

# A Genetic Algorithm for the Generation of Jazz Melodies

George Papadopoulos and Geraint Wiggins  
Department of Artificial Intelligence  
University of Edinburgh  
80 South Bridge, Edinburgh EH1 1HN, Scotland

Email: {georgep,geraint}@dai.ed.ac.uk

## Abstract

This paper describes a system for the generation of jazz melodies over an input chord progression. A genetic algorithm was used to search through the space of possible solutions. A symbolic, as opposed to binary, approach with domain-specific reproduction operators was chosen because it allowed knowledge based constraints to be imposed on the search space. The objective, algorithmic fitness function as well as the domain-specific genetic operators orientate the search to promising musical paths.

## 1 Introduction

Genetic algorithms (GAs) (Holland, 1975; Goldberg, 1989) have proven to be very efficient search methods, especially when dealing with problems with very large search spaces. This, coupled with their ability to provide multiple solutions, which is often what is needed in creative domains, makes them a good choice for a search engine in a musical application.

In this paper, we describe a series of experiments using a GA to generate melodies in a jazz style, given an underlying harmonic progression. Our GA exhibits three significant characteristics which are uncommon in GA applications to music:

- *An algorithmic fitness function for objective evaluation of the GA results.* In this work, we focus on the searching behaviour of our algorithm, and so we need to know exactly what criteria we are using to evaluate the melodies. By the word ‘objective’ we do not mean ‘correct’ or ‘universal’, since the evaluation of any artistic endeavour is subject to many parameters, not least personal preference. The point is that we need a consistent way to evaluate our results since this will give us the necessary feedback about any shortcomings of our work. We are interested in understanding the search and how it might simulate human behaviour and not just in achieving a musical result.
- *Problem-dependent genetic operators.* In traditional GA implementations, the genetic operators do not have knowledge of the domain. They often operate at bit level and therefore they are suitable for any problem encoding (*i.e.*, they are a “weak” search method). In contrast, problem-dependent operators are designed for a specific problem. For this reason, they can deal more effectively with it.
- *Symbolic representation of the structures and the data.* Many GA problem encodings are binary. A symbolic representation is easier to interpret by the user and can restrict the use of genetic operators to actions which are meaningful in the context of the domain knowledge.

The structure of this paper is as follows: first we present a review of existing GA applications in music. We then proceed with the implementation and evaluation of our system presenting also some results. We discuss a few possible extensions to our work and finish with our conclusion.

## 2 Related Work

GAs’ popularity in the AI community has increased rapidly over the last decade. There are several systems using this technique to generate music algorithmically. We can divide these attempts into three different categories.

**Use of a prescribed, knowledge-based fitness function:** There are not many attempts in this category because it is difficult to find objective rules for the evaluation of music. Horner and Goldberg (1991) used GAs for thematic bridging between very simple melodies; McIntyre (1994) generated a four part Baroque harmonisation of an input melody. McIntyre used only the C major scale in order to reduce the search space.

**Use of a human user as a fitness function:** In this case a user replaces the fitness function in order to evaluate the chromosomes. Jacob (1995) devised a composing system; Horowitz (1994) generated “tasteful” rhythmic patterns; Ralley (1995) developed melodies; and Biles (1994), whose work is most closely related to ours, evolved a “novice jazz musician learning to improvise”. All these attempts exhibit two main drawbacks associated with all IGAs: *subjectivity* (Ralley 1995) – it is very likely that the user will be biased from previous listenings, or even change her mind over time because of a change in her mood or of boredom – and *efficiency*, the “fitness bottleneck” (Biles 1994) – the user must hear all the potential solutions in order to evaluate a population. Moreover, this approach tells us little about the mental processes involved in music composition since all the reasoning is encoded inaccessibly in the user’s mind.

**Use of a neural network as a fitness function:** Gibson and Byrne (1991) created simple harmonizations, using only the tonic subdominant and dominant chords, of short melodies using cooperating neural networks. Spector and Alpern (1995) used genetic programming (Koza, 1992) to create a one measure response to a one measure ‘call’, with a neural network as a fitness evaluator of the response. Biles et al. (1996), in an attempt to increase the efficiency of Biles’ (1994) system, used also a neural network, without successful results.

