DISTRIBUTED COMPUTING GROUP
ETH ZÜRICH

LAB REPORT

Musicexplorer Winamp Plugin

Authors:
Chahine BENCHOHRA
Rahul JAIN

Supervisors:
Michael KUHN
Samuel WELTEN

Abstract

Traditional methods of browsing music collections like navigating through hierarchies of folders or searching for a song by metadata prove to be cumbersome when dealing with large music collections. In previous projects, a Euclidean map of the world of music was created where the Euclidean distance between two songs is inversely proportional to their similarity. In this project, we have implemented a plug-in for the Winamp Media Player which uses this map for discovering and playing similar songs from the user's media library.

October 6, 2010

Contents

I. Motivation

A. Introduction

The increasing popularity of online music stores (and file sharing) has resulted in a significant shift in the way in which people manage their music collections. Coupled with the increasing affordability of mass storage devices, it has resulted in most users now preferring to download songs rather than buying CDs or cassettes. Existing media player software provide us with something commonly referred to as the library view of the music collection. In this library view, it is possible to sort songs according to Artist, Album, Genre and other metadata encoded in the file. The same metadata can also be used to search for a particular song. Alternatively it is also possible to browse the music collection by navigating through the folder structure on the storage device. However, these methods can prove to be quite tedious when dealing with sufficiently large music collections. There is thus a need for alternative methods of browsing the music collection. We aim to implement a similarity based music player plugin called "Smart Shuffle". The motivation behind the plugin is to gauge the mood of the user from the currently playing song and his preference for the same. If he/she wishes to stay in the same mood, the plugin computes the closest matching song from the user's music collection and play it after the current song. The same procedure is then repeated for the next song. Conversely, if the user wishes for a change of mood, a dissimilar song is picked from his/her music collection and played after the current song. We now take a look at the background needed to find similar (and dissimilar) songs.

In previous projects [mus], an algorithm was developed that generates a point in a high-dimensional Euclidean space from an input song. The Euclidean distance between any two points in this space is inversely proportional to the similarity between the corresponding songs. We thus aim to build a map for the user's music collection and then use the same to find similar (and dissimilar) songs.

B. Platform

We chose the Winamp Media Player [win] as the software for which we develop our "Smart Shuffle" plugin. We decided to develop the plugin for Winamp since the software is popular amongst music aficionados, that is, users who are expected to have a large collection of songs. Winamp also provides a comprehensive API and examples projects that illustrate how plugins can be written for Winamp. This is collectively referred to as the Winamp SDK [SDK]. Moreover, the Winamp Forums [for] house a significant number of active plugin developers who are willing to provide feedback and answer questions regarding the Winamp SDK.

II. How it works: frontend

This section describes how the plugin works from the user's point of view. Note that all mentioned files are located in the user's Winamp application directory; e.g. under Windows 7 with Winamp 5.58: `C:\Users\<Username>\AppData\Roaming\Winamp\Plugins\`.

A. Start-up

The following is done when Winamp starts.

a) Loading configuration: The configuration is loaded from a file (`museek.conf`) which contains constants to be used in various parts of the plugin which may alter its efficiency; each line of this configuration file follows the same pattern:

`<parameter>  <value>`

Table I (Appendix) summarizes the list of recognized parameters.

b) Loading and building the map: The song coordinates are loaded from a file (`map.txt`). In case the file does not exist, or is outdated, the user is asked whether the media library should be scanned, that is, whether songs' coordinates should be downloaded from the MusicExplorer database. Downloading coordinates for big collections may take several minutes, during which Winamp remains usable thanks to multithreading. Note that there is, for now, no progress indicator indicating the number of songs for which the coordinates have been downloaded. However, the user is presented with a notification when the scan is complete.

c) Adding menu entries: Since "Smart Shuffle" is a new way of exploring the media library, a new menu entry is added in both the `Play` menu and contextual menu to toggle "Smart Shuffle". Besides, another menu entry is added to manually trigger a library scan. Note that these entries are grayed while the library is being scanned to indicate that "Smart Shuffle" is currently unusable.

B. Running

In order to function, "Smart Shuffle" needs a starting track which acts as a seed based on which the next tracks to be played are computed. The way this seed is determined varies:

- if "Smart Shuffle" is enabled while playing a track, this one will be considered as the seed;
- if no track is being played when "Smart Shuffle" is enabled, then a random track is selected from the user's media library and set as the seed.

Note that in the first case, the seed track is played back from the beginning as the "Smart Shuffle" mode is enabled. Though inconvenient, the playlist management API of Winamp doesn't make it possible to do better.

