A Genetic Algorithm for Generating Improvised Music

Ender Özcan, Türker Erçal

Yeditepe University,
Department of Computer Engineering,
34755 Kadıköy/İstanbul, Turkey
Email: {eozcan|tercal}@cse.yeditepe.edu.tr

Abstract. Genetic art is a recent art form generated by computers based on the genetic algorithms (GAs). In this paper, the components of a GA embedded into a genetic art tool named AMUSE are introduced. AMUSE is used to generate improvised melodies over a musical piece given a harmonic context. Population of melodies is evolved towards a *better* musical form based on a fitness function that evaluates ten different melodic and rhythmic features. Performance analysis of the GA based on a public evaluation shows that the objectives used by the fitness function are assembled properly and it is a successful artificial intelligence application.

1 Introduction

Evolutionary art allows the artists to generate complex computer artwork without a need to delve into the actual programming used. This can be provided by creative evolutionary systems. Such systems are used to aid the creativity of users by helping them to explore the ideas generated during the evolution, or to provide users new ideas and concepts by generating innovative solutions to problems previously thought to be only solvable by creative people [1]. In these systems, the role of a computer is to offer choices and a human is to select one. This can be achieved in two ways. The choices of the human(s) can be used interactively or can be specified before and get autonomous results without interaction with the computer. The latter art form is also referred as the *genetic art*. In any case a human specifies some criteria and the computers are used for their capacity to investigate large search spaces. Composing a musical piece involves many stages, but surely, search forms a substantial part. Searching the right notes and durations can be an example. The existence of such a search process might require a pruning process to throw out the useless or unsuitable ideas. Musical composition can be considered as an exploration for an optimal musical piece in a very large search space.

Genetic algorithms (GAs) represent a class of algorithms used for search and optimization inspired from the natural evolution. They were rediscovered by J. Holland [6], and have been used to solve many difficult problems [5, 11, 12]. In GAs, a set of *chromosomes*, representing candidate solutions is evolved towards an optimal solution. This set is referred to as *population*. A chromosome consists of *genes*, where

each gene receives a value from an *allele* set. The most common representation scheme is binary encoding that uses {0, 1} as an allele set. Depending on the problem, other appropriate encodings are allowed. In an evolutionary cycle, a set of genetic operators, such as, *crossover* and *mutation* is employed on initially randomly generated chromosomes. *Good building blocks*, possibly some part of an optimal solution are maintained within the population during the evolutionary process, while the *poor* ones are pruned based on an evaluation measured by a *fitness function*. With these properties, GAs can be considered as a suitable approach for generating musical composition. At the first glance, it might seem to be impossible for an algorithm to simulate the creative thinking process of a human. But GAs have achieved considerable results in the artistic fields.

The suitability of GAs for musical composition is researched and explained by Jones et al. [2]. The opportunities for evolutionary music composition are presented by Brown in [3]. Probably one of the most famous software about the topic is *GenJam* by Biles [4]. It uses an interactive GA to generate jazz solos over a given chord progression. Jacob also used interactive GA to implement a composing system and did research on the algorithmic composition [7, 8]. Papadopoulos et al. [13] and Wiggins et al. [16] produced a similar work to Biles. Instead of user intervention for evaluation, a fitness function is utilized in their implementation. Amnuaisuk et al. studied harmonizing chorale melodies [14]. 21 different melodic features of a musical fitness function are presented in [15]. Johnson used an evolutionary algorithm to generate melodies in the style of church hymnody and presented important fitness objectives for this purpose [9].

In this paper, a creative GA is presented. It is embedded into a Java user interface as a musical expert, named as AMUSE for generating improvised melodies over a musical piece given a harmonic context. AMUSE integrates a modified representation scheme and different fitness objectives under a GA approach for generating melodies automatically without a human feedback. The evaluation of such a subjective work is not trivial. In this study, the implementation is evaluated with respect to the GA performance, influence of each objective within GA and the variance of the generated melodies from that of a human composer by a group of human listeners.

