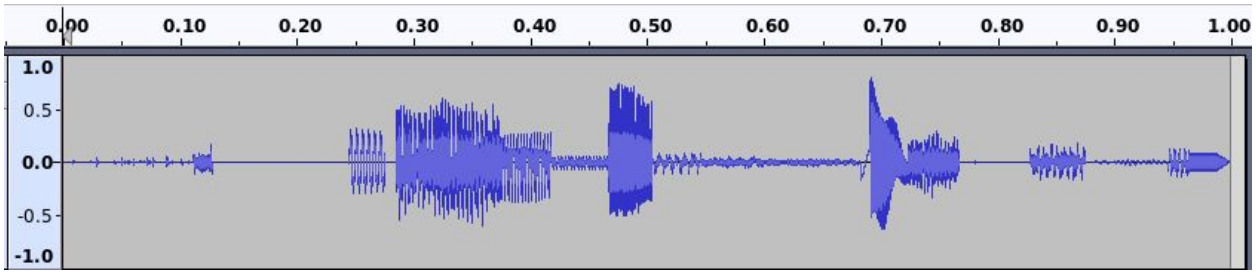
Charles Bolton, Willis Hoke, Patrick Rademacher
Artificial Intelligence

Euphonomimesis: Genetic Algorithms for Sound Generation

" ... a tangible duration that could be cut up ad infinitum, up to one hundred parts per second, allowing the realization of abstract rhythmic speculations ... " [1]



Above: Time-domain waveform generated after 5 generations

Euphonomimesis is an experiment in the application of a genetic algorithm to audio samples in the time and frequency domains. The name came about as a portmanteau of *euphonic* ("pleasant sounding") and *mimesis* ("imitation of the real"). We developed this project based on our interest in applying AI/ML algorithms to digital audio for creative purposes.

There are many audio datasets available for free online which have been used in numerous machine learning experiments, whose applications include natural language processing, algorithmic music generation, and sound classification. One very similar example was a project called Timbre Mimicking[2]. However, this experiment differs in approach from our experiment, as it was focused more specifically on spectral/timbral information to produce mimicry, as opposed to ours, which experimented with domain-based comparisons. However, both projects share similar goals and are sufficiently kindred.

[1] Cited in *Microsound*, Curtis Roads, MIT Press, 2001
[2] https://ccrma.stanford.edu/~lja/timbreMimicking/

Wanting to investigate the abilities of genetic algorithms in more depth, we invented an interesting problem involving digital signal processing. We decided to evaluate the performance of GAs on audio files in both the time and frequency domains. Precisely, we were intrigued by the problem of audio mimicry, and concerned ourselves with whether or not a GA could, given some source audio clip (called the "goal clip") and a dataset of 8,657 other clips (not including the goal), produce a clip that "sounded" the same as the goal clip. Here the non-technical term "sounded" was evaluated by simply listening to the output and comparing it to the goal clip, but we also included a more rigorous "fitness score" evaluation, discussed below.

Two datasets comprise the inputs to our program. Module **fft.py** takes as input a set of 351 *goal_clips* and another set of 8657 *clips*. The first set contains real instruments playing a single sustained note for a single second. These clips, generated using Logic X, are all the "same" in that each is a 1 second, mono, clip with 44100 16-bit samples—all clips have the same MIDI values in regards to both pitch and velocity. Likewise, each clip in the second set uses the same specifications. Although we used the above specs, our program can accept .wav files of any length. The decision to use clips to the above specs was made after it was determined by earlier trials that the algorithm would run quite slowly with larger files. The second set of clips uses modified clips from the FSDKaggle2018[3] dataset, "an audio dataset containing 11,073 audio files annotated with 41 labels of the AudioSet Ontology." We wanted all the clips to be 1 second long, so we needed to process the clips in Logic X before we used them, since they were of varying size. It has many varied sounds from real instruments, urban sound, voices, farts and burps, etc. We culled 8,657 clips from this set, bringing the total number of input clips to the program to over 10,000.

Once the data is loaded, the program will generate a single random *goal_clip* from the *goal_clips* set. For the initial population, the program will take this clip

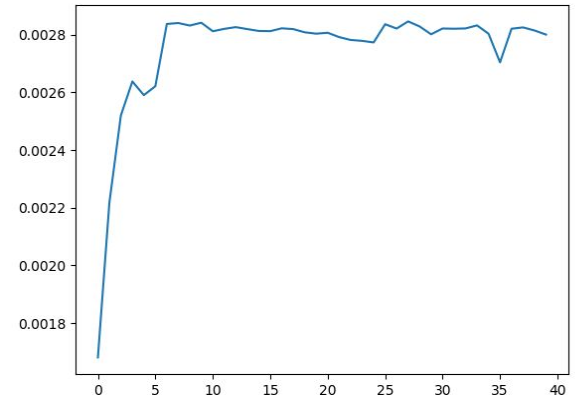
[3] https://zenodo.org/record/2552860#.XIpVoIU8twF

plus all the other clips in the *clips* set. Next, will extract all of the samples and the 16-bit sample values from each clip and optionally run all these clips through a Fast Fourier Transform. In the former case, the GA is working with 8,657 arrays with time-domain information, and in the latter, the same arrays with real-valued frequency-domain information. Each array contains 44,100 values, dependent on the domain choice. These encodings of each "member" comprise the initial population, which is subsequently fed to the *breed_loop*, which iteratively crosses these arrays a number of times, according to the hyperparameter *N*. After a number of trials we determined $N = 1000$ to be a sufficient number of iterations, since the fitness score after this point did not change in a measurably significant way.

