Evolutionary methods for musical composition

Geraint Wiggins[*], George Papadopoulos[†], Somnuk Phon-Amnuaisuk[‡], Andrew Tuson[§]
Department of Artificial Intelligence
University of Edinburgh
80 South Bridge, Edinburgh EH1 1HN, Scotland

Abstract

We discuss the use of genetic algorithms (GAs) for the generation of music. We explain the structure of a typical GA, and outline existing work on the use of GAs in computer music. We propose that the addition of domain-specific knowledge can enhance the quality and speed of production of GA results, and describe two systems which exemplify this. However, we conclude that GAs are not ideal for the simulation of human musical thought (notwithstanding their ability to produce good results) because their operation in no way simulates human behaviour.

1 Introduction

In recent years, the idea of Genetic Algorithms (GAs) has generated signficant interest in the artificial intelligence and computer science communities. This has been reflected in a number of publications in the computer music world, some of which we discuss below.

However, as GA research proceeds, it is becoming clear that the operation of a GA need not be enormously different from that of a knowledge-based system. Indeed, Wolpert and Macready (1995) have suggested that for a GA-based method to be really effective, domain-specific knowledge is not just desirable, but strictly necessary.

In this paper, we set out to explore two aspects of GA applications to music:

1. the use of knowledge-rich structures and procedures within the algorithm itself, as opposed to the more traditional use of GA components which are not problem-specific;

2. the strict use of objective methods, in the sense that any reasoning encoded in the GA should be stated explicitly, rather than being implicit in the expressed opinion of a human user.

These criteria are important to us because we are interested in simulating human behaviour, and not just in achieving a musical result. So we wish to be able to examine the internal behaviour our of methods, compare them with human behaviour, and attempt to explain any discrepancies.

First, we present an overview of the structure of a typical GA. We then proceed to outline existing applications of GAs in computer music. We present two case studies of knowledge-rich musical GAs, and then draw conclusions about the implications of the work for musical GAs in general.

[*]Email: `geraint@ed.ac.uk`
[†]Email: `georgep@dai.ed.ac.uk`
[‡]Email: `somnukpa@dai.ed.ac.uk`
[§]Email: `andrewt@dai.ed.ac.uk`

2 What are Genetic Algorithms?

Genetic algorithms (GAs) are a stochastic, heuristic optimisation technique first proposed by Holland (1975). The idea is loosely based upon the process of evolution by natural selection proposed by Darwin (1859). GAs have been successful in previously difficult or intractable problems such as atmospheric pollution monitoring (Cartwright and Harris (1993)), and scheduling (Fang (1992)). Ross and Corne (1995) give a useful overview of GA applications.

For our purposes here, we merely outline the constituents of a GA, and describe a typical implementation, only briefly discussing each of the constituent parts. For more detail, see (*e.g.*) Michalewicz (1992).

A GA consists of the following components:

- A *representation* for chromosomes, the candidate solutions to the problem being solved.
- An *initial population* of chromosomes.
- A set of *operators* to generate new candidate solutions from members of the population, and information as to when they should be applied.
- An *evaluation function* to assess the *fitness* (quality) of a given candidate solution.
- A *selection method* which gives good solutions a better chance of survival.

The GA is applied iteratively, each time generating new candidate solutions from the population by the application of operators, evaluating them, and then allowing the fittest of the available solutions to comprise a new population.

We now describe the GA and its components in more detail.

The Algorithm. The following sequence of steps describes the algorithm for a GA with *steady-state reproduction*. Other methods exist; this is one of the simplest.

1. Generate an initial population of chromosomes, usually at random.
2. Apply the evaluation function to each chromosome.
3. Select *parent* solutions according to their fitness (fitter solutions are more likely to become parents).
4. Randomly pick and apply an operator to generate a new chromosome.
5. Evaluate the new chromosome and, if it is fitter than the least fit member of the population, substitute it into the population.
6. Go back to step 3 until a *stopping criterion* is reached. Examples of stopping criteria are: all members of the population are identical (*convergence*), a fixed number of evaluations have been computed, or a solution of a given quality has been found.

Representation. As with the vast majority of knowledge engineering problems, the first question to ask, once we have stated our problem, is: How do we represent the chromosomes in a form that the GA can manipulate? The GA designer must encode the required information so that correlations existing in the search space are made explicit: only then can the GA exploit them.