In the next section, an overview of the GA used is presented. The representation scheme is explained in detail with examples and descriptions are given on how the genetic operators work. In Section 3, the objectives are explained and illustrated with example cases. In Section 4, the results of three different public evaluations are presented and finally, the conclusions derived from those evaluations are discussed and the performance of AMUSE is assessed in different aspects.

2 A Genetic Algorithm for Generating a Melody

AMUSE (A MUSical Evolutionary assistant) is a graphical user interface with which a user controls the parameters of GA and the attributes of the melody or solo to be generated as illustrated in Fig. 1. AMUSE can be downloaded from the web address:

http://cse.yeditepe.edu.tr/ARTI/projects/AMUSE. AMUSE allows a user to load the musical piece (substructure) for which an improvised melody will be generated in MIDI format (see http://www.midi.org for more).

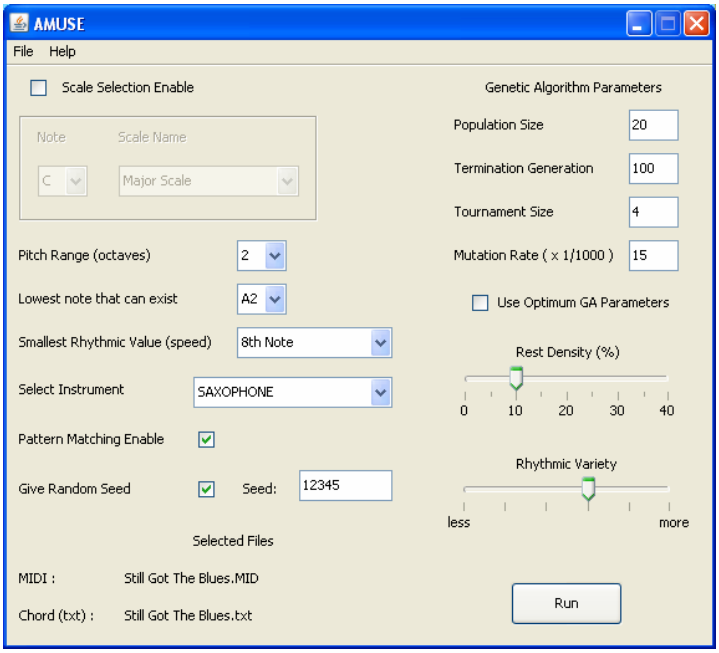

Fig. 1. A snapshot of AMUSE graphical user-interface

The GA requires a special representation scheme for generating melodies. The first scheme that comes to mind is having each gene to denote a musical note. Unfortunately, such a scheme is not sufficient alone, since it does not cover the duration of each note. This problem can be overcome by keeping two different allele values at each locus in a chromosome; a musical note and a rhythmic value. But, this produces another problem. The duration of each separate note can only be changed during mutation, limiting the variety of different rhythmic patterns. The representation scheme used in AMUSE addresses these issues.

The GA in AMUSE makes use of the traditional operators whose related parameters are chosen with respect to the chromosome length. One point and two point crossovers are both used with equal probability. One of them is employed to each selected individual pairs (mate) in the population. During the preliminary experiments, it is observed that this hybridized crossover performs better than applying just one type of crossover. All mates are chosen using tournament selection with a tour size of four. The population size is one third of the chromosome length. The offspring pool size is the same as the population size. The traditional trans-generational re-

placement scheme is used. The mutation operator in AMUSE randomly perturbs an allele within an allowed range using a mutation rate of 1/*chromosome-length*. Choice of default values for GA parameters is arbitrary and based on the previous studies in [10-12]. AMUSE allows the user to modify these default values.

