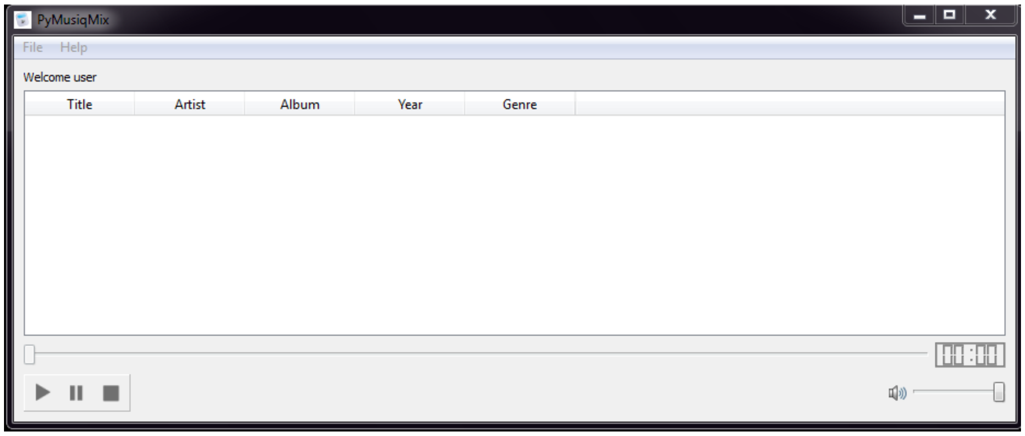


Figure 12: Main module

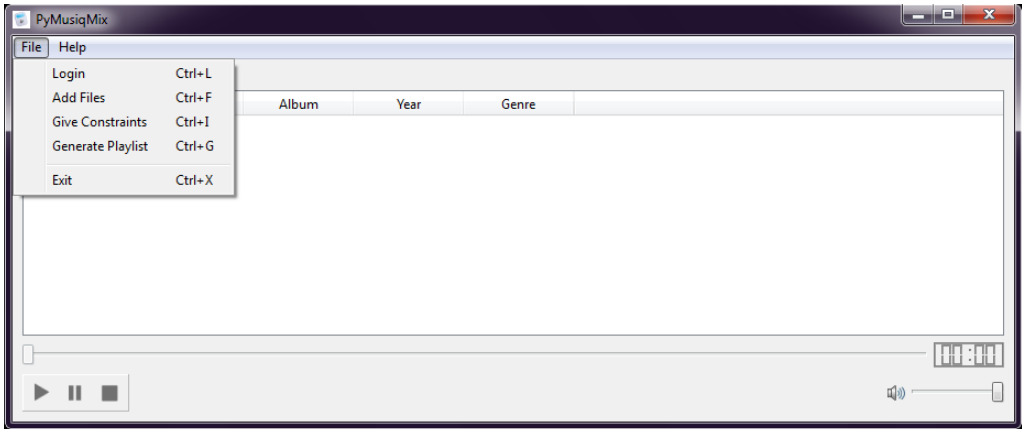


Figure 13: Options

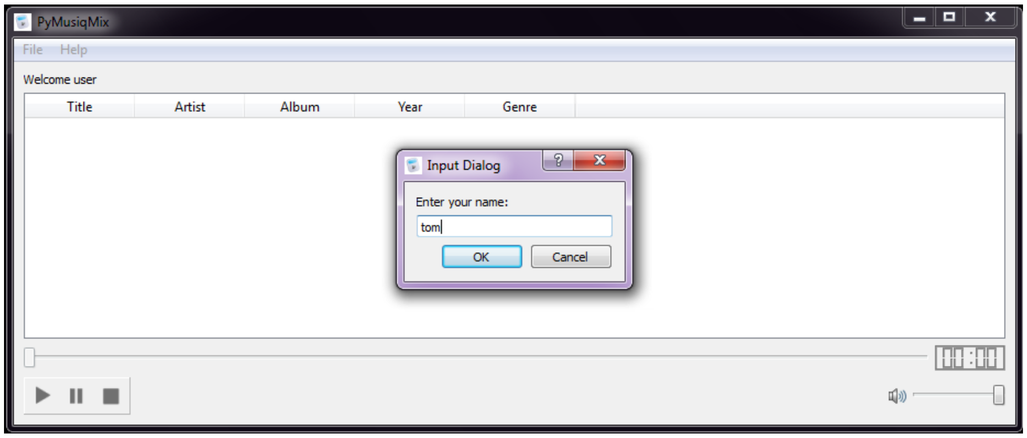