Most of the approaches above exhibit very simple representations in an attempt to decrease the search space, which in some cases compromises their output quality. Musical questions are sometimes left unanswered, too. For example, in Spector and Alpern’s (1995) and Ralley’s (1995) case, how can we expect to evaluate the system’s response if we do not have a harmonic context for it? We discuss these and other related issues in section 3.1 below. For a more complete summary of GA work in music see Burton and Vladimirova (1997).

In the current state of the art, it is not known how to implement a fitness function which will rate the quality of music, even in a very restricted domain. However, it is possible to provide the user with a number of choices, then implementing a fitness function which rates how well the musical output fits the user’s preferences.

The GA described below uses an objective and consistent fitness function which encodes knowledge borrowed from research in cognitive psychology and statistical analysis of pieces. It is more efficient than an IGA – it takes less than a minute to produce a 12-bar melody on a single-user Sun Ultra 1 – and it exhibits generality in the representation of knowledge, allowing variable chromosome lengths, any number of chords in any position in the piece – not just in the downbeats – and mixed note durations.

## 3 Implementation

### 3.1 Design Motivation

In section 1, we introduced the idea of using an *objective* fitness function, as opposed to the interactive approaches often used elsewhere in the GA music field. The reason for this is that we have a particular interest in understanding the searching behaviour of our GA: we are interested in *simulating human behaviour* and not just in the quality of our results. In order to understand the search patterns produced by our system, it is important to have a fitness function which is consistent, and whose criteria we fully understand. This could not be the case with an interactive GA, because of the subjectivity of the human listener – it would be impossible to determine which choices were made because of emergent behaviour of the system and which were made because of the inconsistencies of the human judge. Equally, the intermediate position of training a neural network according to a human listener’s expressed preferences is unsuitable for our activities: while the network could be relied upon to reflect the listener’s choices consistently, generalisations made might be musically groundless; and we would still be in the position of not being able to analyse the behaviour of the system in terms of the knowledge encoded in the fitness function. However, let us emphasise that we are *not* suggesting that the interactive or network-aided approaches are inappropriate in all cases – they are simply inappropriate to our purposes here.

This said, it would be a challenge indeed to build the perfect algorithmic fitness function which would direct the search towards maximally desirable melodies. So instead of trying to approach this intractable problem, we have built domain-orientated operators and a fitness function which imitate the basic improviser’s “work tools” and mental processes, drawing on the literature on jazz improvisation (Coker, 1964; Fakanas, 1990; Sabatella, 1996, for example) for inspiration. We expect, therefore, that although the final output may not be as impressive as some of the existing jazz improvisation GAs (see section 2), it will nonetheless help us understand what is missing from this and other systems more clearly.

### 3.2 The Genetic Algorithm

The implementation of the Genetic Algorithm is in the style of Davis (1987). The selection method used was tournament selection. The merge operator simply copies the intermediate population to the new population. Figure 1 shows the different steps of the algorithm for each generation.

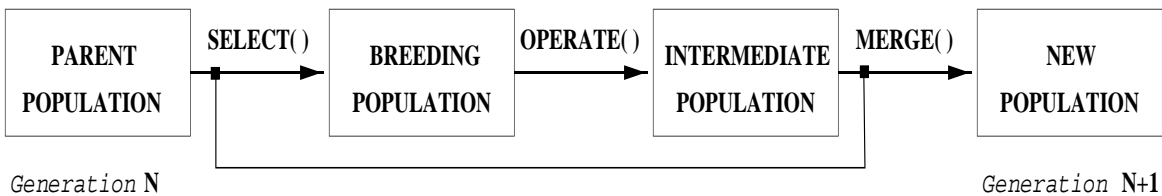


Figure 1: The steps of the Genetic Algorithm

### 3.3 Knowledge Representation

Our motivation was to find a flexible and efficient representation for the chromosome. We decided that the chromosome would represent the degrees of the scale, relative to the current chord, rather than the absolute pitches – a *degree-based* representation. The advantage of this approach is that it uses the combination of the degree and its corresponding chord to specify the actual pitch of the melody. This means that we cannot generate non-scale notes, except in the potentially musically desirable case where a melody note spans two different chords, which, in jazz terminology, means it spans two different scales. Because the interpretation of the note is based on the first chord, there is a possibility of a dissonant suspension over the chord change, which is then dealt with by the fitness function. Instead of simply having degrees from 1 to 8 (where 8 is used for 8-degree jazz scales) we used an *extended-degree* representation, giving 21 different values which correspond to 3 octaves for a 7-note scale, and about 2.5 octaves for an 8-note scale.

