Faculdade de Engenharia da Universidade do Porto

Spotify-ed: Music Recommendation and Discovery in Spotify

José Lage Bateira

Working version

Mestrado Integrado em Engenharia Informática e Computação

Supervisor: Fabien Gouyon

Co-Supervisor: Matthew Davies

June 23, 2014

Spotify-ed: Music Recommendation and Discovery in Spotify

José Lage Bateira

Mestrado Integrado em Engenharia Informática e Computação

June 23, 2014

Contents

List of Figures

Chapter 1

Methodologies

In this chapter, further details about the methodologies used in the developed prototype, as well as in the the validation process, will be explored.

The development process will be compared to the previous expectations of the outcome of the planned prototype.

By this point, the validation of the prototype will be analysed, by explaining the performed user tests, as well the analysis of the results.

1.1 Prototype

1.1.1 Main Features

The main features of RAMA's Spotify Application are: visualization of a map of a network of connected artists; edit the graph (expand and new map functions, as well as depth and branching parameters); Tags overlay and music artist info.

1.1.1.1 Visulization of the Artist Map

The application automatically draws the map with the current playing artist as the main node, as seen in Figure 1.1.

The graph-like structure of the map, is created by recursively fetching a list of related artists from each artist. Once a certain pre-established limit of recursive levels is reached[1], the algorithm stops.

The graph creation algorithm is as follows:

Listing 1.1: Simplified graph creation algorithm in Javascript (duplicate nodes checking is encapsulated in the insertNode function, as well as duplicate edges in the insertEdge function)

```
1 function buildGraph() {
2   // create a node with the root artist and insert it into the graph
3   this.insertNode(this.rootArtist);
4
5   // start constructing the graph recursively
6   this.expandNode(
7     this.depth - 1,
```

[1] depth value of a graph

```
8       this.rootArtist
9     );
10  }
11
12  // Expands the node of the parent artist by this.branching.
13  // It recursively decreases the depth parameter.
14  function expandNode(depth, parentArtist) {
15    var node = this.getNode(parentArtist);
16
17    // after expanding, the node will stop being a leaf
18    node.isLeaf = false;
19
20    // retrieve this.branching number of childs of the parent artist
21    var relatedArtists = parentArtist.getRelatedArtists(this.branching);
22
23    // for each child artist, insert and create a node into the graph
24    // and do the recursive call for the child, but with decreased depth.
25    for (var childArtist in relatedArtists) {
26      this.insertNode(childArtist);
27
28      // note that the stop condition of the recursion is depth <= 0
29      if (depth > 0)
30        expandNode(depth - 1, childArtist);
31    }
32  }
```

This algorithm, albeit simplified, represents the basic flow when constructing a graph, or more specifically, a tree. Since that, in this case of study, the direction of the edges of the graph is not relevant in any way to the artists' map, all of the edges are considered to be undirected.

Assuming that the insertNode() function only checks for duplicate nodes, i.e., it only inserts unique nodes into the graph, then the resulting graph is one of a tree, since there are no simple cycles in the graph. An example of this behaviour can be seen in Figure 1.2.

This approach, however, is not showing all of the available information. Given this same example (Figure 1.2), the artist node "Stone Temple Pilots" is a child of the node artist "Faith No More". The algorithm inserted the latter first into the graph. After that, when retrieving the childs of "Jane's Addiction", "Stone Templo Pilots" is contained in that list, and so the insertNode() function discards the node since it is a duplicate. But that means that there is a connection between the both of them that is being discarded.

Graph's Tree Mode

To build a graph with all the connections that exist between all of the artists in the graph, the insertNode() function would need to insert the missing edge into the graph by analysing the current graph state. This method creates a graph by definition, while the previous one created a tree graph. An example of this behaviour can be seen in Figure 1.3.

From now on, if the graph is a tree, it will be said that the algorithm is using the Tree Mode. *So the example 1.2 is a* Tree Mode *example, and the examples 1.3 and 1.4 are not* Tree Mode *examples*[2]..

Note (1.3) how "Stone Temple Pilots" is now child of both "Faith No More" and "Jane's

[2] Note that, sometimes, even if the algorithm is not on tree mode, it might generate a tree, simply because there were no missing edges between the nodes to be added to the graph.

Addiction". Also note that the same behaviour is visible with the nodes "Mad Season" and "Screaming Trees".

In this case, the user's perception on the artist "Stone Temple Pilots" is now different from the previous example (Figure 1.2). These added connections are, in a way, clustering together the most connected nodes. In this particular example, one could see an improvement with this approach: it contributes to the user's perception of the artist's network by making sure the user knows that those specific artists are more connected between them, than any others.

However, that is not always the case.

The fact is, that in this example, only three extra edges where added, meaning there is not much visual clutter in the visualization. Although one could argue that the "Mad Season" artist node is already disturbing the visual representation by being drawn over an edge.

With the exact same values of branching, depth and tree mode off, when creating a graph for the artist "Mariza", the visual clutter is so strong that the graph becomes confusing and not very helpful: Figure 1.4.

