On Virtual Machine Architectures for Evolutionary Music Composition

[1]Babeș-Bolyai University, Cluj-Napoca, RO
csaba.sulyok@gmail.com

Csaba Sulyok[1], Christopher Harte[2]

[2]Scored Ltd. London, UK
chris@scored.film

Introduction

- *'All composers are algorithmic composers, not just the ones who profess it.'* Cope
- We model the thought process of a music composer as a Turing-complete virtual machine which outputs pieces of music based on their own existing work as well as the work of others.
- Our framework[1] uses linear genetic programming whose genotypes are programs written in an arbitrary sequential language instead of a functional one.
- We investigate the effect of virtual machine architecture and memory size on the quality of the results and performance of the algorithm.
- Our approach is deliberately general in nature making it not only suitable for music, but for many other types of complex data as well.

System Overview

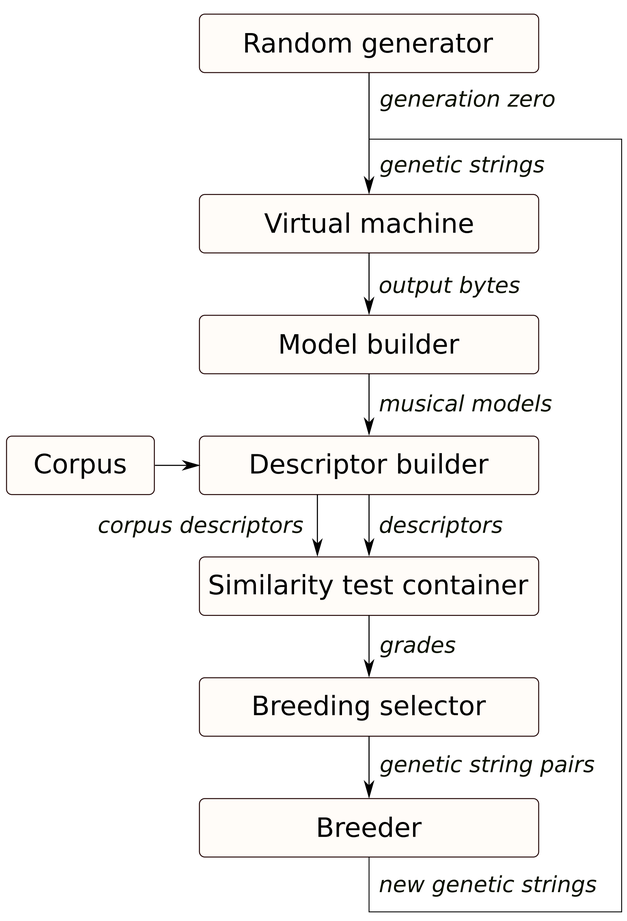


Figure: The workflow of our algorithm

- The framework follows linear genetic programming conventions;
- *Genotype:* a fixed size byte array representing the initial condition of the virtual machine;
- *Phenotype:* a piece of music, encoded similarly to MIDI;
- the virtual machine executes the genotype code and interprets the output as a musical model;
- *Fitness:* statistical similarity to a corpus of real music; in this case, a set of Bach keyboard exercises[2].

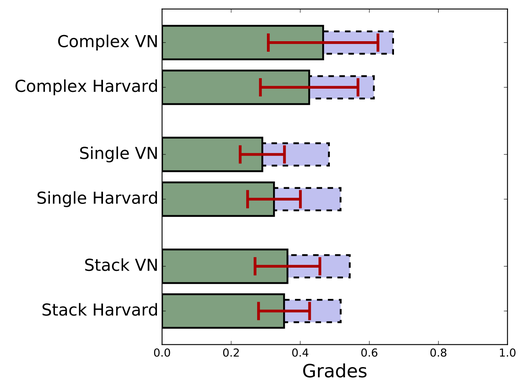
Virtual machine architectures

Our framework is able to simulate any real life Turing-complete processor. We compare the following different virtual machine architecture and design decisions[3]:

