Interactive Evolution for Melody Generation

Noemi Canovi, Roberto Mazzaro, Michele Presutto, Olga Zaghen
University of Trento

***Abstract*—An interactive evolutionary system for the creation of short human-pleasing musical compositions is presented in this paper. This is achieved through a genetic algorithm (GA): a population of melodic and rhythmic lines is evolved dependently on the evaluation of a human user. Although naïve, the system demonstrates to produce meaningful and interesting results.**

***Index Terms*—genetic algorithm, evolutionary music, IEC, music, music melody generation**

I. Introduction: Evolutionary Music

Evolutionary Computation includes a broad set of computational techniques aimed at replicating natural evolution processes. Being general-purpose, it has been applied to many different fields, one of them being music.

The application of EC methods to musical tasks began to emerge in the early 1990s, and along the years lots of studies have been published. Each study involves the application of either Genetic Algorithms (GA), Genetic Programming (GP), Grammatical Evolution (GE) or a variation of these. Even if differently, all of them have to take into consideration three fundamental points: problem domain, individual representation and fitness measure.

The problem domain defines what type of music the application has to evolve, e.g., melodies or chord progression, and what genre, e.g., jazz or rock. Individual representation determines what the genome represents, if either pitch values, duration or other properties. Finally, fitness defines if and how a piece of music is better than another.

Regardless of the adopted representation, music evaluation is most often a subjective task and, as such, a reliable and robust fitness measure can be difficult to define. Many researchers circumvented this issue by employing the use of a human judge as the fitness function; such systems are known as Interactive EC (IEC).

The well-known GenJam [3] is one of these systems: it uses a GA to evolve jazz solos that will be graded by the user in real time as either "good" or "bad". Over the previous two decades, GenJam has been modified many times and has evolved into a real-time, MIDI-based, interactive improvisation system that is regularly used in live performances in mainstream venues. In [1] another IEC application can be seen, where simple melodies are generated and the user has control over the fitness via interface. Although naïve, this application demonstrated that nice melodies can be created and some interesting ideas, for instance composing the initial population by notes distributed according to Zipf's law, have been applied. However, it is limited by many constraints, like the confinement of the pitches to a single octave interval and the fixed rhythm.

II. Methodology

For this specific system we implemented a genetic algorithm (GA), a methodology that borrows concepts from natural evolutionary theory. GAs are characterized by a population of individuals that evolve over time through specific operators. Each individual is characterized by a fitness function that influences its probability to produce offspring.

The core point of the project is to produce melodies having almost no constraints in the search space, and the evolved individuals are the melodic and rhythmic lines themselves. In addition to this, the fitness that drives the evolution corresponds to the evaluations of a human supervisor, so our system can be classified as an IEC.

In this section a general layout of the algorithm is presented, followed by a more detailed description. The difficulties and the results are shown respectively in sections III and IV. The contributions of the members can be found in section V and the possible improvements are listed in section VI.

A. Main Steps

A schematic representation of our systems can be seen in Fig. 1.

In the beginning a simple interface is shown to the user, which allows to choose how many melodies will be created at every generation and the number of octaves the notes will range over. Then the evolution of the melodies starts and a new population is created through some random functions. We decided to differentiate the melody and the rhythm evolution in order to make the steps clearer and reduce the user's confusion. Once a population is generated, the respective tracks are composed and put at disposal of the user via interface. After listening to all or at least some of them, the human evaluator rates the fitness of the melodies and the individuals go through crossover and/or mutation. A new population is formed and converted again to tracks available for the user's evaluation. The whole process is repeated until the user is satisfied. At this point, the user can decide to quit or to proceed with the rhythm evolution.

In the latter, the user must choose a melody to start with and set the number of different rhythms to rate at each generation. The evolution process is repeated as before: crossover and mutation are again applied to form the new population. Once the evaluator is satisfied the algorithm definitively ends and the final melody with the rhythm of his choice (the best one) can be downloaded.