When a track starts playing, the potential next track to be played is prepared in a different thread as as to not slow down Winamp. This process actually selects two potential tracks:

- a *local* one in case we want to keep the same musical mood;
- a *remote* one otherwise.

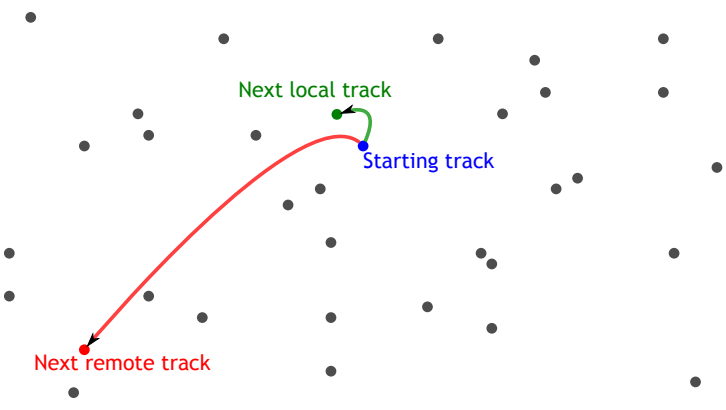

Figure 1. Basic idea of the algorithm: Prepare a local track (same musical mood) or remote track (different musical mood)

Whether the *local* or *remote* one is played depends on how much time the user spent listening to the previous track. Basically, if x is the number of seconds during which the user has been listening to the current track, and L the length of the track in seconds, then:

- with probability $\frac{x}{L}$, next track will be the *local* one;
- with probability $1 - \frac{x}{L}$, next track will be the *remote* one.

This means that if the user skips a song very quickly, it is highly likely that he/she will jump to a *remote* song. Note that x takes into account the time duration for which the song might have been paused, that is, if the user pauses the currently playing song then the duration for which the song was paused is not taken into consideration while computing the time that the track was played for.

The plugin only acts when **both** the conditions listed below are fulfilled:

- currently played song had ended or the user has explicitly switched to the next song;
- currently played song is the last one from the current playlist.

Only when both the above conditions are satisfied, does the plugin add a new song to the end of the playlist and starts playing it. Note that this makes it still possible to manipulate the playlist as usual: one may add/remove tracks to/from the playlist, or browse them...

In case the project has been compiled in debug mode, many operations give feedback in the log file (`museek.log`).

C. Exiting

Apart from waiting for all threads to terminate, nothing particular is done at exiting.

III. How it works: backend

A. Retrieving coordinates

Coordinates are available in an online database which can be queried via a web service. Each time the plugin needs some missing coordinates, it sends a GET request to the following URL (see Table I (Appendix)):

`http://<DATABASE_HOST><DATABASE_SCRIPT_PATH>`

with, as arguments, the artist and title of each song for which coordinates are needed. These arguments are properly encoded so as to not conflict with any special character used in URLs (e.g.. "/").

Coordinates retrieval was successfully tested on a collection of over 17 000 songs, covering most cases:

- well-tagged songs;
- badly-tagged songs;
- untagged songs;
- songs with all kinds of characters in their tags (including Chinese characters).

We thus expect it to work in all cases *with high probability*!

Coordinates are saved in a map file (`map.txt`) according to the following conventions:

- the first line will contain the number of tracks in the whole media library;
- then every line is about a single track of the map and is formatted as follows:

`<database response code> [<artist ID>] [<title ID>] <length> [<coordinate 1>] [<coordinate 2>] ...[<coordinate n] <path>`

where brackets denote optional fields, and where:

- `database response code` is the error code used by MUSICEXPLORER database (see Section A (Appendix));
- `artist ID` is the matching artist ID in MUSICEXPLORER database (see section A (Appendix));
- `title ID` is the matching track title ID in MUSICEXPLORER database (see section A (Appendix));
- `length` is the length of the track in seconds;
- `coordinate` i is the i^{th} coordinate of the track in the map;
- `path` is the absolute path to the song file in the user's machine; note that spaces are replaced by the character "|".

B. Finding a nearest neighbour

In an n dimensional space, finding the nearest neighbour of a point among N others is solvable in a reasonable time using k-dimensional trees (kd-trees), especially if $N >> 2^n$. In our case, N is assumed to be in the order of thousands (since we expect to deal with sufficiently large music libraries), and n is 32. Although we are not in the optimal case for k-dimensional trees, we still expect the nearest neighbour algorithm using kd-trees to be faster than an exhaustive search. However, no tests were run to verify this claim.

