

Frankenstein:
a Framework for musical improvisation

Davide Morelli

24.05.06

summary

- what is the frankenstein framework?
- step1: using Genetic Algorithms
- step2: using Graphs and probability matrices
- step3: toward mpeg7
- wishlist

Frankenstein

- is an Open Source Framework for music composition and improvisation
- the team:
 - Davide Morelli - www.davidemorelli.it
 - David Plans Casal - www.davidcasal.com
- David's PhD project in UEA (Norwich)
- Project partially financed by LAM www.livealgorithms.org - Goldsmiths College (London)

who is Frank? motivations

- interest in applying AI techniques to music
- create a compositional tool
- an aid while improvising in difficult musical languages (for example: microtonal, spectral, etc..)
- understand the musical language

who is Frank? objectives

- capable of jamming with one (or more) human musician
- rhythm + melody + harmony + structure
- cross-stylistic
- realtime
- interactive (not only reactive)
- usable on-stage
- pass the Turing-test

why Frankenstein?

- “Frankensteinian Methods for Evolutionary Music Composition” Peter M. Todd & Gregory M. Werner
 - we started from Todd's idea of co-evolution
 - because we share the same hubris as Victor Von Frankenstein
 - to keep in mind that our creature will probably be a monster, unable to survive in the real world
 - but we love it!



puredata

the software is in the form of a set of pd externals.

pd is an audio/video dataflow programming language.

why?

- basilar audio components ready to use
- it is Open Source
- nice community
- easy to prototype

an external is an extension: a new object usable as a native object.

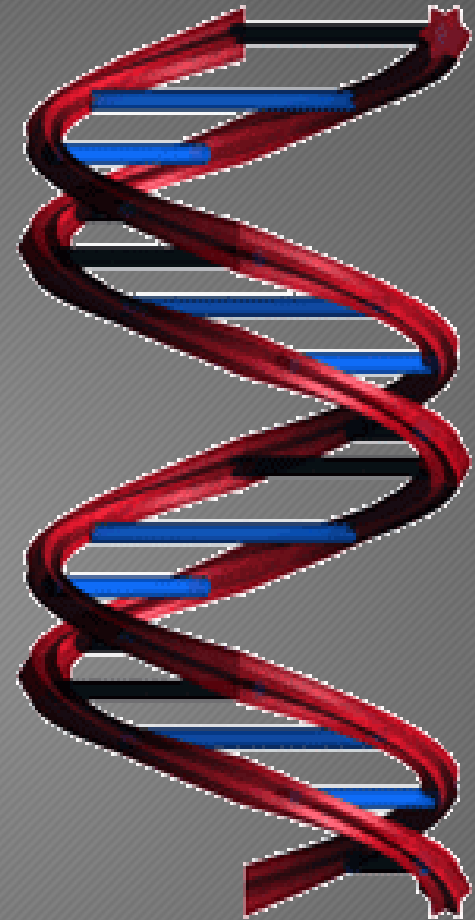
Externals can be coded in C and C++ (Python, Ruby, C#)

the Frankenstein framework – Davide Morelli



step1: GA

- traditional GA vs Co-evolutionary GA
 - fitness function evolves together with the solutions
- darwinian selection vs sexual selection:
 - females choosing males
- nice because it gives us
 - a new solution every generation
 - every solution is somehow related to the previous ones



GA: co-evolution

- 2 populations: male or female
- males and females have identical genotype structure
- each generation every female evaluates N males (randomly chosen)
- one male is selected and becomes father
- he and the other males can be chosen by another female
- the child can be both male or female
- female population changes every generation

GA: the genotype

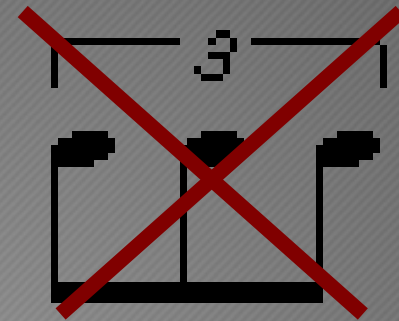
a rhythm is expressed as an array of bits:

- each genotype is a bar
- 8 voices
- 16 beats in a bar
- I need an array of 16 Bytes to express rhythm:
- every Byte is a beat
- every bit tells us if that voice is active or not in that beat