The chromosome is then a sequence of  $\langle \textit{extended-degree}, \textit{duration} \rangle$  pairs, rests being distinguished by the constant *rest* in place of the *extended-degree*.

The chord progression, input by the user, on which the melody is to be based, is a sequence of  $\langle \textit{root}, \textit{chord-type}, \textit{duration} \rangle$  triples, using standard musical nomenclature, e.g.,  $\langle \text{Bb}, \text{maj7}, \text{minim} \rangle$ .

### 3.4 Initialisation

In our implementation, the input chord progression determines the duration  $d$  of the song. A population of size  $n$  is a set of  $n$  chromosomes represented as  $\langle \textit{Fitness}, \textit{Genotype} \rangle$  pairs. The preparation of the population consists of two steps: initialisation; and then initial evaluation. For each chromosome in the population, a sequence (genotype) of  $m$   $\langle \textit{extended-degree}_i, \textit{duration}_i \rangle$  pairs is generated, where  $\sum_{i=1}^m \textit{duration}_i = d$ . The pitches are chosen randomly with uniform probability. The probability of generating a rest was set to 12.5%. The user can specify the possible note durations and their respective probabilities. The fitness function is then used to evaluate each randomly generated genotype.

### 3.5 Genetic Operators

The speed of convergence to high fitness of this system, and the quality of its results are based largely on the genetic operators. Three classes of musically meaningful (Biles, 1994) mutations were implemented, the partition being based on the way the mutations operate on the chromosome. The classes are:

**Local mutations** These mutations operate on a random chromosome fragment of random length. For example, such mutations transpose by a random number of degrees; permute in time; sort into ascending or descending pitch; reverse in time; change a few pitches while maintaining the same rhythm; shuffle the note durations while maintaining the order of the pitches; and concatenate contiguous rests and identical pitches. There is also a simple one-note mutation which just changes the pitch of one note up or down, which can be helpful when the melody needs only small changes for large increases in fitness.

**Copy & operate mutations** These mutations copy a randomly chosen fragment to a different position while possibly operating on it as per a local mutation. Included in this class is also a swap mutation, which swaps two segments, rather than overwriting one. The original material at the target position is overwritten, which can necessitate splitting one or two notes' durations into two, in order to preserve the total duration of the melody.

**Restricted copy & operate mutations** The last class of mutation also chooses fragments randomly, but this time from a set of given starting positions and with constant size. The starting position can be the first or third beat of any bar, and the size is half a  $\frac{4}{4}$  bar (*i.e.*, a minim), so if we have a melody of  $n \frac{4}{4}$  measures then there are  $2n$  fragments to choose from. This restriction seems to be useful because the ear recognises familiar motifs more easily when they start at consistent metric points within a piece. Note, though, that the less regular patterns given by the unrestricted mutations above may be musically interesting – this is why both the types of copy & operate mutations were implemented. The user can then choose which she prefers to experiment with.

One-point and two-point crossover were also implemented. As above, these operations can sometimes necessitate splitting the durations of notes into two.

The operators were selected with probabilities ranging from 10% to 20%.

### 3.6 Fitness Function

The fitness function evaluates eight distinct characteristics of a chromosome, from which it calculates, *via* a weighted sum, the corresponding overall fitness. This section describes these characteristics and explains the reasons why they were chosen. Corresponding weights are given in round brackets.