C. Finding a remote neighbour

To find a *remote* neighbour from a given point A, at about a given distance d:

- first, a uniformly distributed random point X is generated on the n-sphere whose centre is A and radius d; note that X is virtual, that is, it doesn't correspond to any track from the music library;
- then, the nearest neighbour of this virtual point is found using usual (previously explained) algorithm.

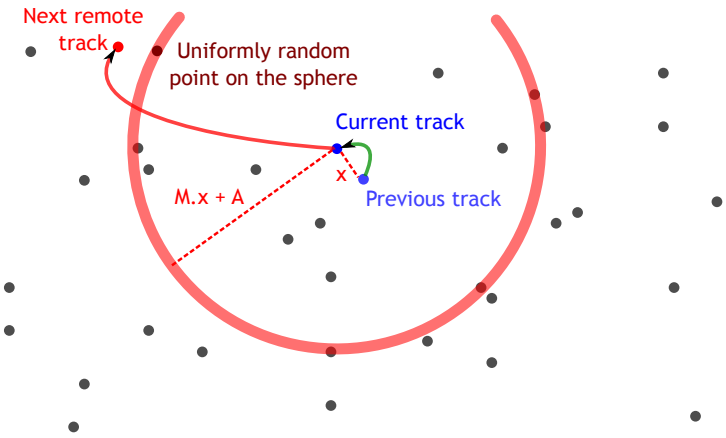

Figure 2. Finding a remote neighbour

As generating a uniformly random point in a n-sphere is not trivial, results from [Pol00] are used. Basically:

- n coordinates are independently generated using normal distribution;
- the resulting n-dimensional point is properly scaled to fall in the n-sphere.

The questions that arises is "how is d fixed?". Let's assume we have just switched from a track A_1 to its local (= nearest) neighbour A_2. Then (see table I (Appendix)):

$$d = \min(\texttt{REMOTE\_SCALE} \cdot |A_1 - A_2| + \texttt{REMOTE\_CONSTANT}, \texttt{REMOTE\_BOUND})$$

Now, let's assume we have just switched from a track A_1 to a *remote* neighbour A_3. To find A_3, we looked at the nearest neighbour of a random point in a sphere of radius r. Then (see Table I):

$$d = \min(\texttt{REMOTE\_SCALE} \cdot r + \texttt{REMOTE\_CONSTANT}, \texttt{REMOTE\_BOUND})$$

Let's first ignore the REMOTE_BOUND part. The radius of the sphere then grows by a constant factor at each track change, as soon as we always choose the *remote* neighbour as next track. Then, it takes $O(\log D)$ times to reach the diameter D of the whole map. For the user, this means less skips before finding a song that is really different from the currently played one.

Now, the bound is such that the new radius must not exceed the $O(D)$, since this would not make sense.

D. Used libraries

All libraries used in the project are cross platform and open source. This makes it possible to easily port MUSICEXPLORER plugin code for another media player running under any operating system.

d) libcurl: To download coordinates from the database, we need to perform HTTP queries; this is done by *libcurl*, the famous multiprotocol file transfer library.

e) ANN: This is an implementation of k-dimensional trees and nearest neighbour algorithms.

f) POSIX threads: Several parts of our algorithm take some time (scanning library, finding a nearest neighbour, loading map). To keep a smooth usage of Winamp, each of those parts are processed in a different thread using POSIX threads library, which is quite famous and widespread.

IV. WHAT COULD BE IMPROVED

A. Benchmarking

As said in section III-B, our exact nearest neighbour algorithm based on k-dimensional trees may not be that efficient. ANN library happens to include a module (ANNperf) to measure how much time and how many memory accesses it took to perform a nearest neighbour search. It would be interesting to compare current performance to an exhaustive search.

Another way of getting k-dimensional trees relevant is to perform only approximate nearest neighbour search. This could be done by setting the ERROR_BOUND constant to a non-zero value (see table I).

B. "Smart Shuffle" on current playlist only

Users with huge music collections may want to use "Smart Shuffle" over only a subset of their library; however, the current implementation does not allow such use of "Smart Shuffle".

C. Improve usability

Some parts of the plugin are still quite inconvenient to use:

- for library scan, the user does not know how long it will take; as it may take several minutes, the user may assume the library scan failed while it's still in progress; adding a progress bar window would be relevant;
- currently, the only way to enable "Smart Shuffle" is via menu entries; it would be much more convenient to add a button in Winamp's window; note that this would require to hack Winamp's skin;
- a configuration window would be more user-friendly to change some parameters; for now, the user has to edit the configuration file (assuming he/she finds it...);

D. Port code to other media players

Many parts of the code are independent from both Winamp and Windows; the others are easily adaptable. Besides, all libraries used in this project are cross platform and open source. This code may then be used as a base to implement similar plugins for any other media player on any other operating system.