GA: the genotype limits

this representation has many limits:

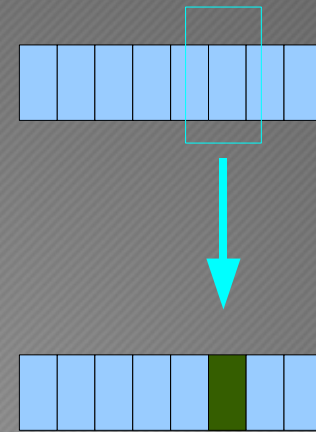
- only 1 bar long rhythms are possible
- no triplets, etc..



GA: mutations

mutation function implemented:

- silence where it was an event
- event where it was silence

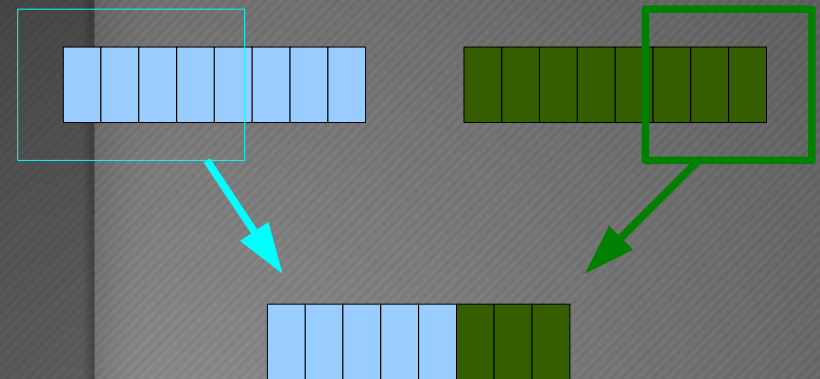


GA: crossover

crossover function:

randomly select a split point N

- bits 0 to N from mother
- bits N to genotype length from father
- sometimes apply the mutation function



GA: fitness

fitness functions:

- how similar are male and female (co-evolutionary)
- how dense is male (traditional)
- how many consecutive events (traditional)



GArhythm

- the implemented external is GArhythm
- has been used on stage:
 - GA2005 in Milan (www.generativeart.com)
 - SMC2005 in Salerno (www.smc05.unisa.it)

chord_melo

another GA external

same idea applied to tonal melodies

- unlike other similar works we take care of melody shape instead of exact notes

example:

- original



- local transitions



- shape



harmonizer

another GA external that generates voicing

- you give it an initial chord
- the starting positions (MIDI notes)
- the target chord

And it will generate a correct voicing

- western musical rules have been coded
- no need of more than 1 generation

GA limitations

- GA are really good for evolving musical material (rhythms, melodies) but
- have no memory
- not possible to create musical form
 - I can't have A-B-A
 - I can only have A-A'-A''