2.1 Representation

In AMUSE, a chromosome (*individual*) encodes an improvised melody having the contribution of some other parameters. An *allele* (set of values a gene can have) value represents the order of a note in a given scale. The advantage of this representation scheme is that it is impossible to generate non-scale notes. For example, a '4' indicates the 4^{th} note in the corresponding scale, '0' indicates a rest. The maximum allowed allele value indicates a *hold event* that makes the previous note to continue. Although the representation scheme is similar to the one used in *GenJam* [4], there are some differences. In *GenJam*, there are two different populations: *measure* and *phrase* populations. An individual in the measure population maps to a sequence of MIDI events. An individual in the phrase population maps to the indices of measures in the measure population. But, in AMUSE, a single population of individuals is maintained; each individual corresponds to a whole melody. Another difference is that; in our implementation, the range of the notes can be expanded or narrowed and the durations represented by each gene can be adjusted.

As stated before, some additional information is required to obtain the actual musical notes from the chromosome. For example, durations of the notes are needed. Those durations are derived from two values; one of them is the *hold event*. Hold events in a chromosome stretch the duration of the notes that are placed right before them in a chromosome. The other one is the *rhythmic value* that is defined the same for all genes. This value provides us the duration of each note identified by a gene (e.g. 'Eighth Note' or a 'Whole Note'). In a way, the rhythmic value specifies the shortest note that can be heard in the melody. On the other hand, it is a parameter that increases (decreases) the search space size with the chromosome length when the duration represented by each gene is shortened (elongated). Such relevant information can be entered into the AMUSE as a user preference, providing more flexibility for the user over the melody. Also, the maximum allele value can be adjusted for expanding or narrowing the pitch range of the melody to be generated. Furthermore, the *beginning note* can be modified for shifting the pitch range to higher or lower octaves. This value is added to all of the alleles when creating the melody from the genotype and it takes the melody to higher or lower octaves.

As an example; assuming that the given scale is C Major, the rhythmic value of all notes is a quarter note, the maximum allowed allele value (hold event) is 15 (corresponds to a pitch range of two octaves for a seven-note scale) and the beginning note of the pitch range is A2 (the third lowest A note in a piano). According to these parameters, mapping of the gene values and musical notes are shown in Fig. 2 (a) and the example chromosome in Fig. 2 (b) is decoded as in Fig. 2 (c). But, sometimes the

user might not want to generate a melody using just a single pitch and scale. Then, each gene will be mapped to a different scale and pitch. In such a case, the same mechanism can be still employed. Only, the values in a chromosome will be mapped to the actual notes based on different scales and pitches.

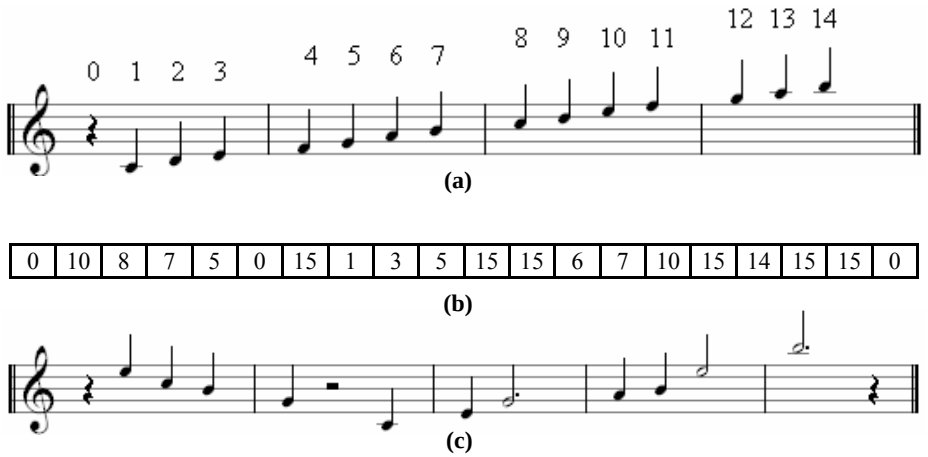

Fig. 2. (a) Gene values and the corresponding eighth notes in C Major Scale **(b)** An example chromosome, and **(c)** The actual notes that might be decoded from that chromosome