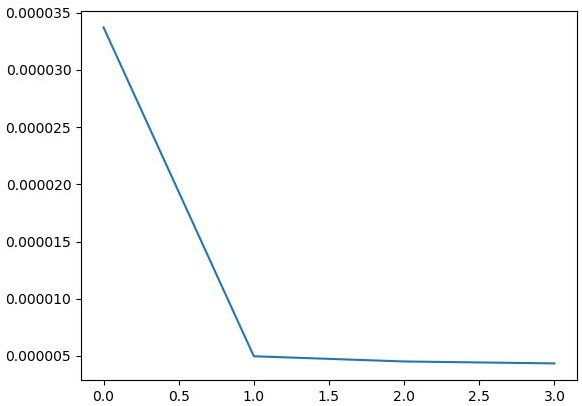
Each of these iterations performs the evolutionary procedure of the algorithm half-the-population times by selecting two members of the population to cross. The crossed gene-arrays produce two child arrays with the spliced (randomly chosen) time/frequency domain information from each gene-parent. The two parent clips are chosen using a probability proportion selection based on the fitness scores of all the clips in the current population. The two children are added to a new population, which overtakes the initial population following a single iteration. The fitness score is calculated by taking the L2 norm of each clip compared with the goal clip. The maximum score is thus 0 (indicating the two sounds are identical). The output of the program is a 1-second audio clip every *save_interval* iterations, as well as a visual plot tracking the fitness score, displayed above.

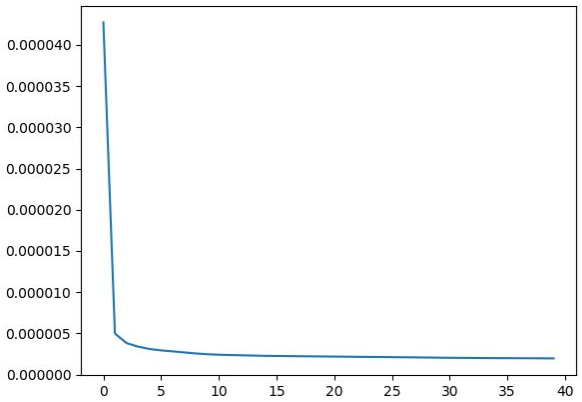
In addition to crossover and fitness evaluation, our algorithm also included a kind of novel approach to mutation using an idea borrowed from simulated annealing. As the algorithm began and the number of iterations low, the probability of mutation was near certain, only to cool to near impossible as the number of iterations approached *N*. Despite several experiments changing mutation parameters and the temperature schedule, however, it was determined

that the stochasticity introduced by even a fairly heavy-handed mutation protocol was insufficient to reach a goal state within a realistic amount of time/iterations.

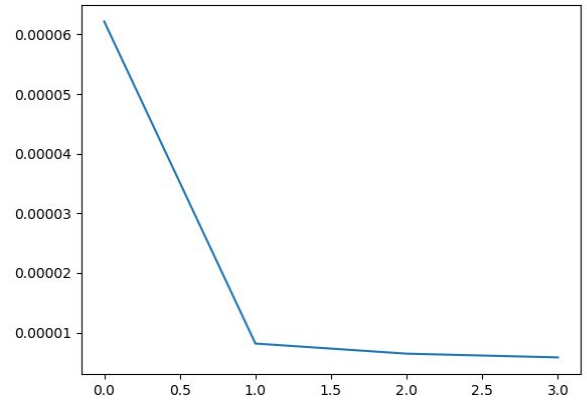


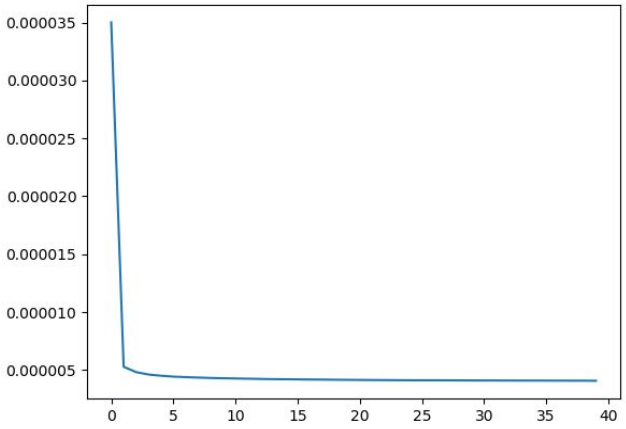
Above: using only goal_clips, we observed that the fitness score converged to 1/350, where 350 was the number of clips in the goal set.