Practical GAs should use whichever encoding is appropriate to the problem (an encoding used by an existing method is often a good start). This might be a string of real numbers, a logical expression, a Lisp S-Expression – whatever is found to work for the problem at hand.

Operators. In the basic GA, there are two main types of operator: *crossover* and *mutation*, drawn from the biological metaphors of sexual and asexual reproduction respectively.

Each operator available to the GA has a probability of being applied (an *operator probability*). Operators may also have *parameters*, which can determine their behaviour.

Crossover is an exchange of information between two (maybe more) chromosomes in the population. This is best illustrated by a commonly-used crossover operator for binary strings.

Two-point crossover picks two points, at random, along the strings and swaps the contents of the string between those points, to produce children, thus:

$$\begin{array}{cc|cccc|cc} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{array} \xrightarrow{\text{Crossover}} \begin{array}{cc|cccc|cc} 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \end{array}$$

Crossover is deemed useful because it can bring together portions of separate strings associated with high fitness to produce even fitter children, and, conversely, bring poor portions of strings together so they can be purged from the population.

Mutation takes a chromosome and randomly changes part of it. In combination with selection (see below) this performs a search analogous to *hill-climbing* (Russell and Norvig (1995)). For a binary string, mutation involves flipping each bit in the string with a (low) probability p_m as follows:

$$0 \quad 0 \quad 0 \quad 0 \quad 0 \quad \boxed{0} \quad 0 \quad 0 \xrightarrow{\text{Mutation}} 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 0 \quad 0$$

Early GA research viewed mutation as a background operator, used infrequently in order to maintain diversity in the population. However, opinion has changed, partly due to the fact that mutation-only optimisation techniques, such as simulated annealing (Kirkpatrick et al. (1983)), can obtain comparable results with GAs. Mutation is now viewed as a search operator in its own right.

The Fitness Function provides a measure of the quality of a chromosome. Devising such a function is non-trivial, especially in the case of multi-objective solutions, where multiple fitness measures have to be weighed against each other in assessing a chromosome. Sometimes, a human opinion is used instead of a fitness function, in which case the GA is said to be *interactive* (IGA).

Population and Selection is where the other components come together. As we explained above, the GA works upon a population of chromosomes and selects the most suitable. However, there are choices to be made, such as:

- Population size: large populations have a larger and more diverse number of candidate solutions, but take longer to evaluate. The size is usually fixed within a given run.
- Spatial arrangement of chromosomes: dividing the population into groups emulates geographical speciation and helps maintain diversity between solutions.
- Replacement strategy: should an entirely new population be generated (a *generational* GA), or should children be incrementally inserted into the current population (a *steady-state* GA)?
- Selection scheme: which breeding partners and surviving children should we select?

3 Existing Work on GAs in Music

GAs have been used in music generation elsewhere. Horner and Goldberg (1991) used GAs for thematic bridging; Jacob (1995) devised a composing system using an interactive GA; Biles (1994) used an interactive GA to produce jazz solos over a given chord progression.

In harmonisation, the most directly related work to that presented here is that of McIntyre (1994) and Horner and Ayers (1995). McIntyre used a GA to generate a four part harmonisation of an input melody, focussing on Baroque harmony. Horner and Ayers focussed on the harmonisation of chord progressions using GAs.

One aim of our harmonisation project here is to investigate the potential of the GA and its performance in the musical domain. So our search is not artificially limited as in McIntyre's system (which only used a C major scale); nor is there problem abstraction as in Horner and Ayres' system, (which uses the GA to generate parts, given a chord progression, which is a significantly simpler task). Our work aims to harmonise input melodies and does not limit itself to a specific key or scale; and it works at the level of individual voices, with all the extra constraints this entails.

In terms of instrumental solo generation, GenJam (Biles (1994)) is the most closely related work to that presented here. It is a "genetic algorithm based model of a novice jazz musician learning to improvise". There is no algorithmic fitness function to evaluate the population of the distinct melodic ideas; instead, a human "mentor" gives real-time feedback, so GenJam is an IGA. Therefore, GenJam exhibits the drawback associated with all IGAs: in order to evaluate a population of potential solutions, the user must hear all of them – and there are many. Moreover, it is likely that bias will arise towards musical structures which are familiar from previous listenings. As such, GenJam can tell us little about the mental processes involved in the improvisation process. GenJam also uses a simplified mapping between the accompanying chord and the scale used for the generation of the solo, restricted the duration of the notes to be all equal. These restrictions can lead to the loss of potentially interesting solutions, containing inflections or passing notes, and rhythm interest, respectively.