memory

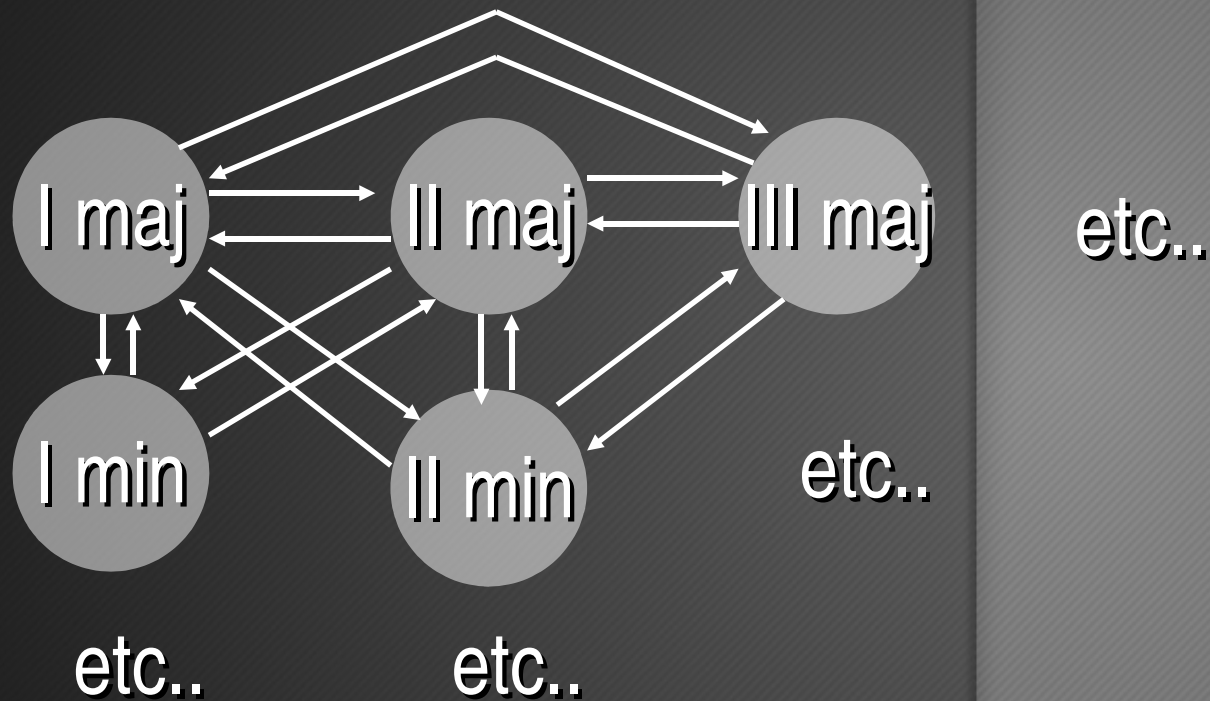
- Harmony: built an object implementing short term memory of played chords sequences [chords_memory]
- Learns the probability of chords transitions while you play
- Once trained can be asked for “normal” or “strange” chords sequences
- We can ask things like:
 - “we are in C major tonality, current chord D minor, where did I usually go from here?”
 - “build a walk 3 chords long from F major to A minor in C major tonality using rarely used chords sequences

[chords_memory]

- The memory is implemented using an oriented graph where the nodes are the possible chords in a given tonality and the arcs are the transitions from a chord to another
 - The arcs have weight: the probability of that transition in this style
 - The weights are set realtime each time a new chord is added to the memory

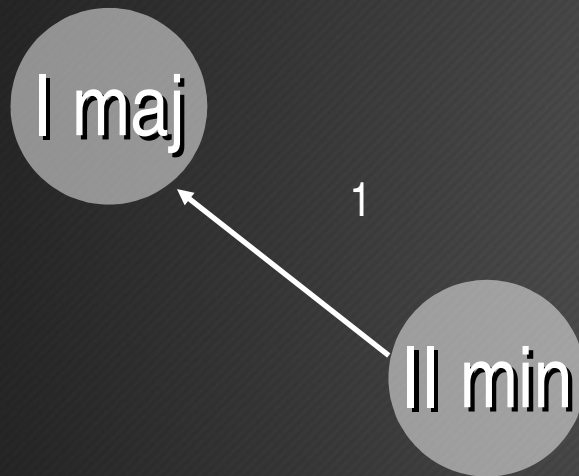
How it works (1)

Initially each arc has weight = 0 (transition never played)