Figure 14: Login module

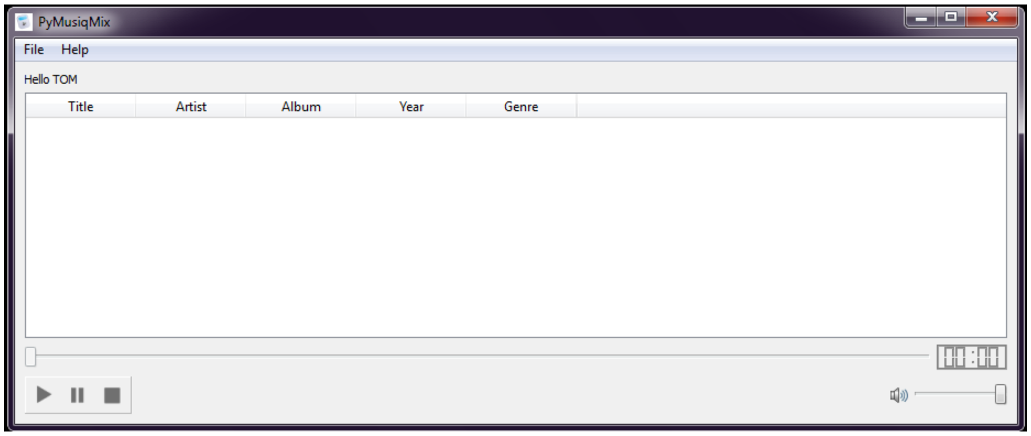


Figure 15: User module

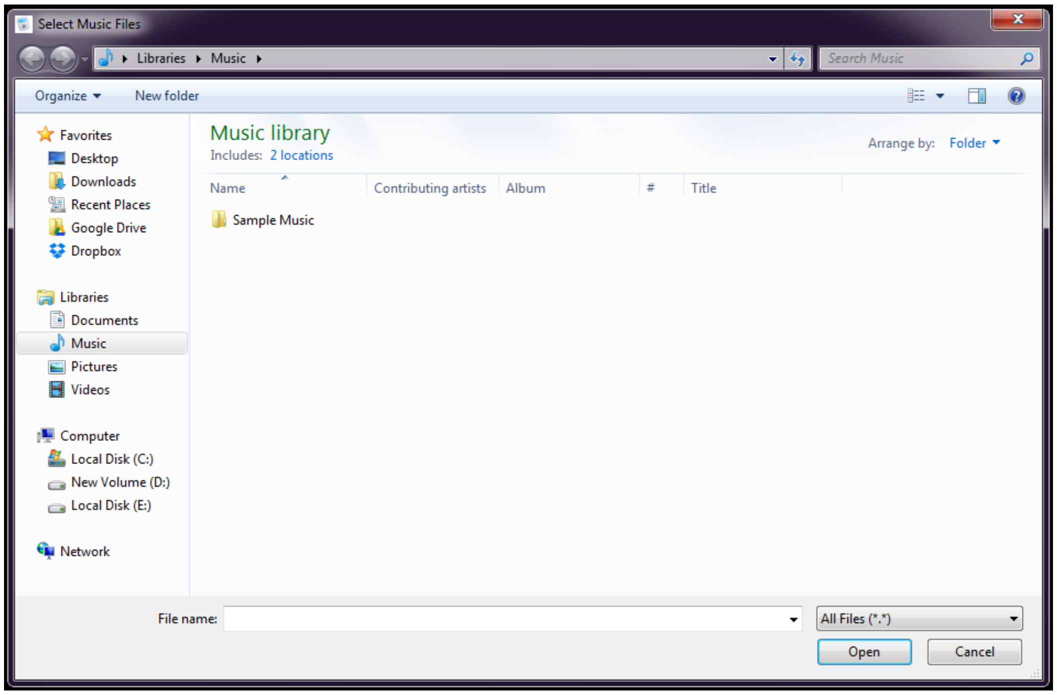
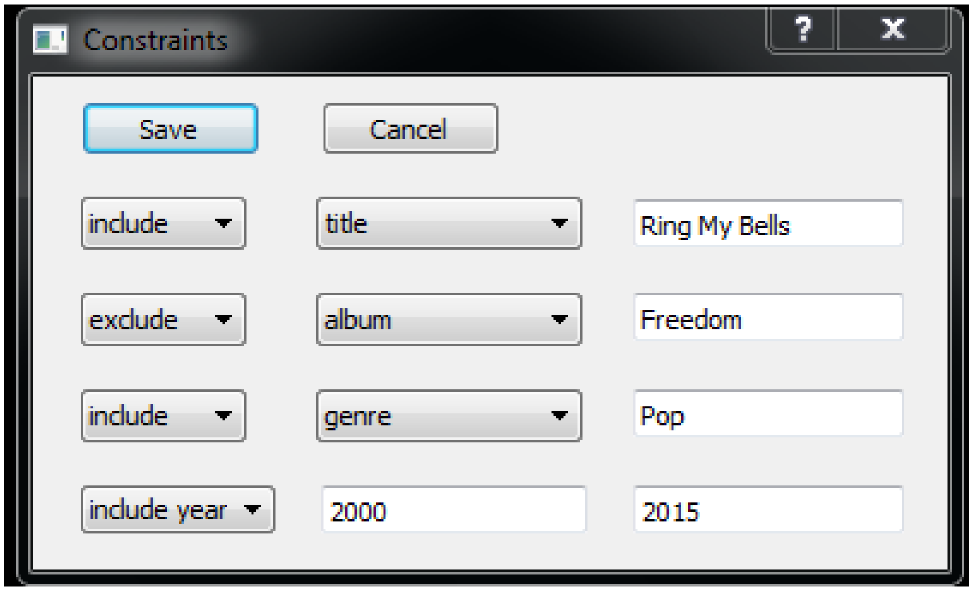
