**YOUTUBE MIX**

**Franco Ramirez, Ricardo**

**Balón Pérez, Natali**

**Conesa Simó, Andreu**

**Quibus Marco, Alejandro**

The project consists in to create a storage system that features songs with self-generating playlists based on criteria specified by the user. The estructure of this Project is using the 3 layer architecture (Application layer, Business Logic layer, Data Access layer). Also the main objective of this project is to use and analyse the differents patterns to create relationships about the songs and the users. For this purpose we use 3 differents algorithms: Clique, Louvain and NewmanGirvan.

The program will have different users with different privileges (Manager and Client). Users will be distinguished by the following attributes: a (unique per user they identify) alias, password, name, date of birth (DD / MM / YYYY), gender, email.

The program will have a set of public songs and include the possibility of associating new songs to the user's account. As for the creation of playlists the user can manually generate them (specifying each song) or automatically (based on a modified criteria).

When logged in as customer this could add songs or delete songs. Moreover you can also create, access, modify and delete your own playlists. You will also have the option of applying self-generated lists according to specific criteria. On the other hand you can modify or delete your account.

When logged in as administrator, unlike the client, you can modify the attributes of the songs globally or for each user and delete any existing account on the system that is a client. However, the administrator will never erase their own.

Class Clique:

The function of this class is to generate n communities from a weighted graph, where each node belongs to at least one community.

This class contains two important algorithms the first is that given a graph of weight applied filter according to the weight of each relationship between two songs and generate a new graph.

The filter consists of applying a percentage to the average weight of the graph. This percentage indicates how strict you want to generate communities. Using the average weight of the original graph and the percentage of strict weight limit W is generated and a new graph is created with the same nodes but where only the edges greater than or equal to W. As weight preserved that obtained the new graph, the first algorithm, which is to generate n cliques (of different sizes) is performed.

To save these cliques obtained, I decide to use an ArrayList <HashSet <String >> that keeps me in each array element contents click (nodes that form the clique).

This set of cliques the order from largest to smallest size since it will allow us to work later in a more optimal way.

The second major algorithm is responsible for generating the k[[1]](#footnote-1)-clique communities given the Clicks communities.

The generation of communities is created based on a parameter K which restricted us on one hand the minimum number of k-clique containing a community (k-clique if you k = K-community and community is a k-it) and partly on the number of nodes in common that must be considered for community cliques (two cliques they are K-community if the two are at least K-1 nodes in common). So a key part of this algorithm is to calculate K, for this I chose to calculate given the maximum clique k, k click the minimum and the average of all k cliques.

To optimize the code aside from making the recursive function to process the binary matrix overlapping the creation of the binary matrix is done in a single loop that only covers the upper triangular matrix (since being a matrix of the same elements as rows and columns equals).

Class Louvain:

This method is used for large communities and get a runtime O (n log n).

This algorithm is an optimization of the detection method Modularity communities.

The steps of the algorithm are:

1 small communities looking by optimizing modularity locally.

2- add nodes from the same community and build a new network whose nodes are communities.

These steps are repeated until a maximum of modularity.

Class NewmanGirvan:

This algorithm detects communities phasing out the original network nodes. It focuses on the nodes that may be between communities and components are connected in the end are the communities themselves. The betweenness of each node is a measure of by how much a node is central, that is, the number of shorter between pairs of nodes passing through the first path. If more than one shortest path, each is assigned the same weight.

The steps of the algorithm are:

1 Reverse the weights of the edges of the graph so that the greater the weight ratio will lower nodes that are connected by that edge.

2- Calculate the node with the largest betweennes graph

3- Calculate how many times, depending on the desired dispersion, the process must be carried

4- Remove the edge by passing smallest roads around the graph and update the number of communities graph

5- Recalculate the node with the highest betweennes graph

6 Repeat steps 4 and 5 until the desired level of dispersion

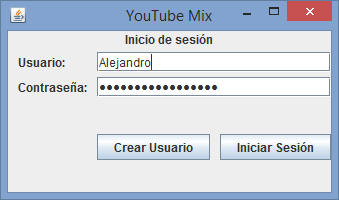
Once the process is completed by the spreading factor required, the remaining nodes in the graph are the various communities. To build these communities must travel to each node and get all adjacent. If a node has not been visited is added along with all nodes that it is connected to all communities. Once all nodes have come, all communities will have been generated.

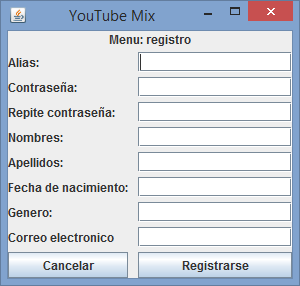
This Project was made in a group of four classmates.