B. Melody and Rhythm Encoding

One of the main issues in GA is how to encode problem specific information into genes. Our system aims at evolving

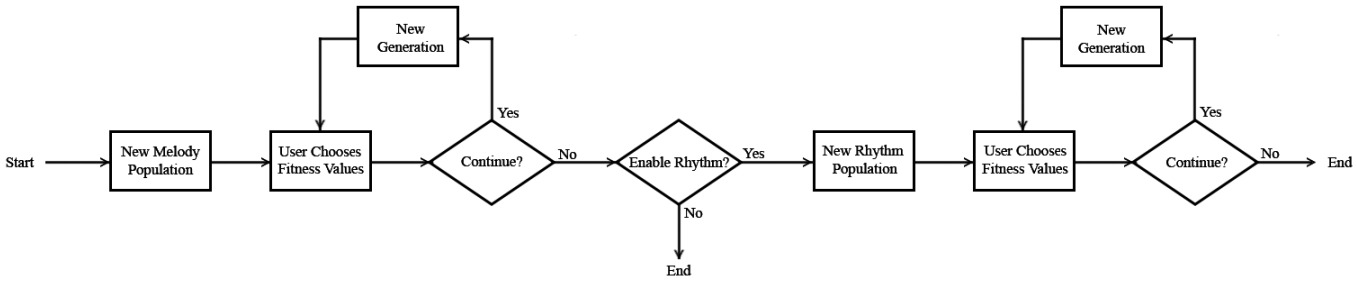

Fig. 1: Main steps of the system

a population of melodies: each individual is a sequence of 8 notes. This length was chosen in order to make the listening and evolutionary processes feasible and not overburden the user. Each note of a melody is characterized by specific pitch and rhythm values, that we expressed through some parameters, making some simplifications we have: *value*, *octave* and *rhythm*. In particular, *value* and *octave* allow to encode the pitch of a note. In order to put in place this structure properly we implemented specific classes: `Note`, which encodes a single note, and `NoteList`, that encodes a melody as a list of `Note` instances and some additional elements corresponding to parameters such as the fitness of the individual.

The main attributes that characterize a `Note` class are the following:

1) Octave: For what concerns the *octave* parameter, it assumes integer values that range from 3 to 5. Indeed, we focused on the three central octaves of a common piano, which means the 3rd, the 4th and the 5th octave. The octave value is generated through a Gaussian distribution centered in 4, by following the principle that the majority of the notes of a melody in Western music belongs to the 4th octave. This implies that less notes fall in the 3rd or 5th octave with respect to the 4th.

2) Value: The *value* parameter, instead, can assume the integer values that range from 1 to 7, because we chose to consider only the main notes without alterations for simplicity: C, D, E, F, G, A, B. The *value* of a note is first assigned by sampling from a Zipf's distribution:

$$f(k, N, s) = \frac{1/k^s}{\sum_{n=1}^{N}(1/n)^s}$$

where N is the number of notes, k is the rank of each note, s is the value of the exponent characterizing the distribution and f is the relative frequency of occurrence of each item ranked by k. This power-law has proven to be a valuable measure for the aesthetic evaluation of music in various aspects.

We followed the idea that in Western music some notes are more frequent than others, like the 1st, 5th and 3rd grades of the scale, and we wanted them to appear more frequently than others in the initial melodies in order to have a good starting point. Indeed, we also tried initializing the note *value* by randomly sampling from a uniform distribution but it was harder to obtain good individuals in few generations in this case, due to worse melodies in the initial population.

3) Rhythm: For what concerns the *rhythm* parameter, it assumes integer values that range from 2 to 4. We implemented a simplified concept of rhythm, that allows to maintain the structure of a melody as a list of `Note` and its attributes and also forces all the melodies to have a total fixed duration. We assume to work with musical compositions of two $\frac{4}{4}$ measures, each one containing four different notes of duration $\frac{1}{4}$ during the entire melody evolution process. Changing the rhythm of a note means splitting the $\frac{1}{4}$ note in more repetitions of less duration, and the *rhythm* value stores the power of $\frac{1}{2}$ corresponding to such duration. The possible values are:

- 2, that corresponds to 1 repetition of the note value, of duration $\frac{1}{4} = (\frac{1}{2})^2$
- 3, that corresponds to 2 repetitions of the note value, of duration $\frac{1}{8} = (\frac{1}{2})^3$
- 4, that corresponds to 4 repetitions of the note value, of duration $\frac{1}{16} = (\frac{1}{2})^4$

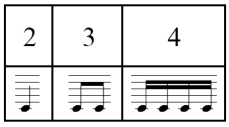

Fig. 2: Durations corresponding to the *rhythm* value

If the user decides to evolve the rhythm, the *rhythm* values are re-initialized by sampling randomly from half a Gaussian distribution centered in 2, so that the most frequent duration is $\frac{1}{4}$. We didn't encode duration values higher than $\frac{1}{4}$, such as $\frac{2}{4}$, in order not to incur into a too complex implementation of the note and genetic operators and we neither considered triplets or rhythm alterations.

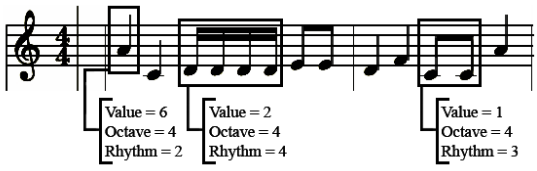

Fig. 3: Example melody as a sequence of 8 notes with different durations, with the explanatory values of the corresponding parameters of some notes

C. Sounds and Tracks Creation

For this project we created a dataset composed of 63 audio files, each file named as `genetic_melody_V_O_R` (e.g. `genetic_melody_1_3_2`), where `V` stands for the value of the note, `O` is the octave and `R` is the variable referring to the rhythm. In order to generate the audio files an electric piano was used, and each audio file has been recorded at 104 bpm for a length of $\frac{1}{4}$ of a musical beat.

The audio files corresponding to each individual in the population are made using the function `create_audio(pop)`. It sequentially concatenates the single notes to create the melody and save it in the `audio` folder as `indiv_n`, where `n` stands for the n$^{\text{th}}$ individual of that population.

D. Genetic Algorithm

1) Subjective fitness and user interaction: The specificity of our project is the centrality of the user interaction, hence their influence, in the evolutionary computation. The main difference from the standard and classic genetic algorithms is the subjectivity of the fitness function, which is completely defined by the user response. In order to evaluate the fitness of an individual, its genotype is converted to its respective phenotype so that the user can listen to the melody and rate it with a score from 1 to 10, according to their personal taste. In this way the fitness function drives the evolution of the melodies in a fashion that completely depends on the current human user. The choice of allowing scores to span from 1 to 10 was made in order to grant detailed rates and make the algorithm better adapt to the evaluator's tastes.

The fitness value of each individual is stored in a proper attribute of the `NoteList` class, with default value equal to 1. This choice was made to properly handle the plausible scenario in which the user decides not to rate (or even listen to) some individuals. The unattended tracks will have very little impact in the creation of the new population, with respect to the ones evaluated by the user.

2) Algorithmic choices: Designing the system required adapting in specific ways the hyperparameters of the evolutionary process, such as the ones related to elitism, crossover and mutation, in order to deal with the limitations (in terms of evaluation time and concentration) given by the human interaction and still achieving satisfactory results.

