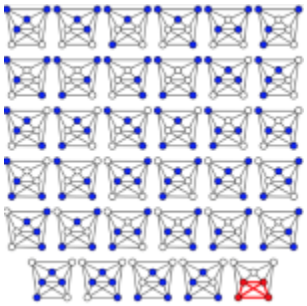
Musical Random Walks Over Weighted Graphs

2016-02-02 BY GENE



tl;dr: https://github.com/ology/Music/blob/master/random-walk

In this post, I illustrate a simple technique in Perl 5 to perform random walks over (node-edge) graphs, adding the named, “semantic” vertices to a MIDI score.

The image on the left is not generated by the random-walk program, but is just a related illustration. :-)

OK – on with the code!

First, there is the standard perl preamble:

```
#!/usr/bin/env perl
use strict;
use warnings;
```

Next, come the handy module imports:

```
use Graph::Weighted;
use List::Util::WeightedChoice qw( choose_weighted );
use MIDI::Simple;
```

Here we can see that the program is going to handle weighted graphs, choose from the node values and track the progress as MIDI data.

Next up is the declaration of the crucial program parameters – the number of notes to play and the starting graph node. These can be taken from the command-line (with shift) or if not, set to defaults:

```
my $n = shift || 4;       # Number of notes
my $initial = shift // 0; # Initial graph node
```

The first task is to define the treble note, bass note, note duration and note velocity transition graphs. The transitions are the probabilities to other nodes in the graph. Here is the graph for the treble notes:

```
my $treble = Graph::Weighted->new();
$treble->populate(
    {
        0 => { label => 'C5',  2 => 0.4, 6 => 0.6 },
        1 => { label => 'D5',  3 => 0.4, 4 => 0.6 },
        2 => { label => 'Ds5', 1 => 0.5, 3 => 0.5 },
        3 => { label => 'F5',  5 => 0.4, 4 => 0.6 },
        4 => { label => 'G5',  2 => 0.4, 3 => 0.6 },
        5 => { label => 'Gs5', 4 => 0.4, 6 => 0.6 },
        6 => { label => 'As5', 0 => 0.4, 3 => 0.6 },
    }
);
```

Combined with the bass, duration and velocity graphs, this can be used to generate a melody of sorts. Check out the code (linked above) to see the other graphs that are used.

So how do we go about creating a melody, you ask? Well, the meat of the program is the note creation and collection:

```
my $notes = collect_notes( $n, $initial, $velocity, $duration, $treble, $ba
```

More on that in a bit… Because the end of the program is just to add the notes to the MIDI score and write it to a *.mid* file (named for the program):

```
my $score = setup_midi();

for my $note ( @$notes ) {
    $score->n( @$note );
}

$score->write_score( $0 . '.mid' );
```

OK. The meat of the program is to do the random walk over multiple graphs and collect the note information. This is the code:

```
sub collect_notes {
    my ( $n, $initial, $velocity, $duration, $treble, $bass ) = @_;

    my ( $t_vertex, $b_vertex, $d_vertex, $v_vertex ) = ($initial) x 4;

    my $notes = [];

    for my $i ( 1 .. $n ) {
        my $treb = $treble->get_vertex_attribute( $t_vertex, 'label' );
        my $low  = $bass->get_vertex_attribute( $b_vertex, 'label' );
        my $dura = $duration->get_vertex_attribute( $d_vertex, 'label' );
        my $velo = $velocity->get_vertex_attribute( $v_vertex, 'label' );

        push @$notes, [ $velo, $dura, $treb, $low ];

        if ( $i < $n ) {
            $t_vertex = next_vertex( $treble, $t_vertex );
            $b_vertex = next_vertex( $bass, $b_vertex );
            $d_vertex = next_vertex( $duration, $d_vertex );
            $v_vertex = next_vertex( $velocity, $v_vertex );
        }
    }

    return $notes;
}
```

First we accept the arguments to this subroutine. Then set the initial vertices to

the given initial node. Next we walk the graph for the number of notes desired. For each iteration, we get the vertex label (the MIDI note information) and append that to our growing note collection. If we are not done iterating, we request the next vertex based on the vertex edge values.

This is done with the following subroutine:

```
sub next_vertex {
    my ( $g, $vertex ) = @_;

    my $successors = [];

    for my $successor ( $g->successors($vertex) ) {
        push @$successors, {
            vertex => $successor,
            weight => $g->get_cost( [ $vertex, $successor ] ),
        };
    }

    my $choice = choose_weighted( $successors, sub { $_[0]->{weight} } );

    return $choice->{vertex};
}
```

This code gathers the immediate successors of the given vertex and notes the edge cost of each – the weight. Then this is used to choose a new vertex by selecting one of its edges, given its weighted probability.

So the final result of this is a potential melody that can be played with something like Timidity or imported into your favorite DAW and tweaked at will. Also, you could generate a small phrase and then enhance it with counterpoint type transformations. This is done in the related program, random-walk-transform.

Anyway, here is a 64 note example: random-walk.mid

FILED UNDER: MUSIC, SOFTWARE
TAGGED WITH: MIDI, PERL

Epistemologist-at-large

^ Top