In summary, while Biles (1994) reports promising results from GenJam, it might be said that the simplifying assumptions made in order to render the problem tractable have rendered the problem rather anodyne. In our solo generation project (also based on jazz harmony), we have attempted to be more general. Our use of knowledge-intensive operators has rendered this extra generality computationally tractable.

For a more complete summary of GA work in music see Burton and Vladimirova (1997).

4 Harmonising Chorale Melodies

In this section, we present the results of a study on the use of GAs in generating four-part homophonic tonal harmony for user-specified melodies. We detail the parts of the GA which are specific to this project – the reader is referred to Section 2 for other details.

4.1 Domain-specific Knowledge

The domain-specific (*i.e.,* musical) knowledge in this system is implemented in three parts of the GA:

Chromosome representations: Generally speaking, keys and chords are the main concepts in harmonisation of western tonal music. Harmonisation rules are expressed in terms of relationships between triads, and between degrees of scale within a key signature (*e.g.,* tonic-dominant, etc) but not the absolute pitch.

Therefore, in this implementation, musical information (*e.g.,* pitch, interval, time, duration) is represented after *normalisation* with respect to key – that is, absolute pitch information is abstracted out. Then, pitch is expressed in terms of scale degree. To express all twelve semitones, the standard five accidentals are used. Different octaves are distinguished by an associated integer. Finally, time intervals are represented as integers.

A knowledge-rich, meaningful representation is used in our chromosome representation (compare with the binary strings more commonly used in GAs). The representation may be thought of as a matrix. It consists of four strings of equal, fixed length. The four strings contain soprano, alto, tenor and bass part. The user inputs the soprano information (assumed to be the melody); the GA will then harmonise the input soprano, homophonically, with a further three voices according to the musical domain knowledge encoded in its operators.

Reproduction operators: both crossover and mutation operators of several kinds are used in this implementation, as follows:

Splice: One point crossover between two chromosomes – selects a point for crossover between successive notes of the melody, so the division is vertical, not horizontal.

Perturb: Mutate by allowing alto, tenor and bass to move up or down by one semitone or tone. The process is random.

Swap: Mutate by swapping two randomly picked voices between alto, tenor or bass. This gives the effect of changing the chord between different open and closed positions, and of changing inversions.

Rechord: Mutate to a different chord type. This mutation generates a new chord from the melody data. A chord is built with the soprano note as root, 3rd or 5th. Doubling (necessary for a four note chord) can be in any position.

PhraseStart: Mutate the beginning of each phrase to start with tonic root position on a down beat.

PhraseEnd: Mutate the end of each phrase to end with a chord in root position.

Fitness function: Many GA applications in the music domain use humans as a means to justify the fitness of chromosomes, in IGAs. This approach is subjective because it relies on individual preferences. Also, human listeners tend to become more open to music on repeated hearings, and are prone to other inconsistencies based on mood, attention span, and so on. So the IGA and does not allow us to study the fitness function itself, to determine how faithful it is to our chosen task. In this project, music-theoretical knowledge is used to construct the fitness functions in objective logical terms, which allows us to examine the behaviour of the system more scientifically. The fitness function judges the fitness of each chromosome according to the following criteria. Within individual voices (as opposed to between voices), we prefer stepwise progression over large leaps, and we keep the voice within its proper range. We avoid progression to dissonant chords, and we avoid leaps of major and minor 7ths, of augmented and diminished intervals, and of intervals larger than one octave.

Between voices, we apply the following criteria: we avoid parallel unison, parallel perfect 5ths, and parallel octaves; we forbid progression from diminished 5th to perfect 5th (though the converse is allowed); we avoid hidden unison; we forbid crossing voices; and, we forbid hidden 5th and octave in the outer voices, when soprano is not progressing stepwise.

Solutions are penalised for note doubling and omission, in the primary major and minor triads: doubling of Root is preferred, while doubling of 3rd is avoided; doubling of 3rd is forbidden in a dominant chord; if it is necessary to omit a voice, omit the fifth only, except in 1st inversion; in inverted chords, doubling of the bass is preferred; and we avoid doubling of tones which give a strong harmonic tendency.