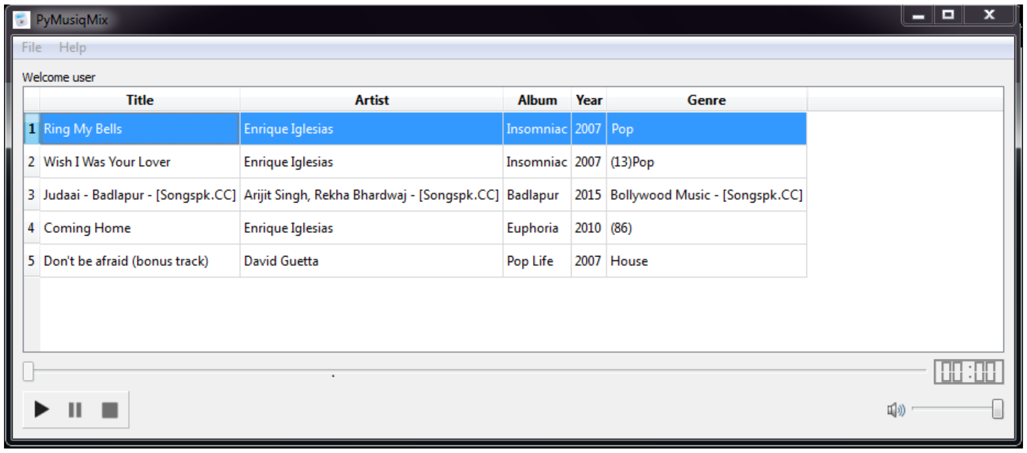


Figure 16: Add Files



include	title	Ring My Bells
exclude	album	Freedom
include	genre	Pop
include year	2000	2015