- **Population size and number of generations.** The number of generations and the population size in each run are quite low, differently from the classical setting of genetic algorithms (in which they are hundreds or thousands). In order to deal with the human fatigue of the user, indeed, the size of the melodies population is customizable and can be set to a value from 1 to 10.
  For what concerns the evolution of the rhythm, instead, the population size is again set by the user, choosing in this case from a set of values that range from 1 to 5. The reason for which the rhythm population is constrained to be smaller is that evaluating multiple rhythms for the same melody is likely to be less intriguing and interesting for the human supervisor with respect to trying out completely different melodies. This happens because the population for the rhythm evolution is obtained by starting from the best melody selected in the previous step and generating a new rhythmic line for each individual.
  The number of generations is in both cases a parameter that depends on the user: once they are satisfied, they may terminate the evolutionary process.
- **Parent selection and replacement.** We chose to perform a fitness-proportionate parent selection through a roulette wheel, in order to bias the selection towards the fittest individuals and promote exploitation: this is particularly helpful because the number of generations is generally low, and during the first ones it may be difficult to randomly generate melodies that suit the user's taste. For what concerns the survivor selection, instead, we decided to entirely replace the old population by offspring, except for a certain number of elite individuals. The presence of elites allows to prevent the loss of best individuals across generations, and their quantity depends on the population size: there are none if the population size is 1, there is 1 elite if the population size is between 2 and 7, and there are 2 otherwise. This holds for both melody and rhythm evolution.
- **Crossover and mutation.** The way in which the evolution proceeds across generations mainly depends on the recombination and mutation operators. When a new offspring has to be generated, it is characterized by a certain probability of crossover: a random real value between 0 and 1 is sampled, and if this is lower than the probability threshold, a one-point crossover is performed between two selected parents in a random point of the `NoteList`.
  After the crossover a mutation step may be applied: each note of the `NoteList` can be mutated with a certain probability, which is the same for all notes. The mutation of the *value* attribute is performed independently for each note in the list by sampling an integer from a normal distribution centered in the current note pitch, obtained as $i \cdot 7 + j$, where i is the *octave* and j is the note *value* (in Fig. 4).
  In the rhythm case, instead, the Gaussian used for sampling is always centered in 2, in order to promote notes with long duration and avoid producing long sequences of fast notes that are not pleasing to the ear.

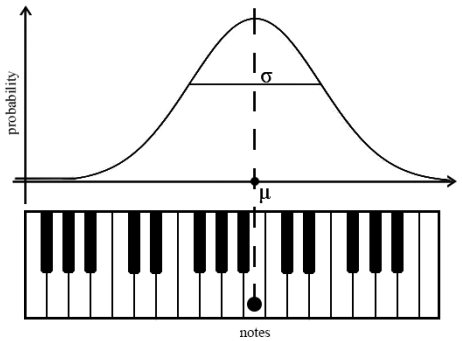

Fig. 4: Visualization of the mutation operator for the value

In order to set the parameters we performed some empirical tests and we noticed that too high mutation and crossover rates make the new generation differ too much from the previous one, while too low ones don't allow enough variability. For this reason we set dynamically decreasing values for these probabilities in the melody evolution: they decay linearly to a final stable value in 5 generations (from 0.7 to 0.3 for the crossover, from 0.6 to 0.2 for the mutation). This approach promotes exploration in the beginning while it favors exploitation after some iterations, where just little refinements to the melodies are needed.

In the rhythm case, instead, these operators are less impactful on the melody's objective enjoyability, hence we set the recombination and mutation probabilities to constant values (0.5 and 0.4 respectively).

Due to the relatively low values for the mutation and crossover rates, we also implemented a control in order to avoid generating duplicates in the current generation.

III. Encountered Difficulties

In this section the issues and the adversities emerged during the development of the system are presented.

Subjective and relative fitness. Every individual of a population should be evaluated independently of the others, with an absolute score from 1 (unpleasing) to 10 (masterpiece). People may tend, instead, to rate a melody dependently on the other scores assigned in the same generation. This may cause for instance the same elite individual to assume different values in consecutive generations, which should not be the case.

Fitness bottleneck. All the tracks must be listened to and evaluated by a human user, and this requires a high cost in terms of fatigue and time. In a musical system this is even more problematic than a visual system, where images can be viewed quickly.

Implementation of the rhythm. One of the main difficulties encountered in this project was the implementation of some form of rhythm while maintaining a fixed length and duration of the melodies. We solved this issue by simplifying the rhythm concept, embedding it in the `Note` class as a value expressing the duration and the repetition of the note itself.

Hyperparameters difficult to set. The definition of each hyperparameter in the GA must suit the presence of a human evaluator. In order not to overbear the user, while not constraining the search space, we carried out specific implementation choices after many empirical tests. This holds for instance for the population size, the elitism and the dynamic mutation and crossover rates.

High influence of the first generation. We have noticed that the first generation has a high impact on the fitness trend in the evolutionary process: if it contains at least one "good" individual, then the convergence is generally faster. In the other case, the search for a better individual becomes difficult also with multiple generations, since the population seems to stagnate as a set of "bad" individuals.