In this preliminary implementation, the system does not have enough knowledge to plan for large scale harmonic progression. The fitness function determines only the plausible harmonic movement between two adjacent chords. The fitness function prefers (in decreasing order of preference): descending 5th movement; progression towards the tonic; retrogression; and repetition.

4.2 Results and Analysis

Figure 1 shows a harmonisation by our system of the first eight notes of "Joy to the world". The output is not perfect, but it is surprisingly good given the limited rules built in to the system. All the output of the system was marked by Dr. John Kitchen, a lecturer in the Department of Music at Edinburgh, according to the criteria he uses for 1st year undergraduate students' harmony. This example scored 5 out of 10 – a clear pass. While other examples were less successful (most earning around the 30% mark), this was mostly due to the lack of coherent large-scale musical progression – the system was felt to be better than student harmonisers at getting the basic rules right.

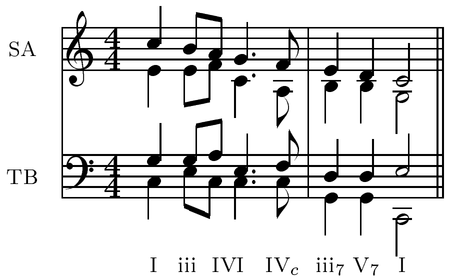

Figure 1: Harmonisation of first line of *Joy to the World*

The experiment was carried out with various GA parameter settings. It was observed that, as expected, the weights of the various penalties applied in the fitness function have a significant effect on the solution. Other parameters, such as crossover rate, mutation rate, and different selection schemes appear to affect the time taken for the population to converge, and do little for the solution quality. This is due to the fact that it is the fitness function which primarily defines the knowledge in the system, while the other parameters define the search strategy.

What is most significant is that, with the current evaluation functions and reproduction operators, the GA still cannot satisfy all the constraints within 300 generations.

An attempt with an *island model* (Gordon et al. (1992)) with four population groups was also carried out, in the hope that different population groups might be able to preserve their own salient cultures, and might be able to bring the GA to a more globally acceptable solution.

Figure 2 is a result from the island model. The experiment showed an improvement in search efficiency, but the GA still could not reduce all the penalties.

The main explanation for this is the relatively shallow knowledge of the system, in particular with respect to overall harmonic movement. A reduction of a fitness penalty in one

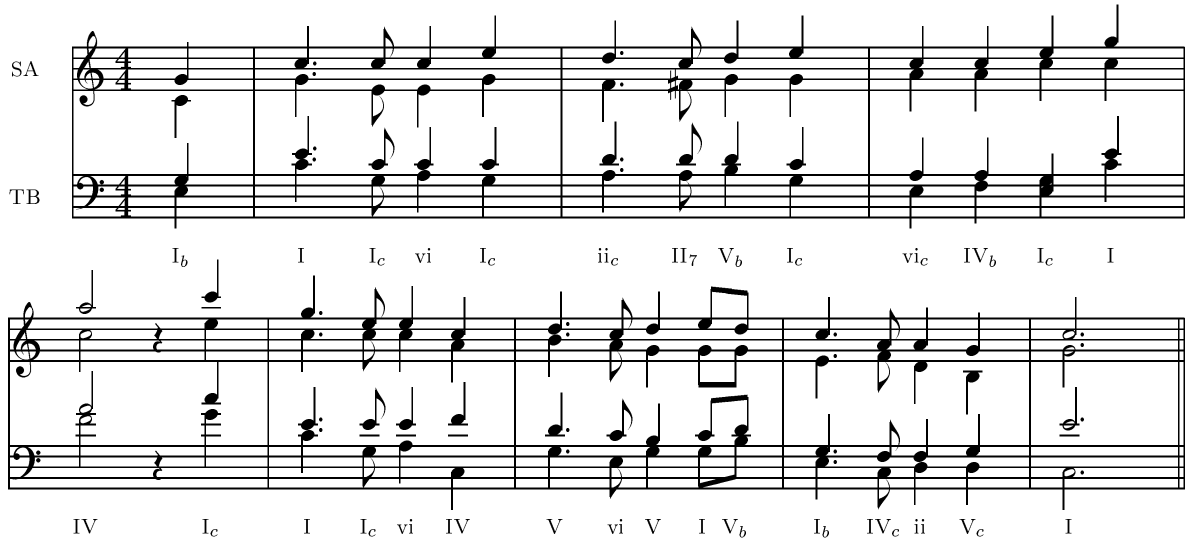

Figure 2: Harmonisation of *Auld Lang Syne*

position may increase penalties in other positions, because the movement from one chord to the next is not considered with respect to overall movement in the phrase. Because GAs work by random perturbation of their chromosomes, the high-level planning necessary to smooth out these penalty spikes is not readily encoded in them – to do so, we would have to build a much more complicated model.