3 Objectives and the Fitness Function

One of the most important components of a GA for generating a pleasant melody is the fitness function. Fitness function can evaluate ten different objectives (Eq. 1). Some of these objectives are also used by Johnson in a different manner [9]. Note that some objectives might not be evaluated depending on the user's preferences. Eight objectives (f_1 to f_5 and f_8 to f_{10}) generate a value in [0,1) for a given candidate melody x that are equally weighted with w_i=1/|no.of_objectives_used|. Two of them (f_6 and f_7) check for some condition and apply punishment, generating a value in (-1,0]. Fitness value varies in the range (-2,1). A higher fitness value denotes a better melody.

$$f(x) = \sum_{i=1}^{10} w_i f_i(x) \quad \textbf{(1)}$$

Objectives can be divided into two groups; *core* and *adjustable* objectives. Core objectives can not be manipulated by the user, whereas the adjustable objectives are maintained according to the initial preferences of a user:

- **Core objectives**: Chord note (f_1), relationships between notes (f_2), directions of notes (f_3), beginning note (f_4), ending note (f_5), over fifth (f_6), drastic duration change (f_7)

- **Adjustable objectives**: Rest proportion (f_8), hold event proportion (f_9), pattern matching (f_{10})

Chord note objective checks whether an actual note decoded from a gene is in the notes of the chord of the same beat or not. The objective value denotes the percentage of such notes over all notes. This evaluation is provided only when a chord file is defined. Otherwise, only the scale is known and this objective is not considered.

Several different type of relationship between notes are analyzed and evaluated. The objective function for the relationships between notes returns a weighted sum of all sub-objective values. All consecutive notes are considered one by one. The number of such note groups is counted with respect to the category each of them falls, returning the sub-objective value. In order to get an overall evaluation of the objective, a scaling that will be referred as *overall scaling* is performed by dividing the total by the number of actual notes minus one. The sub-objectives are as follows:

- *One Step* (Fig. 3 (a)): Next note's scale degree is one step higher or lower than the previous note's degree. When the notes belong to different scales, it is checked whether the next note is one or two half steps higher or lower. The weight is 1.0.
- *Two Steps* (Fig. 3 (b)): Next note's scale degree is two steps higher or lower than the previous note's degree. When the notes belong to different scales, it is checked whether the next note is three or four steps higher or lower. The weight is 1.0.
- *Same Note* (Fig. 3 (c)): Next degree is the same as the previous degree. (0.9).
- *Three Steps* (Fig. 3 (d)): Next note's scale degree is three steps higher or lower than the previous note's degree. When the notes belong to different scales, it is checked whether the next note is five steps higher or lower. The weight is 0.8.
- *Four Steps* (Fig. 3 (e)): Next note's scale degree is four steps higher or lower than previous note's degree. When the notes belong to different scales, it is checked whether the next note is six or seven steps higher or lower. The weight is 0.7.

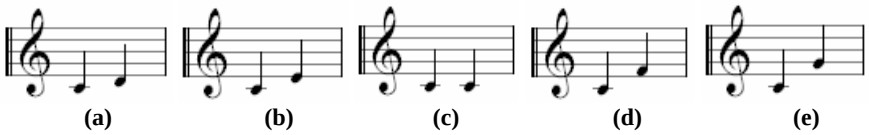

Fig. 3. Examples of relationships between notes in C major scale

When the tonal distance between two consecutive notes becomes smaller, it is more likely that they will sound better. If the tonal gap between notes becomes larger, then the notes should be carefully organized to provide a better sound. Therefore, higher weights are assigned for smaller tonal distances to get a melody progressing with small steps.

In music, *direction*, in other words, *contour* of the melody is also important. A melody might fall, rise or stay stable (Fig. 4). The objective function for the directions of notes evaluates three sub-objectives simultaneously where each objective counts the note triplets that fall into their category in a given melody. The note triplets represent all three actual consecutive notes in the melody. Finally, overall scaling is performed on the weighted sum of all sub-objective values by the number of actual

notes minus two:

- *A Falling Melody* (Fig. 4 (a)): This sub-objective checks whether a given three consecutive actual notes form a falling melody or not. The weight is 1.0.
- *A Rising Melody* (Fig. 4 (b)): This sub-objective checks whether a given three consecutive actual notes form a rising melody or not. The weight is 1.0.
- *A Stable Melody* (Fig. 4 (c)): All three notes are the same. The weight is 0.9.