**Large intervals** Because of the random initialisation of the chromosomes, it is possible that there will be very large intervals between consecutive notes, which can be aesthetically irritating for the listener, as it contradicts Gestalt law of proximity (Leman, 1997). This part of the fitness function reduces this problem. The user can specify the largest permissible interval between consecutive notes. The fitness penalty is the sum of the sizes of the intervals which are larger than permissible, multiplied by a constant weight ( $-20$ ).

**Pattern matching** In this implementation, pattern matching occurs only between pitch fragments, and not rhythmic ones. The addition of pattern matching to the fitness function is psychologically motivated, from the Gestalt law of similarity (Leman, 1997), and can be statistically supported, by analysing human-generated melodies. Listeners not only recognise but expect similar patterns in music – it is by means of a partly repetitive structure that the development of the music is communicated. In accordance with this, musicians have the tendency, when they improvise using patterns, to vary them, for example by transposition or change of rhythm.

Even though our genetic operators were designed specifically to create the feeling of thematic development in the melody by generating variations of motifs, we decided that it was useful to add this feature to the fitness function, partly in order to see if the operators work as expected, but also to allow for the case where motivic development occurred by chance.

In our system, the user can specify whether she prefers that the algorithm will try to find exactly matching interval patterns or she prefers a looser form of matching.

The output of the pattern matching algorithm is a list of numbers, where each number  $n$  denotes that there exist two similar patterns of length  $n$  in the melody. The system does not attempt to find overlapped similar patterns. By default, the patterns must be of five or more notes in length. The allocation of weighting to this part of the fitness function is discussed in section 4.

**Suspensions** As explained in section 3.3, it is possible to generate suspensions – notes which lie across two consecutive chords. If the chord sequence is of length  $n$  then the melody may have up to  $n - 1$  suspensions. This part of the fitness function checks what happens to those  $n - 1$  chord changes. We considered four cases, with distinct weights: there is a consonant suspension, meaning that the note is a

member of both scales determined by the two consecutive chords (+10); there is a dissonant suspension, which means that the note is a member of the first scale but not of the second (-20); there is no suspension (+5); or there is a rest (+5).

**Note at downbeat** The first beat in a bar is usually the most musically significant beat in that bar. The downbeat can be: a chord note (+10); a rest (+10); a non-chord but still a scale note (-10); or a non-scale note (-20).

**Note at half-bar** The same as the downbeat function, but for the third beat of each bar. Since the third beat is weaker than the first beat of a  $\frac{4}{4}$  bar, it is less restricted. Therefore the respective, weights, except for the non-scale note case, are smaller: (+5), (+5), (-5), (-20).

**Long notes** The user can specify what she considers to be long notes. Long notes are mostly used in music as points of stasis. Therefore, it is preferable to have harmonically stable long notes. A long note can be: a chord note (+10); a non-chord scale note (-20); a rest (-20); a consonant suspended note (+20); or a dissonant suspended note (-20). Long rests are penalised because they damage the continuity of the solo.

**Contour** This is a comparison between the contour of the chromosome and the contour specified by the user. The user specifies whether the average pitch of each bar is lower than, the same as or higher than the last. Thanks to Prolog’s unification mechanism, it is also possible to specify that the contour in one place is the same as in another, but without saying exactly what it is. If  $U_i$  and  $M_i$  denote the direction of change in the  $i$ th bar of the user-specified and chromosome contour respectively (ranging over  $(-1, 0, +1)$  in the obvious way), then if  $U_i = M_i$  the fitness bonus is +30 points; otherwise, the penalty is  $-30 |U_i - M_i|$ .

**Speed** Similar to Contour, except that the algorithm is making an estimate of the speed of the piece simply by adding the number of notes and rests in each pair of consecutive bars, and matching them to ‘slow’, ‘medium’, or ‘fast’ as appropriate.

### 3.7 Input, Parameters and Preferences

The user input, parameters and preferences for our experiments are summarised in table 1.

