MelodyShape: a Library and Tool for Symbolic Melodic Similarity based on Shape Similarity

Version 1.3

Julián Urbano
Universitat Pompeu Fabra
urbano.julian@gmail.com

October 25, 2015

Abstract

MelodyShape is a Java library and tool to compute the melodic similarity between monophonic music pieces. It implements several algorithms that compute similarity based on the geometric shape that melodies describe in the pitch-time plane. All these algorithms have obtained the best results in the MIREX Symbolic Melodic Similarity task in 2010, 2011, 2012, 2013, 2014 and 2015 editions, as well as the best results reported for the 2005 collection. This document describes the tool and its execution options; for background information the reader is referred to [5].

1 Dependencies

MelodyShape is available both as source code and as an executable JAR package. It requires two libraries found in the Apache Commons project, which are not included in the JAR package: commons-cli and commons-math. JAR packages for both libraries can be downloaded from the Apache Commons project website. MelodyShape v1.3 was specifically implemented and tested with versions commons-cli-1.2 and commons-math3-3.2, also available from the MelodyShape website. Both libraries will have to be accessible from the Java classpath.

The easiest way to run the MelodyShape tool is to download all JAR files in the same directory, and then run Java from the command line (a graphical user interface is also available, see §2.7):

$ ls commons-cli-1.2.jar commons-math3-3.2.jar melodyshape-1.3.jar data

$ java -jar melodyshape-1.3.jar

usage: melodyshape-1.3 -q <file/dir> -c <dir> -a <name> [-k <cutoff>] [-l]
   [-t <num>] [-v] [-vv] [-gui] [-h]
-q <file/dir> path to the query melody or melodies.
-c <dir> path to the collection of documents.
-a <name> algorithm to run:
   - 2010-domain, 2010-pitchderiv, 2010-shape
   - 2011-shape, 2011-pitch, 2011-time
   - 2013-shapeh, 2013-time, 2013-shapetime
   - 2015-shapeh, 2015-time, 2015-shapetime
-k <cutoff> number of documents to retrieve.
-l show results in a single line (omits similarity scores).
-t <num> run a fixed number of threads.
-v verbose, to stderr.
-vv verbose a lot, to stderr.
-gui run with graphical user interface.

1For the latest version of the software and this document, please visit http://julian-urbano.info.
2http://music-ir.org/mirex/wiki/MIREX_HOME
3http://commons.apache.org
4https://github.com/julian-urbano/MelodyShape
Currently, MelodyShape v1.3 is compiled for Java 7. It can be redistributed and/or modified under the terms of the GNU General Public License version 3.

2 Tool Options

2.1 Basic Execution

MelodyShape requires one or more query melodies, a collection of melodies to sort according to their melodic similarity to the queries, and the name of the algorithm to compute those similarities. These three parameters must be indicated with arguments -q, -c and -a in the command line. For instance, we can compute the similarity of melody data/q01.mid with all melodies in directory data/coll/ using algorithm 2010-shape:

```
$ java -jar melodyshape-1.3.jar -q data/q01.mid -c data/coll/ -a 2010-shape
```

The output is a list of all files in data/coll/ sorted by melodic similarity to the query in descending order, along with the computed similarity score. Note that all files must have extension .mid or .midi. In addition, only one MIDI pitch can be on at any point in time.

2.2 Several Queries

MelodyShape can be run several times, once for each of the queries we have. Alternatively, it can be run once for all queries, resulting in a more efficient execution. The following would run MelodyShape will all queries in directory data/:

```
$ java -jar melodyshape-1.3.jar -q data/ -c data/coll/ -a 2010-shape
```

Results are again sorted by melodic similarity to the queries, one query after another. The first column identifies the query file, the second column identifies the document and the third column shows the computed similarity score.

2.3 Cutoff: -k

The -k command line argument can be used to indicate how many results to show per query. For example, to show only the two most similar melodies for each query:

```
$ java -jar melodyshape-1.3.jar -q data/ -c data/coll/ -a 2010-shape -k 2
```

Throughout this document, underlined text highlights the changes with the previous executions.
2.4 Single Line: -l

The -l command line argument indicates that results should be displayed in a single line, one line per query:

```
$ java -jar melodyshape-1.3.jar -q data/ -c data/coll/ -a 2010-shape -k 5 -l
q01.mid 06C6.mid 1065.mid 07D0.mid 0A5A.mid 04D7.mid
q02.mid 0859.mid 03FA.mid 0B15.mid 0454.mid 0B2C.mid
...
q06.mid 0859.mid 0B15.mid 0B2C.mid 0454.mid
```

Similarly, the first column identifies the query, and the rest identifies the most similar melodies in the collection, from left to right.