We do not desire the generated melodies to change direction up and down rapidly, because that sounds unpleasant in general. Instead, a smoothly flowing melody is desired that has a progress in one direction for a while. The weights of the falling and rising melodies are set to higher values as compared to the stable melodies to support these sub-objectives. Notice that the stable melodies are not ignored completely, since they might still generate a pleasant sound.

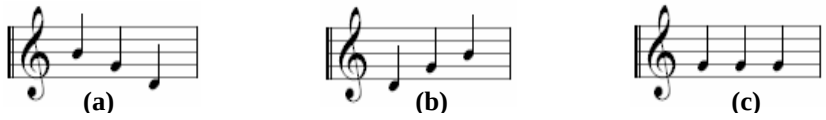

Fig. 4. Example of **(a)** a falling melody, **(b)** a rising melody, and **(c)** a stable melody

The beginning of a musical piece is important. It is like an introduction or a starting step for the song or music. This objective function returns 1.0 for a candidate melody, if it starts with the root note of the scale, since it is always harmonically correct and never disturbs the listener. Similarly, the ending is like a conclusion for a musical piece and the root note of the corresponding scale is again a good choice for an ending note, since, we hear all the notes in a song relative to the root. Due to this affect, the same notes in different songs raise different feelings. As a result, ending a melody with a base sound resolves the music comfortably. This objective function also returns 1.0 for a candidate melody, if it ends with the root note of the corresponding scale.

Over fifth objective is a complementary objective to the relationships between notes objective. All pairs of consecutive notes in a candidate melody are scanned. The objective function checks if the next note's scale degree is more than four steps higher or lower than the previous note's degree for each pair. Such pairs are counted and an overall scaling is performed by the number of actual notes minus one. The objective function punishes such tendencies that might generate unpleasant solo by returning a negative value of the scaled count. An example over fifth violation in C Major Scale is shown in Fig. 5 (a). A is the 6^{th} note from C in C Major Scale.

Drastic duration changes between consecutive notes might sound disturbing. Hence, this objective function punishes such occurrences in a melody by checking the proportion of the duration of two consecutive notes. All pairs generating a proportion that is more than four are counted. After an overall scaling is performed by the number of actual notes minus one, the objective function returns a negated value. An example duration change violation is shown in Fig. 5 (b). C is a whole note and F is an eighth note in this example.

Fig. 5. Example of **(a)** an over fifth violation in C Major Scale, **(b)** a drastic duration change

Rests can make the melodies more pleasant as if providing breaks in the melody. Rest proportion is the ratio of the total rest durations to the overall duration. User determines an expected value, denoted by *rpr* beforehand. Then, AMUSE attempts to arrange a melody carrying a rest proportion as close as possible to the *rpr*. For a candidate melody *x*, given that the rest proportion is denoted by $z = restPropOf(x)$, then the objective returns $f_8(x) = -200(restPropOf(x) - rpr)^2 + 1$, only if z is within ±0.05 of the *rpr*. Otherwise, f_8 returns zero.

Every allele value receives the same rhythmic duration, determined by the user. Notes having different rhythmic durations enrich a melody. An allele value corresponding to a hold event makes the duration of a note longer. Hold event proportion is the ratio of the number of hold events to the chromosome length. Similar to the *rpr*, an expected hold event proportion can be defined by the user, denoted as *hpr*. For a candidate melody *x*, given that the rest proportion is denoted by $z = holdEventPropOf(x)$, then the objective returns $f_9(x) = -50(holdEventPropOf(x) - hpr)^2 + 1$, only if z is within ±0.1 of the *hpr*. Otherwise, f_9 returns zero. The *hpr* and *rpr* provided by the user is also considered during initialization. Every initial individual is generated around the expected rest and hold event proportion.