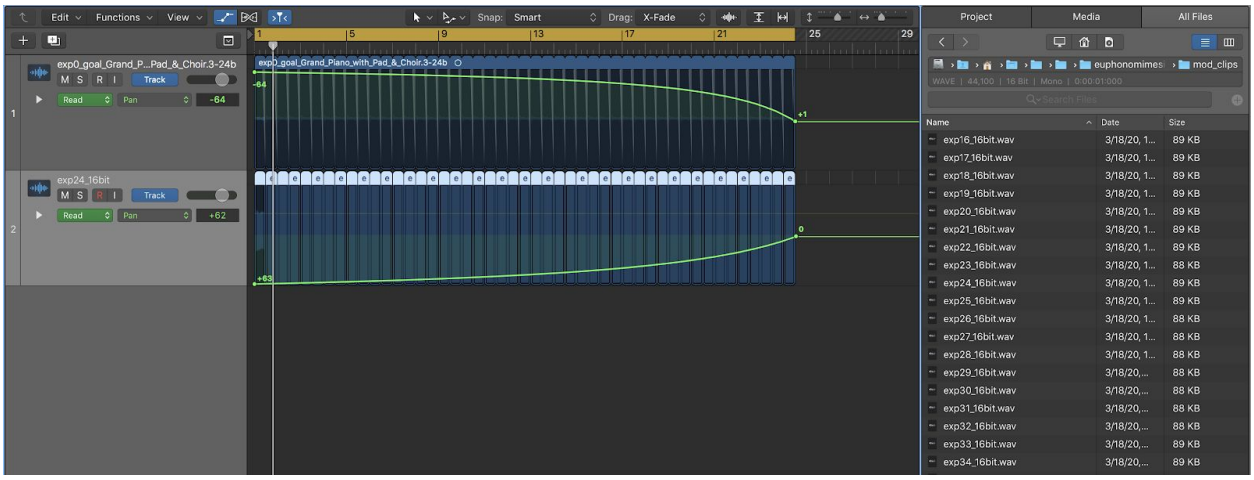


Above: every 25th minimum fitness score over 100 and 1000 (respectively) iterations in the frequency domain





Above: every 25th minimum fitness score over 100 and 1000 (respectively) iterations in the time domain



Above: post-production of successive output audio files vs the goal clip

While our algorithm was designed to work with audio clips, it could easily be adapted to work with other kinds of data. For example, it would be possible to pass in flattened images or textual data with a minimum amount of reconfiguration. Unfortunately, due to the size of the audio files we used, performance was quite slow. With an initial population of 8000, it took several seconds to compute each successive generation. In addition to the slow performance, we also did not have as wide or vast of a population that would be ideal, or perhaps did not reintroduce original members of the population at particular instances, to reach the goal state. To put it simply, this task was more complicated than we initially thought it would be. In contrast to a problem like the well-known 8 queens problem using genetic algorithms, this particular problem instance contains encodings that are much more variable, a more intricate and complex system, and ultimately a greater amount of elements to keep track of and manipulate. Given sufficient computational resources and more time to experiment with mutation and crossover algorithms, we believe it would be possible to eventually reach a goal state. We also concluded that 8,000 clips provided an insufficient amount of genetic material in the form of encodings for the algorithm to properly converge. A more appropriate dataset would likely necessitate closer to 1 million clips. At this point the complexity of the procedure becomes unfeasible.

Despite running for as many as 10,000 generations, this goal was ultimately never met. Although the output audio never “sounded” the same as the target clip, some interesting timbral qualities were observed. Even though we never meet our goal or destination, we can hear the child from generation to generation “making its way” closer and closer to the goal. In the frequency domain, we observed odd spectral components, reversals of envelopes, and admixtures of identifiable sounds. Aesthetically, the results from applying the genetic algorithm to signals in the time domain were more interesting, sometimes producing results reminiscent of *musique concrete*. Abrupt cuts, glitches, and granular effects were observed. In

the time domain, a different crossover algorithm might have been more effective. In particular, crossfading between two parent sounds would have resulted in less jarring transitions between different timbral elements.

Nonetheless, we are still satisfied and content with our results, despite that our output was not what we originally envisioned. The important lesson learned is that applying techniques and practices of genetic algorithms to a very large problem leads to less predictable results. It turns out that implementing vectorized operations on large matrices, performing transformations between frequency and time domains, and working across a wide range of frequencies (and putting that wide range in two different spaces), made the overall task at hand harder to work with than other projects and scenarios which GAs are more adept at solving. Regardless, the fact we still saw results that met our predictions—in one case a population that nearly converged—demonstrates that we were on the right track. We had reasonable doubts throughout the process, but when we finally listened to the output of so many generations stacked up against one another as one complete audio file, its slow yet steady convergence to the goal state was a celebratory moment.