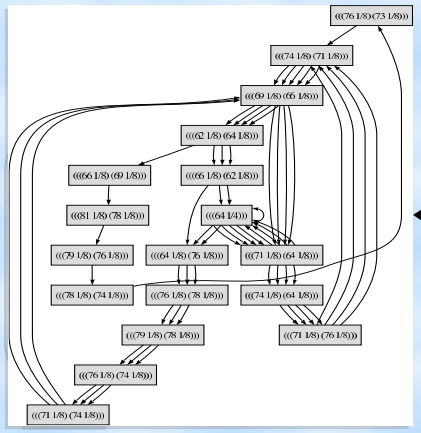
A Markov-chaining melody analysis and composition program written in Common Lisp

By Abraham Stimson, '08
Marlboro College

April, 2008

Features

- Read tunes from MIDI or ABC format; exports tunes in ABC
- Stores note or fragment transition information from an arbitrary number of files
- Employs nth order Markov chaining for composing fragments of melody
- Scans the repetition pattern of a single tune to use as a model, then employs that repetition model when composing for added coherence
- Exports graphs, in DOT format, of the Markov chain in use
- Includes several functions for "massaging"and formatting the input data, such as
  - subdividing a melody into measures
  - removing beginning pickup notes
  - detecting key, and converting input tunes to a common key
- Tested with both CLISP and OpenMCL
- Online tune generator at http://www.cs.marlboro.edu/~astimson/tune-generator



Overview

This is a general-purpose melody analysis and composition package written in Common Lisp. The content is database-driven, where any number human-composed melodies are scanned for note transition information after breaking a melody into melodic units of a given duration. Composition proceeds through taking a "random walk" through this probabilistic state transition graph, also know as a Markov chain.

This project is designed to be particularly suitable for composing in the style of traditional Irish fiddle tunes, and analyzing their internal repetition structures. When composing, large-scale structure of a tune is achieved by mimicking the "rhyme scheme", or repetition pattern, of one particular input tune. This approach generally increases the coherence and listenability of the output.

A Lisp teaser

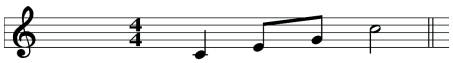
```
;;; This is one of about 80 functions in the package.
;; Beginning at the end of an event list, group into
;; fragments whose duration totals n-beats.  This list will
;; need to be reversed to attain the original sequencing as
;; it appeared in the tune.

(defun group-events-from-end (events n-beats)
  (let ((current (break-off events n-beats)))
    (cons (if (cdr current)
              (cdr current)
              (list events))
          (if (cdr current)
              (group-events-from-end (car current) n-beats)
              nil))))
```

Example session

```
MIDI> (setf *test-tune* (read-tune-in
"../data/oneils/reels/tune494.mid"))
Parse midi file ../data/oneils/reels/tune494.mid
into the CLOS.
Read in tune from ../data/oneils/reels/tune494.mid.
Convert to fractional durations with quantizing
factor 10.
((64 1/4) (71 1/8) (64 1/8) (74 1/8) (64 1/8) (71
1/8) (76 1/8) (74 1/8) (71 1/8) (69 1/8) (66 1/8)
(62 1/8) (64 1/8) (66 1/8) (62 1/8) (64 1/4) (71
1/8) (64 1/8) (74 1/8) (64 1/8) ... )

MIDI> (setf *repetition-pattern* (assign-symbols
(time-divide *test-tune* 1/2)))
(A B C D A B C E A B C D A B C E F G H D F G H E F G
H I J K C E)

MIDI> (setf *test-tune-in-quarter-note-subdivisions*
(time-divide *test-tune* 1/4))
(((64 1/4)) ((71 1/8) (64 1/8)) ((74 1/8) (64 1/8))
((71 1/8) (76 1/8)) ((74 1/8) (71 1/8)) ((69 1/8)
(66 1/8)) ((62 1/8) (64 1/8)) ((66 1/8) (62 1/8))
((64 1/4)) ((71 1/8) (64 1/8)) ((74 1/8) (64 1/8))
... )

MIDI> (setf *state-transition-matrix*(fill-stm
:tune-events *test-tune-in-quarter-note-
subdivisions* :order 1 :input-stm nil))
(((((76 1/8) (73 1/8))) ((74 1/8) (71 1/8)))
 ((((78 1/8) (74 1/8))) ((76 1/8) (73 1/8)))
 (((74 1/8) (64 1/8))) ... ))))

MIDI> (compose-by-pattern *state-transition-matrix*
2 1 *repetition-pattern*)
Compose by pattern:
 Number of STM entries: 18
 Grain size: 2
 Order: 1
 Repetition pattern: A B C D A B C E A B C D ...

((66 1/8) (62 1/8) (64 1/4) (71 1/8) (64 1/8) (64
1/4) (64 1/8) (76 1/8) (76 1/8) (78 1/8) (79 1/8)
(78 1/8) (76 1/8) (74 1/8) (66 1/8) (62 1/8) (64
1/4) (71 1/8)
(64 1/8) (64 1/4) (64 1/8) (76 1/8) ... )
```

Notation methods used

Staff notation:

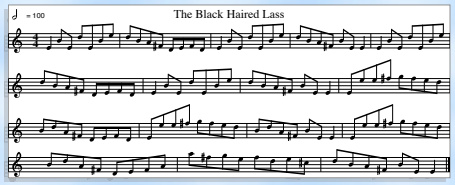


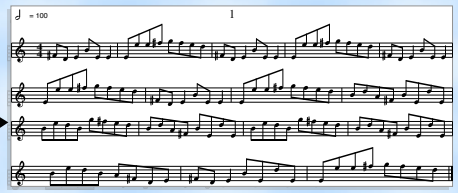
Event notation: `(60 1/4) (65 1/8) (68 1/8) (72 1/2))`

ABC notation: `C2 EG c4`

tune494.abc

```
X: 1
T:The Black Haired Lass
M:4/4
L:1/8
Q:1/2=100
K:C
E2BE dEBe|dBA^F DEFD|E2BE dEBe|dBA^F BEE2|
E2BE dEBe|dBA^F DEFD|E2BE dEBe|dBA^F BEE2|
Eee^f gfed|BdA^F DEFD|Eee^f gfed|BdA^F
BEE2|Eee^f gfed|BdA^F DEFA|a^fge fde^c|
dBA^F BEE2|]
```





• Acknowledgments

• I would very much like to thank Jim Mahoney, my faculty advisor, and without whom this project would never have been possible; and also Matt Ollis, who provided mathematical inspiration along the way, and Stan Charkey for help with the music theory.

• References/Links

• abc2MIDI – used to convert between abc and MIDI and for visual formatting of music notation: http://www.abc.sourceforge.net/abcMIDI/.
• Uses the MIDI Lisp library, available from: http://www.cliki.net/midi.
• David Cope, http://arts.ucsc.edu/faculty/cope/.
• Barfly a shareware WYSIWYG ABC notation editor and player: www.barfly.dial.pipex.com/.