A user might require having similar patterns in a melody. Choruses in the songs are good examples for such patterns in music. Pattern matching objective function searches for repeated patterns with 3 to 10 notes in a melody. For each note in comparison of a pattern; an award of 0.5 is given for the same notes with different durations, 1.0 for the same notes with the same durations. For the sample melodies in Fig. 6 (a) and Fig. 6 (b), the overall awards are 1.5 and 3.0, respectively. The maximum overall award occurs (Fig. 6 (c)) only if a melody repeats the same three note pattern. A scaling is performed on the overall award by this maximum possible value.

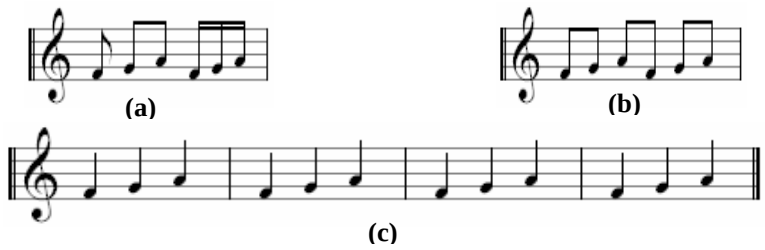

Fig. 6. Three note pattern: **(a)** same note, different duration, **(b)** same note and duration, **(c)** maximum possible overall award situation

4 Results

In this section, the results of a public evaluation are presented. This evaluation is completed by the help of 36 university students around the age of 20 from different schools and departments. It consists of three parts. The first part is a Turing Test (27 students). Two MIDI formatted music files are given to the participants. Both files share the same melodic substructure, but they carry different solos. One of the solos is generated by an amateur musician who has not listened to any solo generated by AMUSE before. The other solo is generated by the GA in AMUSE. The participants are asked to find the correct author of each solo in the musical pieces; human versus computer. The results in the first part show that, the participants cannot differentiate the human work from the computer's work, because 52% of them provided wrong answers. This is a satisfying result, since our goal is not for AMUSE to beat an amateur musician, but to provide a matching performance.

The second part of the evaluation is arranged to observe whether the GA in AMUSE improves the initial population of melodies or not (20 students). At first, five different MIDI formatted music files are generated by using Band-in-a-Box software by PG Music Inc. (http://www.pgmusic.com). Then a pair of solos are obtained for each of them using the AMUSE. One of the solos is obtained from the initial population with a lower fitness, while the other one is obtained from the 1000^{th} generation with a higher fitness. The rest of the parameters are kept the same during the runs. In this part, the participants listened to these ten solos in the given music files and ranked them as they like. 10 indicate the best solo, whereas 1 indicates the worst solo. The results are presented in Table 1. A solo with the higher fitness ranks as the first. Notice the order of the average ranks of each pair of solos based on the same substructure having different fitness values in Table 1. Each solo with a higher fitness has a better rank than the other solo using the same substructure with a lower fitness. The results between the files that share the same substructure are compared to remove the subjectivity of the participants, because they all evaluate the songs according to their musical taste. Still, all solos having higher fitness rank better than the ones having lower fitness.

Furthermore, two-tailed sign test is used to investigate whether these rankings are statistically significant or not. Two related groups are compared according to the distribution similarity. The result of the sign test ('p' value) gives us the probability that those two groups are from the same underlying distribution. It is applied to the ranks obtained from the subjects for the song pairs. According to the results; almost all pairs (4 out of 5) have statistically significant performance variances within different confidence intervals. Among 20 subjects, V1:V2 and X1:X2 gave a result of 3 versus 17; W1:W2 and Y1:Y2 gave 4 versus 16; Z1:Z2 gave 6 versus 14 for the number of correct evaluations along with fitness values versus the number of wrong evaluations. As 3 versus 17 is a significant difference with significant level of 1% ($p<0.01$) and 4 versus 16 is also significantly different with a level of %5 ($p<0.05$). However, 6 versus 14 is not significantly different enough ($p\approx0.1$) according to the minimum accepted significant level of %5. On average, 'Z' substructure is the least

liked one, hence, that might be the reason why 'Z' pair received less significance. Moreover, Z1 among all initially generated melodies and Z2 among all melodies that AMUSE improved in 1000 generations have the least fitness values.