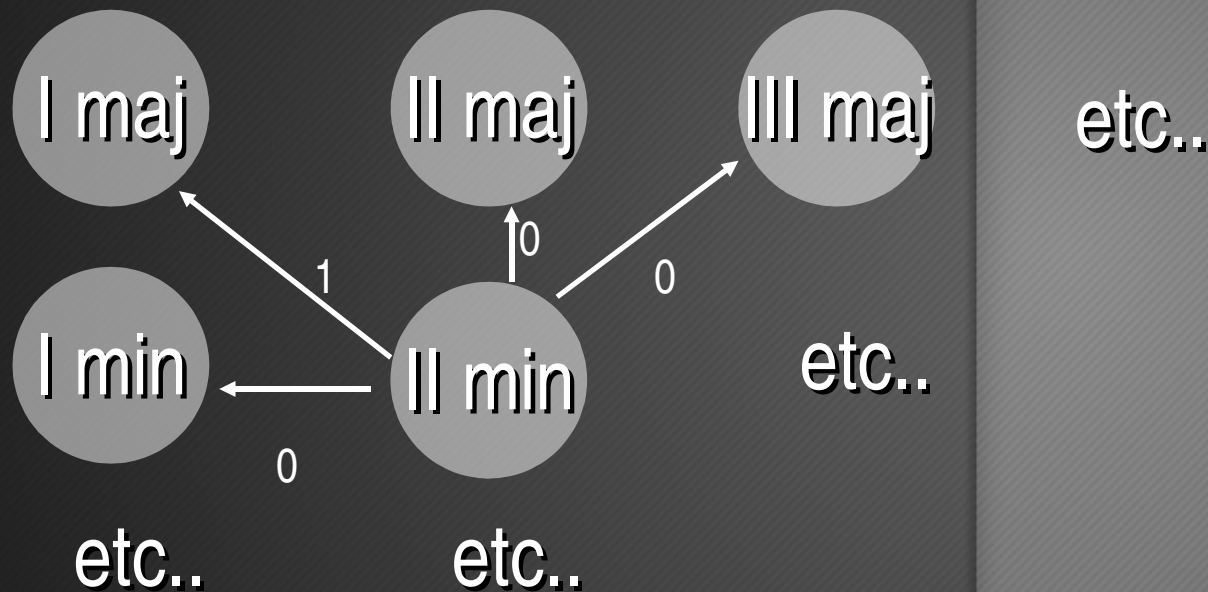
How it works (2)

Let's say we are in C major tonality, last chord is D min and we want to add C maj. First we translate them in relative names: II min and I maj, then we increment the arc's weight



How it works (3)

Now next time we'll be in II min we'll know that 1 time we used this transition: from II min to I maj



[chords_memory]

- there are 69 possible chords type
 - 11 possible tones
- the graph is implemented with a matrix of 11 x 69 x 11 x 69
- each cell stores the probability of the transition from chord x to chord y

[chords_memory]


- Current status: stable, usable and used:
- In an installation: at SMC05 (Salerno, November 2005, www.smc05.unisa.it)
- In a performance: at GA05 (Milano, December 2005, www.generativeart.com)
- In a performance at UEA (Norwich, February 2006, www.uea.ac.uk/mus)
- Needs improvements: modulations

step2: graphs

[chords_memory] was succesfull!

Then we tried to apply the same principle to a rhythm maker object
(then a melody maker object)

- GAs have no memory of the played rhythms (nor themes)
- Using graphs we can store enough informations to represent all the rhythmic (and thematic) material of a musical piece

GA  Graphs

[rhythms_memory]

The idea:

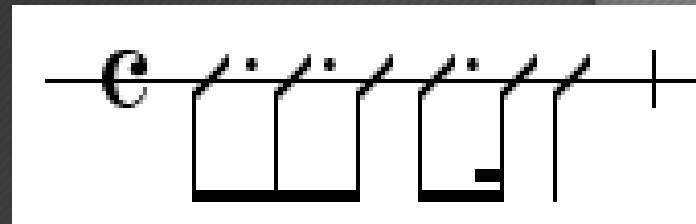
- Each note of a played rhythm is parsed into simple elements
 - Each rhythm is a linked list of simple elements
- Each time a rhythm is heard we match it with rhythms in memory:
 - If has some similarity then this is a variation
 - If has too few similarity then is a new rhythm

[rhythms_memory]

- This object is capable of doing a real-time rhythmic analysis of the played rhythms
- While you play it builds a memory of your rhythms and labels each rhythm with a tag (a1, a2, b1, etc..)

[rhythms_memory]: how?

Let's say you first play this rhythm



It can be expressed as a list of moments.. when each note starts (in musical notation):

1.0/1

2.3/16

3.3/8

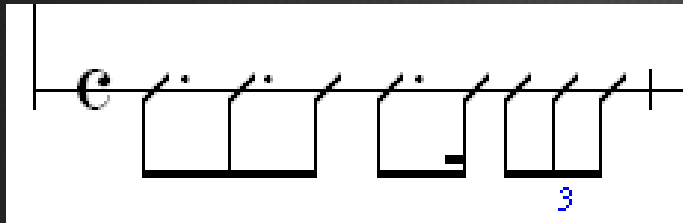
1.1/2

2.11/16

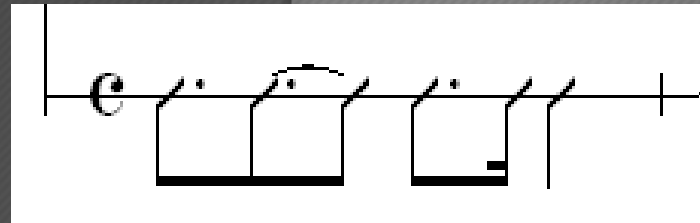
3.3/4

[rhythms_memory]: how?