We conclude, therefore, that a conventional rule-based system (perhaps in conjunction with one or more GAs) is a more appropriate method for the harmonisation task.

5 Generating Instrumental Solos

This section describes a study on generating instrumental solos by GA. Note in particular the distinction between the objective GA with a programmed fitness function, and the more subjective (and so less scientifically enlightening) interactive GA used in some similar studies. Again, we detail here the points which make this GA system different from others – see Section 2 for other details.

5.1 Domain-Specific Knowledge

Chromosome Representation: The representation of the structures (solo, chord progression) is very abstract, flexible and does not allow the generation of non-scale notes (note, though, that this does not preclude the generation of passing notes, as more than one scale can be used with most chords). The solo is generated as a list of `(degree, duration)` terms. The chord progression given as a list of `(root, chord type, duration)` terms.

Operators. The speed of convergence to high fitnesses, and the quality of results, of this system is based largely on the genetic operators. They are directed, in the sense that they contain domain-specific knowledge, and are not the general operators used in traditional GAs.

Crossover Two crossover operators were implemented: one-point and two-point crossover. The first parent belongs to the selected population and the second parent is chosen randomly either from the selected or the previous population. The fitter of the two created children goes into the intermediate population. The crossover operators work in terms of absolute time, not in terms of notes, so some care has to be take with splitting notes up to maintain a coherent flow. This process has a tendency to split notes to smaller durations. Note that these crossover operators apply no intelligence in selecting their cross-over point – this issue is discussed further below.

Mutations In this system, mutations are the operators which try to fix a flawed solo or to direct the music in ways which are pleasing and attractive to the ear.

One-note: randomly pick one note and replace it with a newly generated one or transpose it up or down by a small number of degrees.

Swap: randomly choose two fragments of the same length (number of notes) and swap them.

Transpose: randomly choose a fragment and transpose it modally by up to a perfect fifth.

Permute: randomly choose a fragment and permute its notes.

Sort ascending: randomly choose a fragment and sort its notes into ascending order of degree. Rests in the fragment remain in position.

Sort descending: as above, but sort by descending degree.

Redistribute durations: randomly choose a fragment and permute its durations, maintaining the order of the pitches.

Same rhythm: vary a random fragment by randomly transposing up to half of its notes, maintaining their durations.

Simple copy: copy one fragment and overwrite another with the copy.

Copy & mutate: a family of mutations, which copy a fragment, as *simple copy* but mutate it as per mutations above before pasting it back into the chromosome. The mutations used are *Transpose*, both kinds of *Sort*, *Redistribute durations*, and *Same rhythm.*

Concatenate repeated notes: merge any contiguous notes (in the whole chromosome) of equal pitch into one note of equivalent length. This mutation prevents boring repetition of the same note, which can be caused by the sorting mutations and splitting of notes in copying and crossover.

Fitness Function: In the current state of the art, we do not know how to implement a complete algorithmic fitness function which will direct the search towards desirable, human-like jazz solos. Here, we merely *approach* the problem, by building operators which loosely imitate the improviser's "work tools" and mental process. The final output generated on this basis is at least closer to the desired kind of output than random doodling, and, importantly, its shortcomings will inform future work.

The fitness function discussed here is based on material gleaned informally from many sources (music books, articles) which aim to explain and model the process of improvisation. It is also based on simple informal statistical analysis of jazz solos and finally on some intuitive ideas. As such, it constitutes a first approximation to a fitness function for jazz solos, and provides a point of departure for further development.

Our fitness function is a multi-objective function – that is, various different dimensions are used to measure fitness, and a vector of those dimensions is produced for each chromosome. This is more informative than a single-objective function. Here, we produce a single value by taking the weighted sum of the vector; a more general approach would be to use Pareto Ranking (Fonseca and Fleming (1994)).