Table 1. Evaluation results for the pairs of solos produced by AMUSE, where each pair uses the same substructure and one song in a pair is generated during the initial generation while the other during the 1000th generation. The letters in the song title denote the substructure.

Song Title	*V*1	*V*2	*W*1	*W*2	*X*1	*X*2	*Y*1	*Y*2	*Z*1	*Z*2
Fitness	0,92	0,30	0,29	0,94	0,94	0,27	0,94	0,24	0,17	0,90
id	*Rank of each song provided by each participant*									
1	10	7	1	2	9	8	5	6	4	3
2	10	7	8	9	5	3	6	4	1	2
3	2	1	3	4	8	7	10	9	6	5
4	2	1	3	4	10	5	8	6	7	9
5	9	7	1	6	8	3	10	2	4	5
6	3	4	9	10	7	8	1	2	5	6
7	10	9	6	7	8	5	2	1	4	3
8	9	10	3	4	5	6	7	8	1	2
9	3	1	2	4	9	8	10	7	5	6
10	10	4	3	6	9	2	8	1	7	5
11	6	5	3	7	4	2	8	4	1	3
12	3	4	6	5	8	9	2	1	10	7
13	9	3	6	7	8	2	10	4	1	5
14	10	6	3	4	9	5	8	1	2	8
15	4	3	6	5	8	7	9	10	1	2
16	2	1	4	3	10	9	6	5	8	7
17	7	2	5	8	10	1	9	3	4	6
18	4	1	7	8	10	2	9	3	5	6
19	9	6	8	7	10	3	5	4	1	2
20	9	8	1	3	7	2	10	5	4	6
Avr. Rank	6,55	4,50	4,40	5,65	8,10	4,85	7,15	4,30	4,05	4,90
Std.	3,27	2,89	2,48	2,18	1,77	2,72	2,87	2,72	2,67	2,13

In the third part of the evaluation, the relative importance of each objective composing the fitness function is investigated (13 students). Six different music files are generated by AMUSE using the same musical substructure. One of the files is generated using the fitness function that evaluates all objectives, while the rest of them exclude one of the objectives. The participants ranked each musical piece from one to six, similar to the previous part. As summarized in Table 2, the best solo having the highest average rank is the one that is generated by using the fitness function evaluating all the objectives as discussed in Section 3. Over fifth objective seems to be the most important one. Chord note, relationships between notes and duration change

objectives follow it according to the order of importance from the highest to the lowest. The effect of the direction change objective on the generated melody might not be perceived by all the listeners and in some cases; the direction of the notes can be well-arranged even without using this objective. From the results of this part, it can be concluded that it is the least important objective.

Table 2. Comparison of solos evaluated by different fitness functions, - indicates the excluded objective

Fitness Function	f	$-f_1$	$-f_2$	$-f_3$	$-f_6$	$-f_7$
id	*Rank of each solo provided by each participant*					
1	6	1	4	5	3	2
2	4	3	5	2	1	6
3	4	3	6	5	1	2
4	6	4	3	5	1	2
5	1	3	2	6	4	5
6	4	6	2	5	1	3
7	4	5	2	6	1	3
8	3	6	4	5	2	1
9	6	1	2	5	3	4
10	5	1	2	3	4	6
11	6	1	5	3	2	4
12	5	6	3	2	1	4
13	6	1	2	3	4	5
Avr. Rank	4.62	3.15	3.23	4.23	2.15	3.62
Std.	1.50	2.08	1.42	1.42	1.28	1.61

5 Conclusions and Future Work

An automated evolutionary approach for generating melodies is discussed and evaluated in this paper. The proposed approach aims to fulfill two important requirements; generating melodies that are harmonically correct and sound pleasant to an ordinary listener. The first goal is achieved by making use of an effective representation mechanism based on the theory of the relationships between chords and scales. The second goal is achieved using an evaluation function that includes some fundamental objectives to push the search towards pleasing or at least non-disturbing melodies. The objectives assembled within the evaluation function are not mandatory rules in music. They might restrict the variety of melodies, but at the same time they reduce the risks of unpleasant melody generation.