<b>Selection method</b>	Tournament selection, pick 1 from 4.
<b>Merging method</b>	Simple copy.
<b>Population size</b>	40 chromosomes.
<b>Maximum generation</b>	The maximum number of generations, which was also the termination condition was 200. In many cases the GA converged to the highest fitness much earlier.
<b>Melody duration</b>	The duration of the solo was chosen to be 192. This corresponds to $12 \frac{4}{4}$ bars, as we use 1 for a semi-quaver duration.
<b>Chord progression</b>	$12 \frac{4}{4}$ bar blues chord progressions where taken from (Coker, 1964).
<b>Large interval</b>	Greater than 9 semitones (major sixth).
<b>Pattern matching</b>	Pattern matching, when used, was set to match absolute pitch intervals, with no fuzzy matching. In other words, this means that the pattern matching algorithm was trying to find identical or transposed patterns only.
<b>Weights</b>	All the weights, except the pattern matching weight, were constant for all the experiments.
<b>Long notes</b>	If the duration of a note or a rest is a larger than or equal to a crotchet.
<b>Duration probabilities</b>	We experimented with different duration probabilities. In many of the experiments the note probabilities were: 70% for a semi-quaver, 20% for a quaver and 10% for a crotchet.
<b>Hill-climbing</b>	In many of the experiments we did not allow the child to replace its parent if it was not at least as fit.

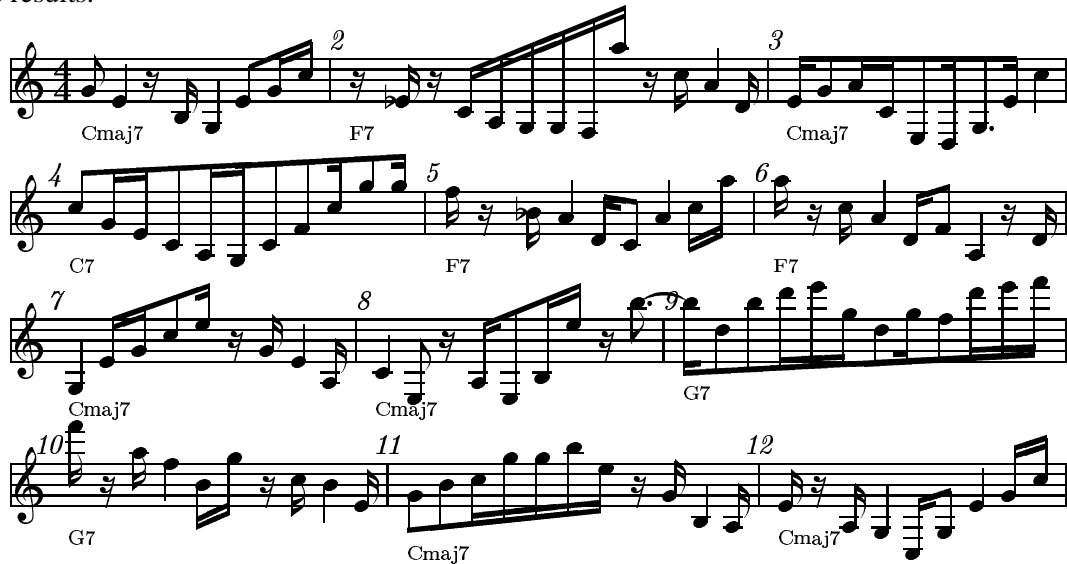
Table 1: Preferences for the melody generation experiments

## 4 Summary of Results and Evaluation

The GA converged very quickly to high fitness because of the domain specific genetic operators and the restricted representation. Conversely, the generated melody line did not reach a musically acceptable standard until the restricted version of the last type of operators (above) were implemented and used. Pattern matching in the fitness function, once it was correctly weighted (see below) gave a feeling of development in the melody, as did the domain specific genetic operators. We performed three sets of experiments, as follows.

**Using local mutations** The GA performed very well, within the constraints of its encoded knowledge. The resulting melodies followed the required contour, with a minimal number of over-large intervals, illegal suspensions, long rests, and a maximal number of valid downbeat notes and valid long notes. The only exception to this is that the initial note-duration probabilities sometimes prevented the search from converging to the required speed profile. This was a shortcoming of the operators used: there was no operator which would break long notes into short ones, and so it was impossible to reach the optimal part of the search space. In spite of this success, the melodies mostly sounded like a collection of random notes (compressed in pitch spread, as the algorithm reduced large intervals). There was no feeling of theme development, as expected, because the mutations used were local.