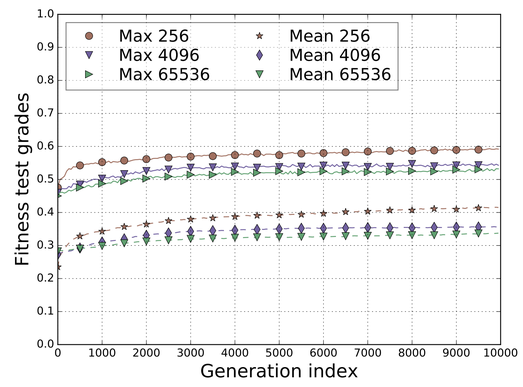
1. Von Neumann vs. Harvard architectures
   - ▷ Von Neumann machines use a common instruction and data space, allowing self-rewriting;
   - ▷ Harvard machines have separate read-only instruction space and data storage.
2. Register vs. stack-based machines
   - ▷ Register machines store local variables in a set of general-purpose or specialized registers;
   - ▷ Stack-based machines hold these variables in a fixed-size push-down stack, creating an arbitrarily large local data space.
3. Complex vs. minimal instruction sets
   - ▷ The complex instruction set assigns different instructions to all 256 values of the 8-bit instructions; these may become vulnerable to small changes;
   - ▷ We also test a modified single-instruction set: `SBNZ` - subtract and branch if not zero.
4. Large vs. small memory size
   - ▷ Our experiments use different memory sizes (256 bytes, 4KB or 64KB);
   - ▷ We also analyse how much memory is actually used in different runs.

Results

- 18 different configuration combinations
- 20 experiments each
- 10000 generations



(a) Final generation grouped by instruction set and architecture



(b) Grade progression grouped by memory size

Architecture	Memory size	Complex		SBNZ		Stack-based	
		Mean	*Max*	*Mean*	*Max*	*Mean*	*Max*
Von Neumann	**256**	0.483	0.673	0.325	0.515	0.381	0.563
	4096	0.469	0.667	0.263	0.475	0.342	0.516
	65536	0.446	0.666	0.282	0.459	0.366	0.551
Harvard	**256**	0.495	0.672	0.426	0.596	0.383	0.540
	4096	0.469	0.639	0.263	0.480	0.336	0.496
	65536	0.314	0.528	0.283	0.474	0.340	0.514

Table: Statistics of the final generation of our test runs

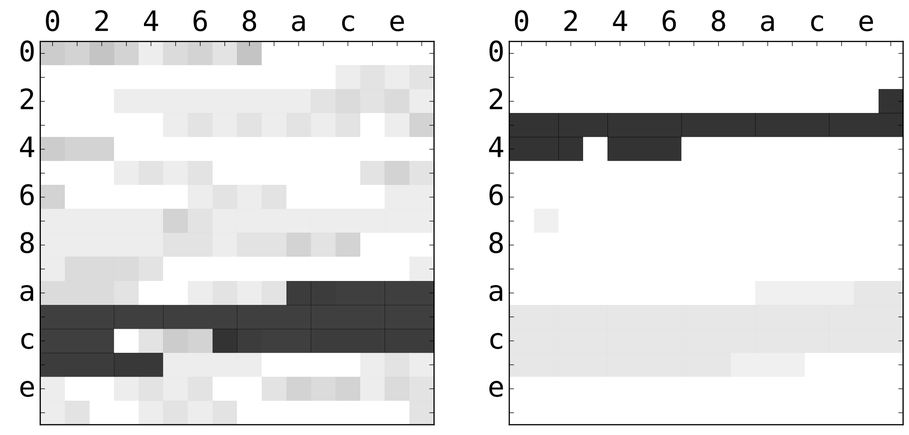


Figure: Heat map of random generation zero genotype vs. a highest scoring one; darker addresses have been touched more often.

Conclusions

- By architecture:
  - ▷ The von Neumann architecture scored slightly higher than the Harvard one in all cases except whe using the single-instruction set; this may be caused by its long instructions being more fragile to self-rewriting.
- By instruction set:
  - ▷ The complex instruction set gives the best results, but it is also vulnerable to change.
  - ▷ The single-instruction machine performed better than expected given its complexity.
  - ▷ The stack-based machine performed generally poorly.
- By memory size:
  - ▷ Using smaller memory sizes consistently produced higher grades, although this might show that the current tests do not properly reward complexity inherent in using more memory.
  - ▷ The size 256 mean values keep steadily rising even after thousands of generations; this is possibly due to the smaller percent of unused memory heightening the effects of mutation/crossover.
  - ▷ Results tend to favor smaller loops which ultimately create simple musical patterns.
- High-scoring MIDIs usually consist of a simple non-melodic pattern repeated throughout the piece.

[1] The project is hosted open-source at `https://bitbucket.com/csabasulyok/emc`
[2] MIDI corpus available at `www.midiworld.com/bach.html`
[3] Instruction set descriptions and MIDIs available at `https://csabasulyok.bitbucket.io/emc`