GAs have already been used successfully in art. This study also showed that they can yield satisfying results in the field of music. The GA in AMUSE generates pleasant solos for a given substructure, that is comparable with an amateur human musi-

cian. An ordinary listener who has never listened to the solos generated by AMUSE before, cannot easily differentiate between an improvisation written by an amateur human composer and AMUSE. But, at this point, it is still not sufficient for an experienced composer to use AMUSE for practical purposes. In this form, it is more suitable for research and provides a basis to develop more sophisticated designs. To achieve better results and make the tool to be a practical one for users rather than a research tool, the fitness function may be rearranged. New fitness objectives can be added and new genetic operators can be embedded to cooperate with the fitness objectives. Current chord and scale mappings can be rearranged and new chord types can be added in order to expand the harmonic vocabulary.

References

1. Bentley, P. J., Corne, D. W. Creative Evolutionary Systems, Morgan Kaufmann Publishers (2002)
2. Gartland-Jones, A., Copley, P. The Suitability of Genetic Algorithms for Musical Composition, Contemporary Music Review, Vol. 22, No. 3 (2003) 43-55
3. Brown, A. R. Opportunities for Evolutionary Music Composition, Australasian Computer Music Conference, Melbourne: ACMA (2002) 27-34
4. Biles, J. A. GenJam: A Genetic Algorithm for Generating Jazz Solos, Int. Computer Music Conf. (ICMC'94). Aarhus, Denmark (1994) 131-137
5. Goldberg, D. E., Genetic Algorithms in Search, Optimization, and Machine Learning, Addison-Wesley, Reading (MA) (1989).
6. Holland, J. H., Adaptation in Natural and Artificial Systems, Univ. Mich. Press (1975).
7. Jacob, B. L. Algorithmic Composition as a Model of Creativity, Organised Sound: Vol. 1, no. 3. Cambridge: Cambridge University Press (1996) 157-165
8. Jacob, B. L. Composing With Genetic Algorithms, Proc. of the 1994 International Computer Music Conference, (1995) 452-455
9. Johnson, M., Tauritz, D. R., Wilkerson, R. Evolutionary Computation Applied to Melody Generation, Proc. of the ANNIE 2004 (2004)
10. Ozcan, E., An Empirical Investigation on Memes, Self-generation and Nurse Rostering, Proc. of the 6th International Conference on the Practice and Theory of Automated Timetabling (2006) 246-263
11. Ozcan, E., Mohan,C. K. Partial Shape Matching using Genetic Algorithms, Pattern Recognition Letters, 18 (1997) 987-992
12. Ozcan, E., Onbasioglu, E. Memetic Algorithms for Parallel Code Optimization, International Journal of Parallel Programming, Volume 35, Number 1 (2007) 33-61
13. Papadopoulos, G., Wiggins, G. A Genetic Algorithm for the Generation of Jazz Melodies, STeP'98, Jyväskylä, Finland (1998) http://citeseer.ist.psu.edu/papadopoulos98genetic.html
14. Phon-Amnuaisuk, S., Tuson, A, Wiggins, G. Evolving Musical Harmonization, The University of Edinburgh, Division of Informatics, Research Paper #904 (1998)
15. Towsey, M., Brown, A., Wright, S., Diederich, J. Towards Melodic Extension Using Genetic Algorithms, Technology & Society, 4(2) (2001) 54-65
16. Wiggins, G, Papadopoulos, G., Phon-Amnuaisuk, S., Tuson, A. Evolutionary Methods for Musical Composition, Proc. of the CASYS98 Workshop on Anticipation, Music & Cognition (1998) http://citeseer.ist.psu.edu/13486.html