**Using copy & operate mutations** The uninspiring musical results arising from the use of local mutations led us to implement the copy & operate mutations. The output, in most cases, is still of little musical interest, because, even if we copy similar fragments of the melody to different positions, creating a feeling of development, later mutations can destroy them. So the randomness of the GA's choices is a problem here. This was the motivation for the implementation of the restricted operators, which partly alleviate the problem. The output of the GA using these mutations sounds much better and in some cases it is impressive (see figure 2). This was the first time that there was any feeling of development (successful or otherwise) in the results.



The image shows a musical score for a melody in 4/4 time, consisting of 12 measures. The melody is written on a single staff in treble clef. The notes are: 1. C4, 2. E4, 3. G4, 4. A4, 5. B4, 6. C5, 7. B4, 8. A4, 9. G4, 10. F4, 11. E4, 12. C4. The notes are connected by a series of slurs and beams, indicating a continuous melodic line. The chord progressions are: Cmaj7 (measures 1-2), F7 (measures 3-4), Cmaj7 (measures 5-6), C7 (measures 7-8), F7 (measures 9-10), G7 (measures 11-12), and Cmaj7 (measures 13-14). The chord labels are placed below the staff, and the measure numbers are placed above the staff.

Figure 2: A melody generated using the restricted mutations

We found that using only the restricted copy & operate mutations gave a more musical feeling, because these mutations put the copied motives in sensible metric positions, which makes it easier to perceive the relation between related motives.

On adding our pattern matching algorithm into the GA's fitness function, we found that the search behaviour seemed to fit into 3 phases:

**Initial Phase:** In the early generations, the genetic operators mostly smooth out the pitches in the melody, enforcing the local characteristics of the melody required by the fitness function. Big changes between generations are unlikely, because our restricted mutations operate on a fixed length and a relatively small fragment of the melody. At this stage, it is also statistically unlikely that melodies featuring desirable repeated patterns will survive selection, because – even if a genetic operator creates similar patterns in some of the population – the resulting increase in their fitness will be smaller than that of other melodies, in which mutations have improved several characteristics at once, resulting in a large combined fitness increase.

**Transition Phase:** The initial state continues until the fitness of some chromosome(s) (or perhaps the average fitness of the population) is high enough with respect to the local fitness constraints that a mutation which introduces similar patterns is preferred over a mutation which will correct local faults. Therefore chromosomes appear which exhibit a few similar patterns.

**Final Phase:** Now the algorithm reaches its critical point. Either there will be a balance between the two kinds of fitness constraint, and hence stasis, or some members of the population will be such that by unboundedly generating similar patterns it will converge to a higher fitness, because pattern matching outweighs the other parts of the fitness function. If this happens, it is possible that the work of the mutations in correcting locally unfit melodies will be undone as a side effect of generating more similar patterns, because the overall fitness would nonetheless be increased.

To clarify this last explanation: suppose that a mutation will generate 5 similar patterns, with a fitness bonus of 20 each, but in doing so will create three undesirably large intervals, with a fitness penalty of -20, and one non-chord downbeat, also penalised at -20. The resulting fitness is  $5 \times 20 - 3 \times 20 - 20 = 20$ , which is an overall increase in fitness. The problem is exacerbated because there is no upper bound on the size of copied segments, so even one mutation can generate numerous similar patterns by copying an existing repeating sequence. One way to solve this problem would be to use a logarithmic function for the calculation of the pattern matching value instead of a linear sum.

The difficulty is to maintain balance between the creation of an adequate but not excessive number of similar patterns, and all the other characteristics of the melody. This is difficult because it depends on many parameters (duration probabilities, weights, generation number, *etc.*).

Setting an upper bound of 50 similar patterns and using a smaller weight for each pattern the results were much more encouraging (see figure 3), so we conclude that the pattern matching weight should not be high with respect to the other fitness weights. Then, the part of the fitness function which finds similar patterns will operate more as an indication that mutations are working well, rather than as an attractor to musical output consisting of numerous repetitive patterns.