2.5 Verbose: -v and -vv

The -v command line argument can be used to display the execution progress, simply showing when each query starts being run:

```
$ java -jar melodyshape-1.3.jar -q data/ -c data/coll/ -a 2010-shape -k 5 -l -v
(1/6) q01.mid...done.
q01.mid 06C6.mid 1065.mid 07D0.mid 0A5A.mid 04D7.mid
(2/6) q02.mid...done.
q02.mid 0859.mid 03FA.mid 0B15.mid 0454.mid 0B2C.mid
...
(6/6) q06.mid...done.
q06.mid 0859.mid 0B15.mid 0B2C.mid 0454.mid
```

The -vv command line argument can be used to display real-time execution progress, some information about the algorithm used and time spent running each query:

```
$ java -jar melodyshape-1.3.jar -q data/ -c data/coll/ -a 2010-shape -k 5 -l -vv
Reading queries...done (6 melodies).
Reading collection...done (2000 melodies).
Instantiating algorithm...done.
Comparer: nGram(3,Hybrid(Cache(Freq(BSplineShape(8.0,1.0,0.5)))))
Ranker: Untie(nGram(3,Hybrid(Eq)))
Threads: 4
(1/6) q01.mid: [========================] 100% comparing...ranking...done (3 sec).
q01.mid 06C6.mid 1065.mid 07D0.mid 0A5A.mid 04D7.mid
(2/6) q02.mid: [========================] 100% comparing...ranking...done (2 sec).
q02.mid 0859.mid 03FA.mid 0B15.mid 0454.mid 0B2C.mid
...
(6/6) q06.mid: [========================] 51% comparing...
```

All this extra information is output to the standard error stream, so that the actual results per query can be easily set apart with a pipeline:

```
$ java -jar melodyshape-1.3.jar -q data/ -c data/coll/ -a 2010-shape -l -vv > out
Reading queries...done (6 melodies).
Reading collection...done (2000 melodies).
Instantiating algorithm...done.
Comparer: nGram(3,Hybrid(Cache(Freq(BSplineShape(8.0,1.0,0.5)))))
Ranker: Untie(nGram(3,Hybrid(Eq)))
Threads: 4
(1/6) q01.mid: [========================] 100% comparing...ranking...done (3 sec).
(2/6) q02.mid: [========================] 100% comparing...ranking...done (2 sec).
...
(6/6) q06.mid: [========================] 100% comparing...ranking...done (4 sec).
```
2.6 Number of Threads: -t

The -t command line argument can be used to indicate the number of threads to use by the algorithms. By default, as many threads as CPU cores found in the machine will be used. We can, for instance, restrict CPU usage to just two threads:

$ java -jar melodyshape-1.3.jar -q data/ -c data/coll/ -a 2010-shape -t 2

2.7 Graphical User Interface: -gui

The -gui command line argument can be used to run MelodyShape with a graphical user interface rather than as a command line tool, which will show up a window as in Figure 1. Alternatively, the graphical user interface can be run directly by running the melodyshape-1.3.jar JAR file from a windows or file explorer, usually by double-clicking on it.

![Figure 1: MelodyShape graphical user interface.](image)

The options available from the graphical user interface are basically those available from the command line. The user can copy the results to the system clipboard by right-clicking on the results panel and selecting the Copy All option.

3 Algorithms

MelodyShape implements several melodic similarity algorithms based on a geometric model that represents melodies as spline curves in the pitch-time plane. The similarity between two melodies is then computed with a sequence alignment algorithm between sequences of spline spans: the more similar the shape of the curves, the more similar the melodies they represent. MelodyShape v1.3 implements algorithms based on this model that have been submitted to the MIREX Symbolic Melodic Similarity task in 2010, 2011, 2012, 2013, 2014 and 2015.

The following table shows the -a command line argument to be used for each algorithm, with the corresponding official MIREX submission name. For instance, argument -a 2011-shape corresponds to algorithm UL1-Shape from MIREX 2011, which is the same as algorithms JU4-Shape, ULMS1-ShapeH, JU1-ShapeH, JU1-ShapeH and JU1-ShapeH from MIREX 2010, 2012, 2013, 2014 and 2015, respectively.

---

6Algorithm JU3-ParamDeriv from MIREX 2010 is not implemented in MelodyShape.
The reader is referred to the corresponding MIREX participation report for a detailed description of each of these algorithms [4, 6, 7, 1, 2, 3]. For a general description of the geometric model, please refer to [5].

**Acknowledgments**

This work is supported by an A4U postdoctoral grant and a Juan de la Cierva postdoctoral fellowship.

**References**