Then you play 2 variations of the rhythm...



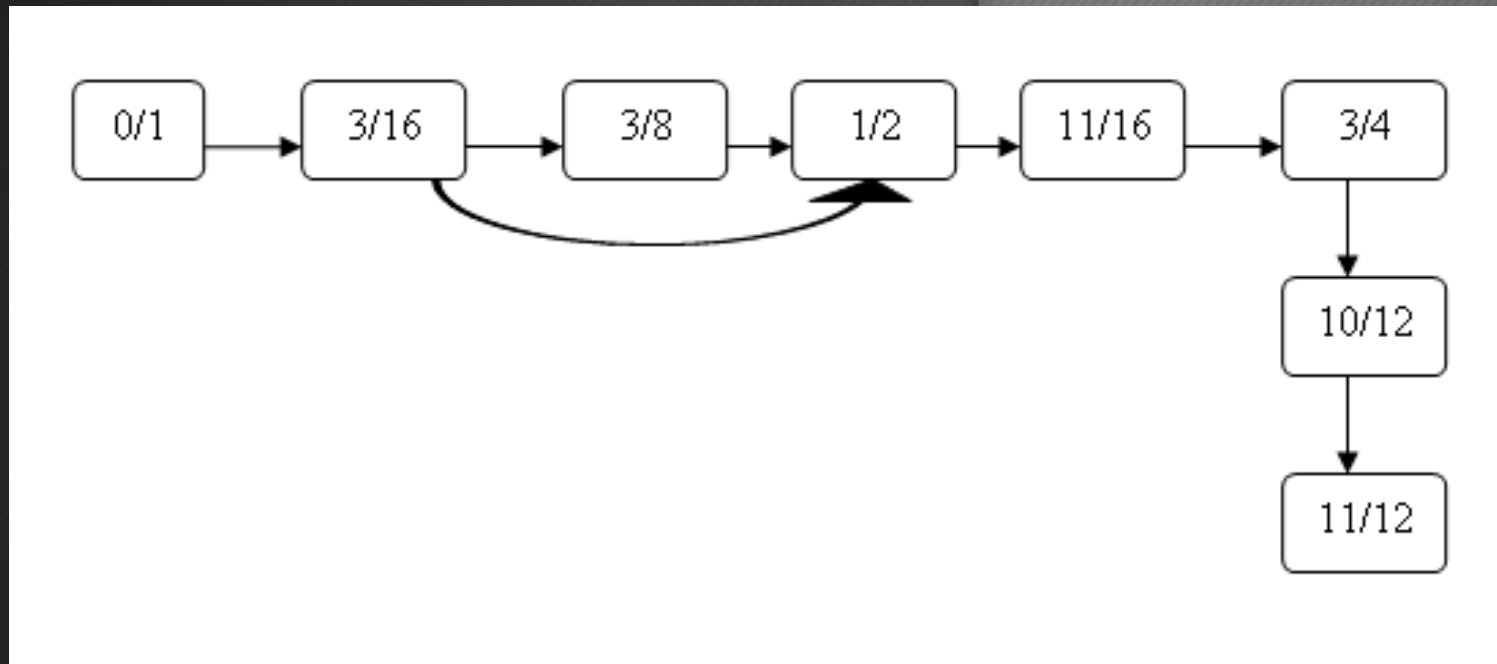
$0/1$, $3/16$, $3/8$, $1/2$,
 $11/16$, $3/4$, $10/12$,
 $11/12$



$0/1$, $3/16$, $1/2$, $11/16$,
 $3/4$

[rhythms_memory]: how?