Another approach, which we have not explored, would to use an even smaller weight, but with fuzzy (*i.e.*, looser) matching, which is, of course, musically plausible. However, if the weight is too small, the GA may never reach the transition stage mentioned above.

**A few more experiments** Another interesting issue is how the system sometimes achieved a constant required contour. Figure 3 shows a melody which is made up of descending followed by ascending patterns (or the opposite). The weight of the pattern matching was very small and the note probabilities were 90% semi-quaver and 10% quaver. Because of the small note duration, there were no suspensions and the the GA could do nothing to match the slow part of the required speed profile. We also see here another advantage of the restricted mutations, in connection with the use of rests and their respective musical connotations. The rests, most probably, will exist in the same metric (rhythmic) positions. In figure 3's melody, for example, there are 8 and 7 out of 12 (12 bars) rests in the downbeat and third beat respectively. Many rests on the downbeat give a funky feeling to the piece.

In one experiment, we initialised the population with quavers only, and switched off crossover operators and the concatenate mutation, which can both change the duration of notes. We were surprised to find that the GA had managed to create crotchets in order to match required speed profile. This was because the algorithm was merging contiguous rests before evaluating the chromosome – necessary because long rests

Figure 3: A melody using pattern matching. The score shows 12 measures of music with the following chords: Cmaj7, F7, Cmaj7, Gm7, C7, F7, F#dim, Cmaj7, F7, Dm7, G7, Cmaj7, Cmaj7.

Figure 3: A melody using pattern matching

Figure 4: A human's cut & paste from two GA melodies. The score shows 12 measures of music with the following chords: Cmaj7, Dm7, D#dim, Em7(b9), Gm7, C7, Fmaj7, Fm7, Bb7, Em7(b9), Ebm7, Dm7, G7, Cmaj7, Cmaj7.

Figure 4: A human's cut & paste from two GA melodies



## 5 Extensions

Many different ideas for extensions have arisen during and since the implementation of this system. A short discussion of a few of them follows.

**GA Parameters** The weights for our operator selection and fitness function were chosen and adjusted intuitively. So, with different parameters we might get better results. One way to optimise the parameters would be to use regression analysis, training the fitness function with existing acceptable melodies and demanding that they should have high fitnesses.

**Statistical Analysis** More statistical musical analysis could be added into the fitness function. For example, the difference between intervallic and linear melodies is that the average interval is larger in the former than in the latter; the average deviation from this average interval should also be small in order to preserve consistency. Data is available from statistical analyses, for example, in Järvinen (1997).

**Musical structure** We have paid no special attention to the beginning and ending of our melodies. The fitness function should test if the chromosome has valid cadences. A simple implementation might ask the user to specify where in the melody cadences are required. A valid cadence might be at a long note or a note which is followed by a long rest, probably preceded by shorter duration notes.

Another softer constraint might evaluate the individual characteristics of some scales, and promote or penalise on this basis. For example the 4th degree of the ionian and mixolydian modes are considered undesirably dissonant (Fakanas, 1990; Sabatella, 1996).

**Motif-based representation** Motives might be used instead of simple notes as the basic units of the melody. This would admit encoding of more complex rhythmic patterns such as triplets or swing. A motif might still represent one note of any valid duration, or it might represent a set of notes with a special connection. This leads naturally to using existing motifs from a real-world database drawn from the jazz theory literature (Coker et al., 1970; Steinel, 1995; Pass and Hibler, 1994), which, we suggest, would simulate human improvisation closely.

**Structured GAs** A hierarchical combination of GAs might also be useful, each GA engine operating on specific parts of the chromosome, communicating as they did so. One possible architecture might be to use different GAs which will find fit rhythms and melodies and a higher level GA will find fit combinations.

**Musical Tension** A much more ambitious extension would be to calculate the tension curve of the melody, and match it against user-specified requirements. This would be a big step forward in overcoming the lack of reasoning which the system exhibits, since the manipulation of tension by composers is cited as the main reason of meaning in music (Meyer, 1956, 1973; Narmour, 1990, 1992). Our further work will be directed to this end.