IV. Statistics and Results

In order to evaluate the effectiveness of our system, we ran the code and acquired the statistics from 40 different executions. In particular we stored in a `.txt` file the average score and maximum score assigned by the user in each generation.

First of all, the average number of generations across all executions resulted to be 4.96 for the melody and 1.67 for the rhythm evolution. This constitutes a very fast convergence rate for a GA and an execution of this length takes approximately 10-15 minutes of runtime. Moreover, the results show that the rhythm is generally not subject to a proper evolution, since the user tends to be satisfied with the solutions belonging to the first or second generation.

For what concerns the analysis of the fitness trend, people tend to have dissimilar standards about the "goodness" of a musical piece, so we resolved in computing the mean values of the acquired statistics over all executions. The study reveals that the mean of the average scores grows from 3.75 in the first generation to 5.10 in the last, with an average total gain (between first and last) of 1.46, and of 0.35 between two consecutive generations. The maximal score, instead, assumes an average value of 5.67 in the first generation, that goes up to 7.30 in the last one.

Moreover, the last generation considered for the computation of the mean values is the penultimate one: once the evaluators are satisfied with a track they tend to select it and end the algorithm without rating the other ones. This means that the last population usually has lots of scores equal to 1, the default value, and this may mislead the results.

	First gen	Last gen
average fitness (mean)	3.75	5.10
maximal fitness (mean)	5.67	7.30

The tendency to a substantial positive gain in the average of the scores shows that the algorithm works well in adapting the melody evolution to the taste and the subjective evaluation of the human user.

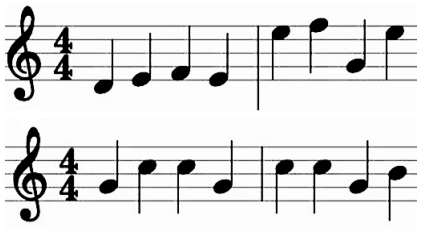

Fig. 5: Example of two melodies created by the GA. They have been generated in the same evolutionary process and they are objectively musically coherent, in addition to complementing each other well

V. Contributions

The main contributions of each member are:

- Michele set up the initial framework and he was responsible for generating the sounds and creating the tracks

- Roberto focused on the GA, implementing the classes used to encode the individuals, the population and the genetic operators
- Olga and Noemi constructed the interactive interface and the statistics. Moreover, they linked the different parts of the project together and did most of the debugging.

In general, from the beginning we settled on making all the main decisions on the system together (e.g. the main features of the genetic operators and the structure of the genotype). Note that, even if each of us concentrated on a different part of the project, we were rather dynamic and collaborative in searching for possible improvements and bugs in the whole system. Additionally, we met often in order to ensure every member was up to date with the project implementation process.

VI. Conclusions and Possible Improvements

We implemented an interactive evolutionary computation system which provides rather good results. We can establish that, even if naïve, the system works well, producing nice melodies in a restricted number of iterations. During the design of the framework we laid out a fair amount of possible ideas, but some of them were discarded to make the project feasible and self-contained.

Possible improvements can be obtained by relaxing some assumptions imposed in the note and rhythm encoding, for instance not just splitting the same note, but changing its total duration. Also, adding more constraints can be beneficial, such as note freezing or starting from an initial melody already decided by the user. A more abstract idea may be to build a prediction system based on the frequency and context of the notes in a given dataset of melodies, for instance, belonging to a certain musical genre. The prediction model could be used as a starting point to define an objective fitness function.

References

[1] V. M. Marques, C. Reis, J. A. Tenreiro Machado. Interactive Evolutionary Computation in Music. IEEE International Conference on Systems, Man and Cybernetics. 2010.

[2] R. Loughran, M. O'Neill. Evolutionary Music: applying evolutionary computation to the art of creating music. Genetic Programming and Evolvable Machines, Vol. 21, Issue 1-2, pp. 55–85. 2020.

[3] J. Bile. GenJam: a genetic algorithm for generating jazz solos. Proceedings of the International Computer Music Conference. pp. 131–131. 1994.