Can be stored as a graph:

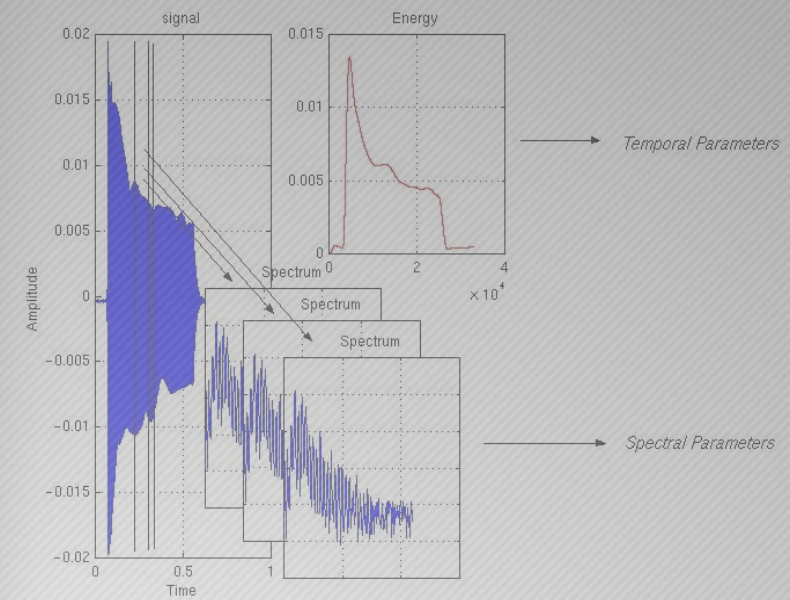


limits

- all these externals live in a “small world”:
 - MIDI
 - simple tempo (4/4)
- no reference to timbre
- concrete, electronic, microsound music impossible

mpeg7

- “Multimedia Content Description Interface”
- we use the audio part
- tools to extract audio/musical features
- we use:
 - AudioSpectrumBasis, AudioSpectrumProjection
- also interesting:
 - AudioSpectrumEnvelope, AudioSpectrumCentroid, AudioSpectrumSpread, AudioSpectrumFlatness, HarmonicSpectralCentroid, HarmonicSpectralDeviation, etc...



mpeg7

- its goal is classification
- possible applications:
 - you whistle a melody and the computer tells you the song name
 - speech detection
 - segmentation
 - automatic description and classification

soundspotter

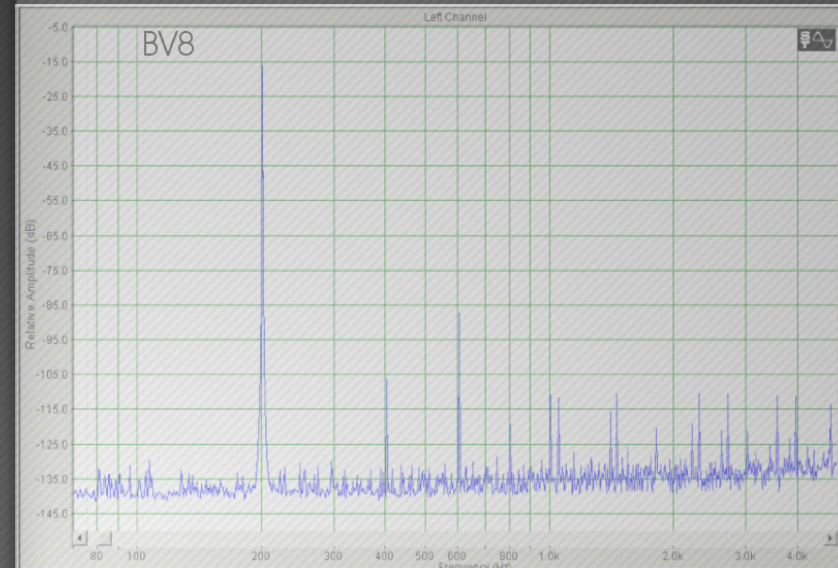
- we are using a pd object from M. Casey: [soundspotter2~]

what does it do?

- you give it a soundfile and it will analyze it
- now you can play an instrument and it will search the closest audio segment in its memory
- it sounds like it is following you

soundspotter: how?