E. Improve next neighbour randomization

Instead of using a uniform distribution to choose whether a next neighbour will be *local* or *remote*, another kind of distribution could be used, for example a Gaussian one; indeed, the Gaussian distribution would weight much less the few first and the few last seconds of listening.

F. Do not repeat artists

Current algorithm often happens to repeat artists, considering the local next track case, which may be undesired. To improve this, clustering could be used for example to gather all map points belonging to the same artist into a single point; the algorithm may then be applied to the map of those clusters.

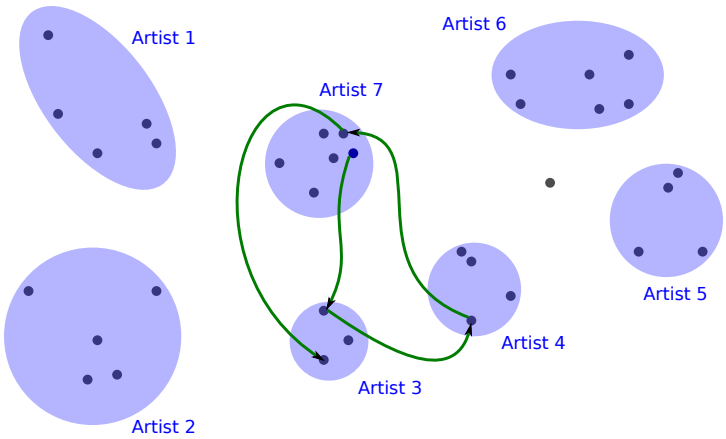

Figure 3. Algorithm using clustering with respect to artists

G. Finding a remote neighbour

The trick explained in section III-C works well provided current point is not on the edge of the universe, that is, the whole map (which is bounded since all coordinates are between 0 and 1). If it is, the uniformly random point may fall completely outside the map, and looking for its nearest neighbour wouldn't make sense any more. In the end, the random point should be sampled only the intersection of the n-sphere and the universe.

H. Perform incremental library scan

The current algorithm only allows to perform complete library scans. This is due to a restriction from the implementation of ANN library which seems to be unable to allocate memory dynamically. However, it is possible to fetch the coordinates of only newly added songs. The code can be modified so that we can check whether the coordinates of the song already exist in the map file, and only fetch the coordinates if we do not have the coordinates for the song.

V. Conclusion

The "Smart Shuffle" plugin implemented as part of this project provides the user with an adaptive playlist, that is, the songs in the playlist reflect the change in the user's mood. We believe that the plugin provides an intuitive way to interact with the music collection as the user does not need to manually select similar songs. We also hope that the code for the plugin can prove to be basis for similar plugins for other media players.

APPENDIX

Parameter	Default value	Comments
DIMENSIONS	32	How many coordinates are used for each song.
TRACKS_PER_QUERY	25	Coordinates are retrieved using an HTTP query with GET arguments; it is possible to process several tracks in a single query, which increases the number of GET arguments used; this parameters sets the number of tracks to be processed with each query.
ERROR_BOUND	0	Instead of finding the exact nearest neighbor, the algorithm may simply look for an approximate solution which distance is at most ERROR_BOUND over the distance to the exact one. You may use the exponential notation (that is, 1.234567E+03).
DATABASE_HOST	www.musicexplorer.org	See section III-A.
DATABASE_SCRIPT_PATH	/services_museek/getCoordinatesInPackagesNoXML.php	See section III-A
REMOTE_SCALE	1.1	See section III-C
REMOTE_CONSTANT	0.3	See section III-C
REMOTE_BOUND	$2\sqrt{2}$	See section III-C

Table I
CONFIGURATION FILE OPTIONS

REFERENCES

[for] Winamp Forums. http://forums.winamp.com/.

[mus] Music Explorer Overview. http://www.musicexplorer.org/page/index.php/home/background.

[Pol00] Jan Poland. Three different algorithms for generating uniformly distributed random points on the n-sphere. Oct 2000. http://www-alg.ist.hokudai.ac.jp/~jan/randsphere.pdf.

[SDK] Winamp SDK Contents. http://dev.winamp.com/wiki/SDK_Contents.

[win] Winamp Media Player. http://www.winamp.com/.