The cluster of nodes in the center makes it clear that those specific nodes are really connected between them. But then, the edges, that are forcing those nodes together, are creating a visual clutter that might not be so desirable. Instead of creating a visual map of the artists' network for the user to perceive, explore and change in its own way, the representation is more contained, static and not so visually appealing.

So on one hand, the *clustering* of the nodes seems like an interesting feature. It allows for a much more in-depth user access to the underlining information of the related artists. But on the other hand, the focus of the visualization (to show a map of the artists' network) was shifting to this cluster-like representation.

Both perspectives are advantageous depending on the data they are operating on (as seen in the previous examples). So instead of choosing only one method, both were chosen: by default, the graph creation algorithm is on tree mode, and the user can turn it off from a settings menu. This allows for the user to choose the visual representation method that suits its needs.

Depth and Branching values

The depth and branching values have been mentioned before in this chapter, but not further explained.

The **depth** value of a graph

1.1.1.2 Graph Edition

graph edition stuff

1.1.1.3 Tags Overlay

1.1.1.4 Artist Info

1.1.2 Development Process

1.2 Validation

1.2.1 User Tests

1.2.2 Data Analysis

1.3 Conclusions

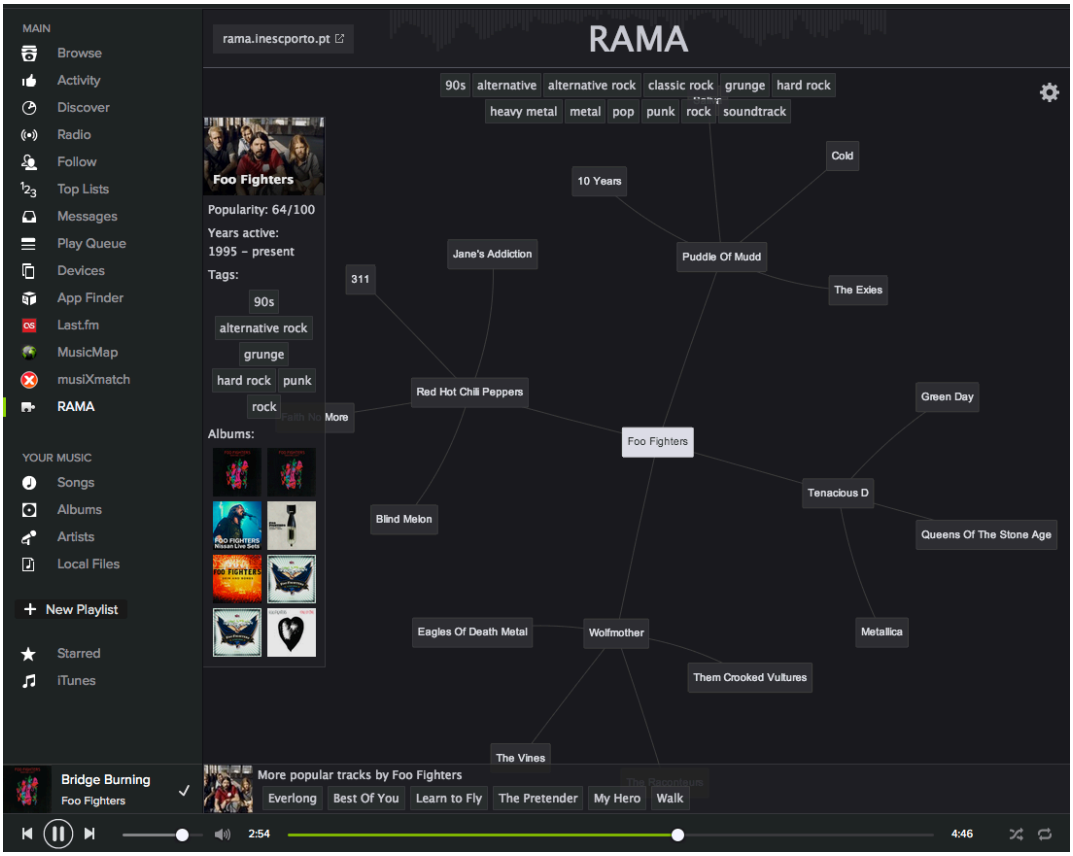

Figure 1.1: The first drawn graph uses the current playing artist (lower left corner) as the root node.

Figure 1.2: Graph created like a tree with "Red Hot Chilli Peppers" as the root node.