The function checks eight different characteristics of the solo line, which it then uses to calculate the corresponding overall fitness. Different coefficients may be used to apply more or less significance to the different dimensions. The eight characteristics used are as follows.

Illegal jumps: A solo will tend to lose coherence if it jumps around in pitch too much, because of lack of auditory streaming.

Pattern matching: Looks for repeated pitch patterns within the chromosome, particularly on musically significant beats, and favours chromosomes which exhibit this property.

Suspensions: Because of the way the chromosome is represented, it is possible to have suspensions – notes which lie across to two consecutive chords. This part of the fitness function checks what happens to those N-1 chord changes. There are two cases: there is a good suspension, *i.e.,* the note is a member of both scales determined from the two consecutive chords; or there is a bad suspension, *i.e.,* the note is a member of the first scale but not of the second. Note that the issue of whether a suspension is dissonant or not is orthogonal to this test – the suspended chord may in principle be dissonant with either or both of the underlying chords.

First downbeat: The first beat of a bar is harmonically significant. This part of the fitness function requires that the note on the first beat be a member of the current chord, unless the scale is a symmetrical one, such as whole tone, in which case any note in the scale is allowed.

Third downbeat: As for the first downbeat restriction, but less strongly so.

Long notes: Relative length of notes in a solo contributes to its feeling of tonality. In particular, long notes which are not in consonance (in whatever terms are appropriate!) with the current chord are not generally desirable. Equally, long rests leave unsatisfying gaps in the solo. This aspect of the fitness function penalises chromosomes with these features.

Pitch Contour: The system favours close matches between the pitch contour of the generated solo and that specified by the user.

Speed: The system favours close matches between the speed contour of the generated solo and that specified by the user.

Cleary, there are many other features which might be modelled in the fitness function – for example, valid cadences. With a system such as ours, which is clearly modular, other features may be added in easily.

5.2 Results and Analysis

The solo generator GA converged very quickly to high fitness because of its domain-specific genetic operators. Pattern matching in particular was beneficial to the creation of a feeling of theme development in the solo. However, the choice of weight for pattern matching as compared with the other fitness dimensions is crucial – it easily becomes unnoticeable or outweighs all the other dimensions, resulting a boring, repetitive solo.

The results are really quite encouraging, though they are clearly amenable to improvement – Figure 3 gives an example. As with the harmonisation program, it is surprising that