- it segments the audio data in frames
- FFT is computed over each frame (8193 floats)
- mpeg7 data extracted from FFT (86 floats)
 - only the shape is kept
 - logarithmic frequency space (instead of linear) as Human Auditory System is closer to log than lin



soundspotter: how?

searching the database for similar content:

- extracts mpeg7 features from realtime incoming audio data
- for each frame in memory:
 - computes the distance
- the nearest is the most similar

soundspotter: usefull

David Casal used it in many concerts in (Belfast, Norwich, London)

he used soundfiles with orchestra and choir from Ligeti

he is a pianist



extending soundspotter

LAM asked us to extend soundspotter

- we should provide it with memory
- initially short term only
- we will apply same code from rhythms_memory (graph, transition matrix)
- ID, hashes or lexemes?

the easy way: ID

- exploiting mpeg7 tools implemented in soundspotter we get an estimation of the difference between a specific audio segment and the the segments in memory
 - if a segment is very close it is a repetition
 - if it is quite close it is a variation
 - if it is far it is a new segment

the easy way: ID

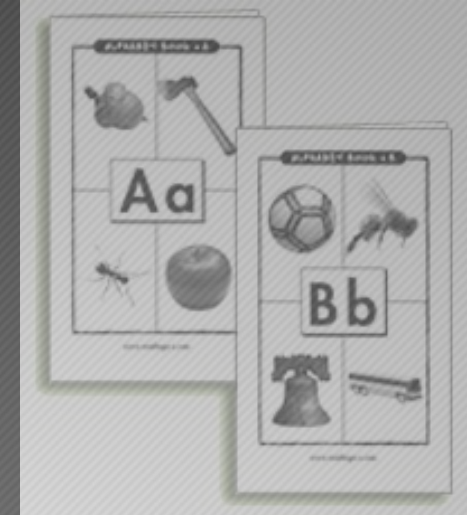
- every new segment has an unique identification number
- this way you can build a graph and a transition table

easy but

- each instance will have its id list
- impossible to build a shared memory

common alphabet

- more interesting is finding a way to share knowledge between instances
- robust audio hashes:
 - similar audio segments should have similar hashes
 - but how do we get back from hash to audio?
- lexemes:
 - create an alphabet using only the most common segments used



lexemes

- the audio stream is fragmented in frames
- each frame is matched against an alphabet
- frames changes over time
- the way they change is the sound fingerprint
- the alphabet is constructed with:
 - Hidden Markov Models
 - k-means

lexemes

- once we have an alphabet we can consider music as a language and apply NLP techniques
- we can create a shared long-term memory: a cultural context

tomorrow?

- all our efforts are about syntax
- what about semantic?
 - associations and emotions? we need a real cultural context for this!

conclusions

- applying AI techniques to music is hard because is hard to define what music is:
 - many different models to express music syntax
 - who can define its semantic?
- too subjective, too cultural dependent

frank chronicles

- summer 2005: first melody externals based on Todd's article
- fall 2005: co-evolutionary GA code applied to rhythm. first graph based externals
- winter 2005: SMC05 (Salerno), GA05 (Milano), LAM meeting (London). graph based rhythms analyzer and variatitor. Concert in Belfast, Norwich
- spring 2006: mpeg7. concert in London. NIME.

links

- <http://www.davidmorelli.it>
- <http://www.davidcasal.com>
- <http://cvs.sourceforge.net/viewcvs.py/pure-data/externals/frankenstein/>
- www.puredata.info
- Peter M. Todd: <http://www-abc.mpib-berlin.mpg.de/users/ptodd/>
- <http://mpeg7.doc.gold.ac.uk/>
- <http://www.livealgorithms.org>
- <http://www.generativeart.com>
- <http://www.smc05.unisa.it>

references

- co-evolutionary GA: “Frankensteinian Methods for Evolutionary Music Composition” Peter M. Todd & Gregory M. Werner
- lexemes: “Sound Classification and Similarity” M. Casey
- audio hashing: “Robust Audio Hashing for Content Identification” J. Haitsma, T. Kalker, J. Oostveen
- mpeg7: <http://en.wikipedia.org/wiki/Mpeg7>
- k-means: <http://en.wikipedia.org/wiki/K-means>
- HiddenMarkovModels: <http://en.wikipedia.org/wiki/HMM>