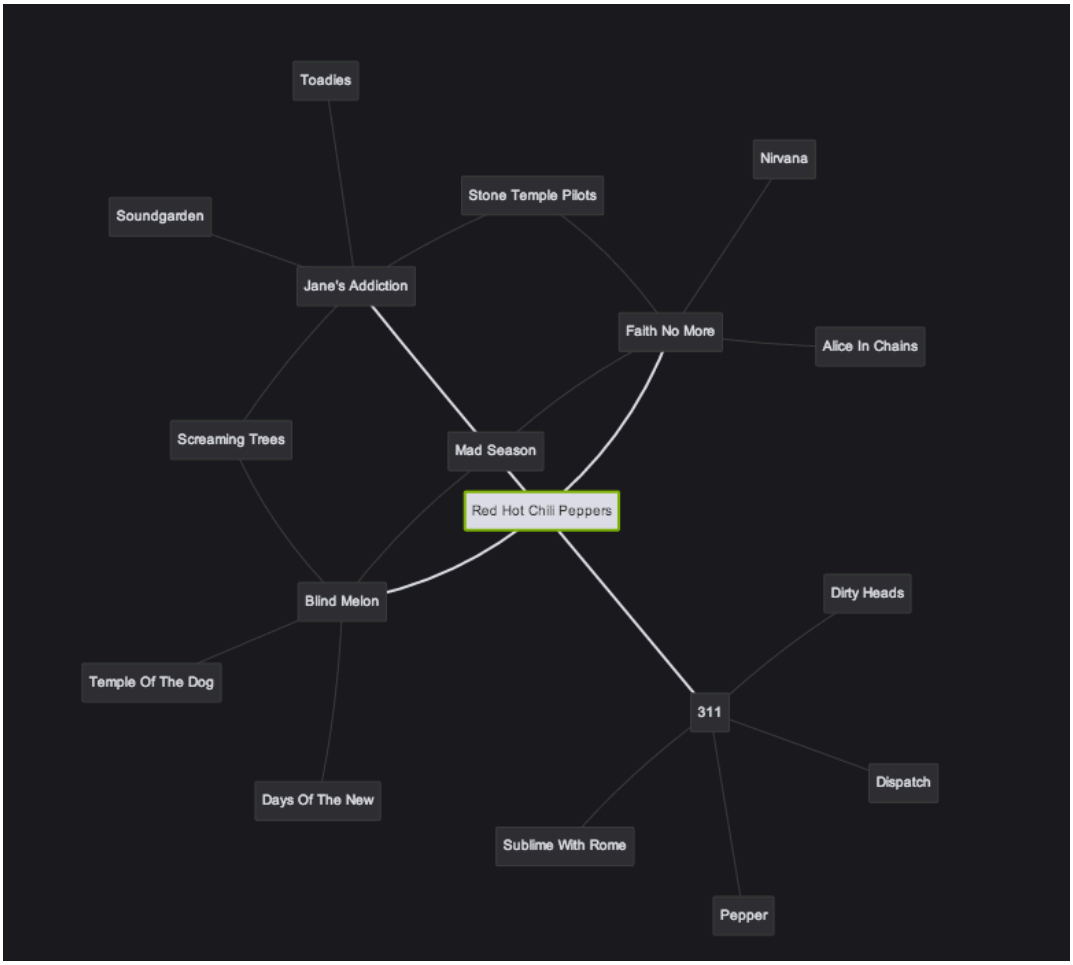

Figure 1.3: Graph created with all the connections with "Red Hot Chilli Peppers" as the root node.

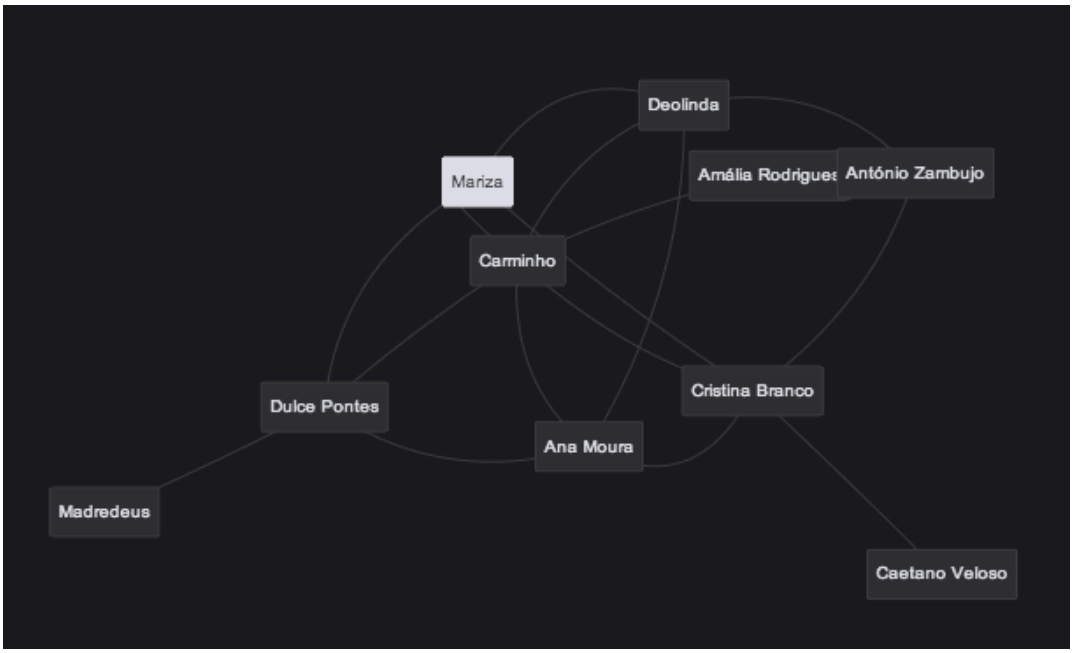

Figure 1.4: Graph created with all the connections with "Mariza" as the root node.

References