My job was to code “Usuario”, “ConjuntoUsuario” (with his controller), “CtrlConjuntoUsuarios” (with his controller), “CtrlDatosConjuntosAgrupaciones”, “CtrlPresentacionConjuntoAgrupaciones”, “CtrlAlgoritmo” with their corresponding views.

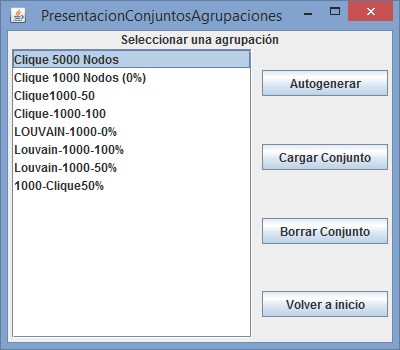
“CtrlDominio” and “CtrlVista” was coded between all the group.

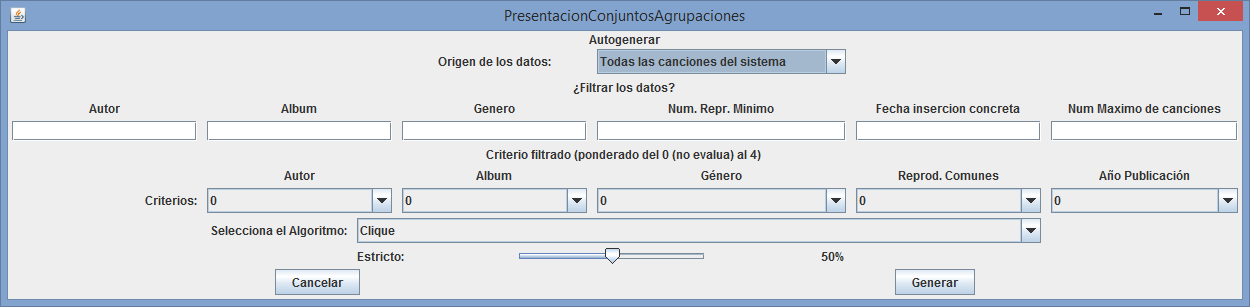
Screenshots

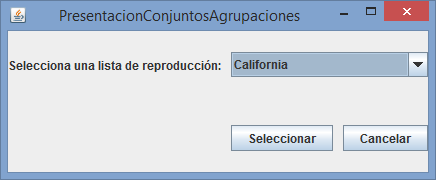


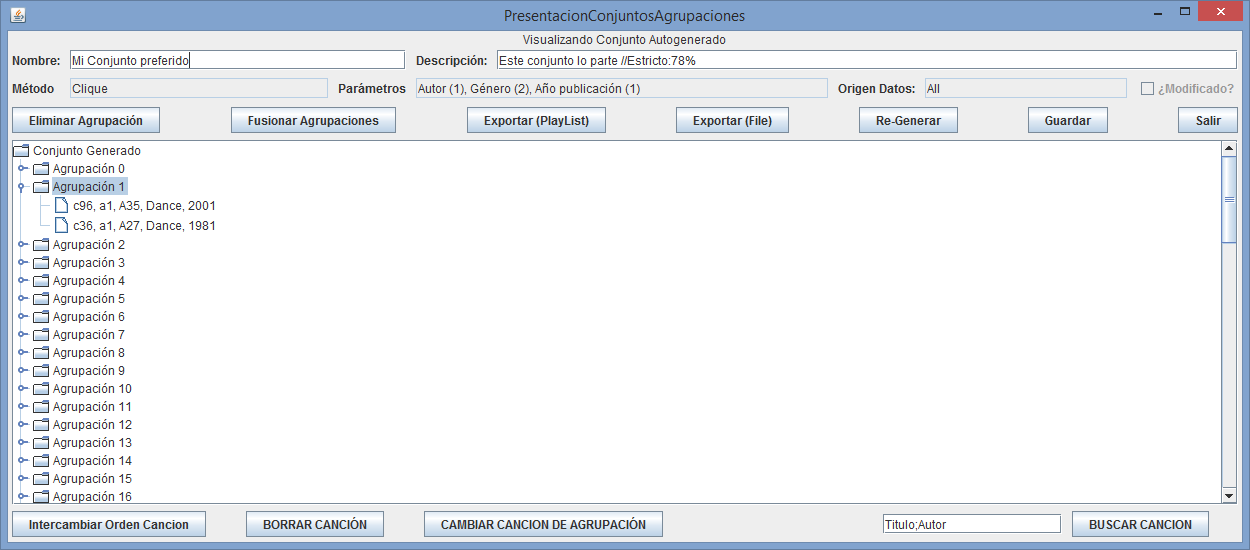












1. [↑](#footnote-ref-1)