Figure 17: Constraints



Welcome user

	Title	Artist	Album	Year	Genre
1	Ring My Bells	Enrique Iglesias	Insomniac	2007	Pop
2	Wish I Was Your Lover	Enrique Iglesias	Insomniac	2007	(13)Pop
3	Judaai - Badlapur - [Songspk.CC]	Arijit Singh, Rekha Bhardwaj - [Songspk.CC]	Badlapur	2015	Bollywood Music - [Songspk.CC]
4	Coming Home	Enrique Iglesias	Euphoria	2010	(86)
5	Don't be afraid (bonus track)	David Guetta	Pop Life	2007	House

Figure 18: Generation of playlist

8 DISCUSSION

The project was aimed to work on a large set of songs but the intended amount was not met. The number of songs was limited to three hundred. We intend to reduce the time to minimum but could not achieve as the dataset is large and reading time increases. So,when the dataset kept on increasing,the time required to generate the playlist increases.

Also the reading time also kept on increasing. Inorder to compare the effectiveness of applying genetic algorithm rather than using simple search algorithm,the complexity was needed. But since finding the complexity was a tedious task,it was not obtained. So the comparison between the genetic algorithm and the basic search algorithm was not done. So,for the comparison to be done,the idea of user review was put forward. But the graphical representation of this was not able to be done.The concept of web integration with gui was not able to be done.

9 CONCLUSION

Dealing with a huge amount of music, a variety of music services are required to manage music in massive storage devices. So,inorder to reduce this tedious work by the users,we focus on passive applications, e.g., recommendation system and playlist generation.Here,a set of constraints is applied upon a huge set of songs according to the user preference and user behaviour. We define different types of constraints, including parameter-specified constraints and derived constraints

Our system constructs the derived constrains via the user behaviour and the user interaction. The concept of seed songs,where the user itself can individually specify the songs to be added into the playlist to be generated.We apply genetic algorithm to solve the playlist generation problem. We implemented our prototype, and performed experiments to show the feasibility and effectiveness of prototype. The prototype music player system was able to create a good playlist.

10 FUTURE ENHANCEMENT

We are planning to extend this application on all mobile phones. The mobile application developed will be user friendly so that large no of people can use.There are wide possibilities regarding the application of the project. Due to fact that the requirements are least for this system. We are planning to include additional features like mood of user.

References

[1] Jia-Lien Hsu and Shuk-Chun Chung, Department of Computer Science and Information Engineering, Fu Jen Catholic University,"*Constraint based Playlist Generation using genetic Algorithm* ", IEEE-2011

[2] S. Pauws and B. Eggen, Pats: "Realization and user evaluation of an automatic playlist generation," *Proceedings of the International Conference on Music Information Retrieval*, 2002.

[3] J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. T. Riedl," Evaluating Collaborative Filtering Recommender Systems," *ACM Transactions on Information Systems*, vol. 22, no. 1, January 2004.

[4] G. Adomavicius and Y. Kwon, "New recommendation techniques for multicriteria rating systems",*IEEE Intelligent Systems*, vol. 22, no. 3, pp. 4855, 2007.

[5] A. S. Harpale and Y. Yang, "Personalized active learning for collaborative filtering", *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval* , vol. 22, no. 3, 2008, pp. 9198.

[6] Y. Dai, H.W. Ye, and S.J. Gong, "Personalized recommendation algorithm using user demography information", *Proceedings of the Knowledge Discovery and Data Mining*, 2009, pp. 100103.

[7] T. Magno and C. Sable, "A comparison of signal-based music recommendation to genre labels, collaborative filtering, musicological analy-

sis, human recommendation, and random baseline", *Proceedings of the International Conference on Music Information Retrieval*, 2008.

[8] P. Lamere and D. Eck, "Using 3D visualizations to explore and discover music",*Proceedings of the International Conference on Music Information Retrieval*, 2007.

[9] T. Mitchell, "Machine Learning", *The McGraw-Hill Companies*, 1997.

[10] Genetic algorithm,"Wikipedia",available at *http://en.wikipedia.org/wiki/Genetic Algorithm.*