In summary, we emphasise that, the more musical knowledge we encode in the system, the better are the results.

## 6 Conclusion

Subjectively, our system often generates interesting music patterns. The results were particularly encouraging if we bear in mind the small amount of knowledge encoded in the system.

More importantly, even in this prototypical form, the system is certainly more useful to us as a research tool than an interactive GA; we have suggested also that it is a more practical musical tool.

We expect that the extensions mentioned in this paper, some of which we will follow up in future, will guide the search to more consistent and human-like musical paths.

## Acknowledgements

We gratefully acknowledge the help and advice of Andrew Tuson, Department of AI, University of Edinburgh, who helped to supervise the early stages of this project.

## References

- Biles, J. (1994). Genjam: A genetic algorithm for generating jazz solos. In *Proceedings of the International Computer Music Conference*.
- Biles, J., Anderson, P., and Loggi, L. (1996). Neural network fitness functions for a musical IGA. Technical report, Rochester Institute of Technology.
- Burton, A. and Vladimirova, T. (1997). *Applications of Genetic Techniques to Musical Composition*. Available by WWW at <http://www.ee.surrey.ac.uk/Personal/A.Burton/work.html>.
- Coker, J. (1964). *Improvising Jazz*. Prentice-Hall.
- Coker, J., Casale, J., Cambell, G., and Greene, J. (1970). *Patterns for Jazz*. Warner Bros. Publications, 3rd edition.
- Davis, L. (1987). *Genetic Algorithms and Simulated Annealing*. Morgan Kaufmann.
- Fakanas, G. (1990). *Scales in Contemporary Music*. Contemporary Music Publishing (Greek).
- Gibson, P. and Byrne, J. (1991). Neurogen, musical composition using genetic algorithms and cooperating neural networks. In *Proceedings of the 2nd International Conference in Artificial Neural Networks*.
- Goldberg, D. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley.
- Holland, J. (1975). *Adaptation in Natural and Artificial Systems*. University of Michigan Press.
- Horner, A. and Goldberg, D. (1991). Genetic algorithms and computer-assisted composition. In *Proceedings of the Fourth International Conference on Genetic Algorithms*.
- Horowitz, D. (1994). Generating rhythms with genetic algorithms. In *Proceedings of the International Computer Music Conference*.
- Jacob, B. (1995). Composing with genetic algorithms. In *Proceedings of the International Computer Music Conference*.
- Järvinen, T. (1997). *Tonal Dynamics and Metrical Structures in Jazz Improvisation*. University of Jyväskylä, Finland.
- Koza, J. (1992). *Genetic Programming*. MIT Press.
- Leman, M., editor (1997). *Music, Gestalt and Computing: studies in cognitive and systematic musicology*. Number 1317 in Lecture notes in artificial intelligence. Springer, Berlin.
- McIntyre, R. (1994). Bach in a box: The evolution of four-part baroque harmony using the genetic algorithm. In *Proceedings of the IEEE Conference on Evolutionary Computation*.
- Meyer, L. (1956). *Emotion and Meaning in Music*. University of Chicago Press.
- Meyer, L. (1973). *Explaining Music*. University of California Press.
- Narmour, E. (1990). *The Analysis and Cognition of Basic Melodic Structures. The Implication-Realization Model*. University of Chicago Press.
- Narmour, E. (1992). *The Analysis and Cognition of Melodic Complexity. The Implication-Realization Model*. University of Chicago Press.
- Pass, J. and Hibler, J. (1994). *Improvising Ideas*. Mel Bay Publications.
- Ralley, D. (1995). Genetic algorithms as a tool for melodic development. In *Proceedings of the 1995 International Computer Music Conference*.
- Sabatella, M. (1996). *A Whole Approach to Jazz Improvisation*. ADG Productions. Also available from <http://www.outsideshore.com/primer/primer/>.
- Spector, L. and Alpern, A. (1995). Induction and recapitulation of deep musical structure. In *Proceedings of the IJCAI-95 Workshop on Artificial Intelligence and Music*.
- Steinel, M. (1995). *Building a Jazz Vocabulary*. Hal Leonard Corporation.