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

## Version 1.0

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

November 27, 2013

**Abstract**

`MelodyShape`[1] 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[2] Symbolic Melodic Similarity task in 2010, 2011, 2012 and 2013 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 [3].

## 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[3]. `MelodyShape` v1.0 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:

```
$ ls
commons-cli-1.2.jar  commons-math3-3.2.jar  melodyshape-1.0.jar  data

$ java -jar melodyshape-1.0.jar
usage: melodyshape-1.0 -q <file/dir> -c <dir> -a <name> [-k <cutoff>] [-l]
                       [-t <num>] [-v] [-vv] [-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
              - 2012-shapeh, 2012-shapel, 2012-shapeg, 2012-time, 2012-shapetime
              - 2013-shapeh, 2013-time, 2013-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.
-h            show this help message.

MelodyShape 1.0 Copyright (C) 2013 Julian Urbano <urbano.julian@gmail.com>
```

[1] For the latest version of the software and this document, please visit http://julian-urbano.info.
[2] http://music-ir.org/mirex/wiki/MIREX_HOME
[3] http://commons.apache.org

```
This program comes with ABSOLUTELY NO WARRANTY.
This is free software, and you are welcome to redistribute it
under the terms of the GNU General Public License version 3.
```

Currently, `MelodyShape` v1.0 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-1.0.jar -q data/q01.mid -d data/coll/ -a 2010-shape
06C6.mid   0.48581398
1065.mid   0.16719302
07D0.mid   0.15724328
0A5A.mid   0.14414513
04D7.mid   0.10826124
...
```

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/`[4]:

```
$ java -jar melodyshape-1.0.jar -q data/ -d data/coll/ -a 2010-shape
q01.mid   06C6.mid   0.48581398
q01.mid   1065.mid   0.16719302
...
q01.mid   0C29.mid   -0.18696516
q02.mid   0859.mid   0.44809958
...
q06.mid   0B63.mid   -0.19795232
q06.mid   0EA2.mid   -0.22072806
```

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-1.0.jar -q data/ -c data/coll/ -a 2010-shape -k 2
q01.mid   06C6.mid   0.48581398
q01.mid   1065.mid   0.16719302
q02.mid   0859.mid   0.44809958
q02.mid   03FA.mid   0.18398533
q03.mid   13ED.mid   0.35113359
q03.mid   015D.mid   0.32187245
```

[4] Throughout this document, underlined text highlights the changes with the previous executions.

```
q04.mid   0D1E.mid   0.15161275
q04.mid   0BEA.mid   0.14805929
q05.mid   0B2C.mid   0.18844743
q05.mid   0F60.mid   0.13771394
q06.mid   0859.mid   0.26753167
q06.mid   0B15.mid   0.24442237
```

### 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-1.0.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   0F0A.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-1.0.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   0F0A.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-1.0.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-1.0.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).

$ cat out
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   0F0A.mid   0B2C.mid   0454.mid ...
```

### 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-1.0.jar -q data/ -c data/coll/ -a 2010-shape -t 2
```

## 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` v1.0 implements algorithms based on this model that have been submitted to the MIREX Symbolic Melodic Similarity task in 2010, 2011, 2012 and 2013.

The following table shows the `-a` command line argument to be used for each algorithm, with the corresponding official MIREX submission name[5]. For instance, argument `-a 2011-shape` corresponds to algorithm UL1-Shape from MIREX 2011, which is the same as algorithms JU4-Shape, ULMS1-ShapeH and JU1-ShapeH from MIREX 2010, 2012 and 2013, respectively.

| `-a` argument | MIREX 2010 | MIREX 2011 | MIREX 2012 | MIREX 2013 |
|---|---|---|---|---|
| `2010-domain` | JU1-Domain | | | |
| `2010-pitchderiv` | JU2-PitchDeriv | | | |
| `2010-shape` | JU4-Shape | UL1-Shape | ULMS1-ShapeH | JU1-ShapeH |
| `2011-shape` | JU4-Shape | UL1-Shape | ULMS1-ShapeH | JU1-ShapeH |
| `2011-pitch` | | UL2-Pitch | | |
| `2011-time` | | UL3-Time | ULMS5-Time | JU3-Time |
| `2012-shapeh` | JU4-Shape | UL1-Shape | ULMS1-ShapeH | JU1-ShapeH |
| `2012-shapel` | | | ULMS2-ShapeL | |
| `2012-shapeg` | | | ULMS3-ShapeG | |
| `2012-time` | | UL3-Time | ULMS5-Time | JU3-Time |
| `2012-shapetime` | | | ULMS4-ShapeTime | JU2-ShapeTime |
| `2013-shapeh` | JU4-Shape | UL1-Shape | ULMS1-ShapeH | JU1-ShapeH |
| `2013-time` | | UL3-Time | ULMS5-Time | JU3-Time |
| `2013-shapetime` | | | ULMS4-ShapeTime | JU2-ShapeTime |

The reader is referred to the corresponding MIREX participation report for a detailed description of each of these algorithms [2, 4, 5, 1]. For a general description of the geometric model, the reader is referred to [3].

## Acknowledgments

This work is supported by an A4U postdoctoral grant.

[5] Algorithm JU3-ParamDeriv from MIREX 2010 is not implemented in `MelodyShape` v1.0.

# References


[1] Julián Urbano. MIREX 2013 Symbolic Melodic Similarity: A Geometric Model supported with Hybrid Sequence Alignment. Technical report, Music Information Retrieval Evaluation eXchange, 2013.

[2] Julián Urbano, Juan Lloréns, Jorge Morato, and Sonia Sánchez-Cuadrado. MIREX 2010 Symbolic Melodic Similarity: Local Alignment with Geometric Representations. Technical report, Music Information Retrieval Evaluation eXchange, 2010.

[3] Julián Urbano, Juan Lloréns, Jorge Morato, and Sonia Sánchez-Cuadrado. Melodic Similarity through Shape Similarity. In S. Ystad, M. Aramaki, R. Kronland-Martinet, and Kristoffer Jensen, editors, *Exploring Music Contents*, pages 338–355. Springer, 2011.

[4] Julián Urbano, Juan Lloréns, Jorge Morato, and Sonia Sánchez-Cuadrado. MIREX 2011 Symbolic Melodic Similarity: Sequence Alignment with Geometric Representations. Technical report, Music Information Retrieval Evaluation eXchange, 2011.

[5] Julián Urbano, Juan Lloréns, Jorge Morato, and Sonia Sánchez-Cuadrado. MIREX 2012 Symbolic Melodic Similarity: Hybrid Sequence Alignment with Geometric Representations. Technical report, Music Information Retrieval Evaluation eXchange, 2012.