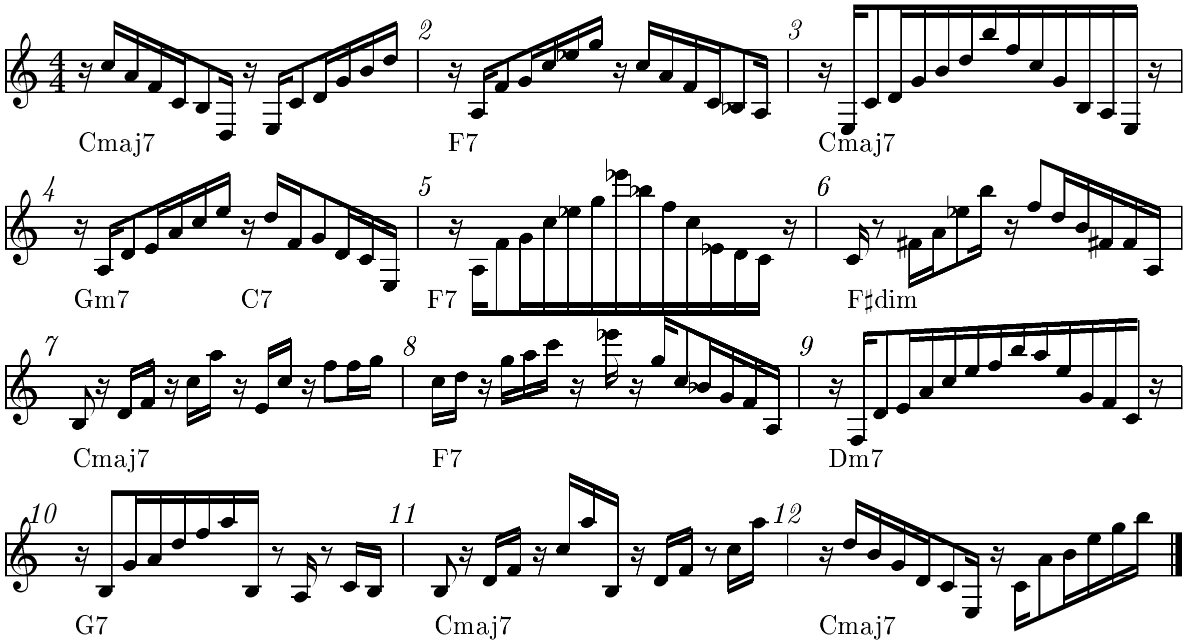

Figure 3: A solo generated from the chord sequence shown

quite acceptable results can be obtained with relatively simple rules. The example shows how the system realised the notion of constant pitch contour. The solo is made up of repeated descending and ascending patterns. The weight of the pattern matching was very small and the probability of the generated notes or rests were 90% for a semi-quaver and 10% for a quaver. In this case, sorting mutations prevailed over the other types.

6 Conclusion

It is quite clear from the experiments here and elsewhere that Genetic Algorithms can be applied successfully in the musical domain – up to a point. It is also clear that the efficacy of the GA approach depends heavily on the amount of knowledge the system possesses.

Looking at the output of our systems from an aesthetic viewpoint, the results are still far from ideal. The harmonisation produced by the GA has neither clear plan nor intention, and the solo generator, too, lacks intention, though this is less obvious in the solo context. This is not a surprise: we cannot expect large scale structure to arise from the kind of programming inherent in a GA containing relatively little domain knowledge.

In summary, we conclude that while GAs can be surprisingly good at small, constrained tasks, their performance, at least in a context of simulating human behaviour, is limited by two issues. First, GAs are a stochastic, heuristic search method, so one cannot be sure that a solution will be reached, even if there is one. Second, they lack structure in their reasoning – composers have developed complex and subtle methods over a period of centuries involving different techniques for solving the problems addressed here. Noone would seriously suggest that an author of hymn tunes works in the same way as the GA presented here, so while we may be able to produce (near) acceptable results, doing so sheds little or no light on the working of the compositional mind. In the solo generator, there is a direct attempt to address this; even so, there is still a lack of intent in the structure which renders the output less than completely musically satisfying.

Acknowledgements

Thanks to Dr. John Kitchen for his help in assessing the harmonisation system. Andrew Tuson is supported by EPSRC studentship number 95306458.

References

Biles, J. A. (1994). GenJam: A genetic algorithm for generating jazz solos. In *ICMC Proceedings 1994*. The Computer Music Association.

Burton, A. R. 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`.

Cartwright, H. M. and Harris, S. P. (1993). Analysis of the distribution of airborne pollution using genetic algorithms. *Atmospheric Environment*, 27:1783–1791.

Darwin, C. (1859). *On the Origin of Species.* John Murray, London.

Fang, H.-L. (1992). Investigating genetic algorithms in scheduling. Master's thesis, Department of Artificial Intelligence, University of Edinburgh.

Fonseca, C. M. and Fleming, P. J. (1994). Multiobjective Evolutionary Algorithms: An Overview. In *AISB Workshop on Evolutionary Computing, Leeds University.*

Gordon, V., Whitley, D., and Böhn, A. (1992). Dataflow parallelism in genetic algorithms. In Männer, R. and Manderick, B., editors, *Parallel Problem Solving from Nature 2*, pages 553–42, Amsterdam. Elsevier Science.

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems.* Ann Arbor: The University of Michigan Press.

Horner, A. and Ayers, L. (1995). Harmonisation of musical progression with genetic algorithms. In *ICMC Proceedings 1995*, pages 483–484. The Computer Music Association.

Horner, A. and Goldberg, D. E. (1991). Genetic algorithms and computer-assisted music composition. Technical report, University of Illinois.

Jacob, B. L. (1995). Composing with genetic algorithms. Technical report, University of Michigan.

Kirkpatrick, S., Gelatt, C., Jr., and Vecchi, M. (1983). Optimization by Simulated Annealing. *Science*, 220:671–680.

McIntyre, R. A. (1994). Bach in a box: The evolution of four-part baroque harmony using a genetic algorithm. In *First IEEE Conference on Evolutionary Computation*, pages 852–857.

Michalewicz, Z. (1992). *Genetic algorithms + data structures = evolution programs.* Artificial Intelligence. Springer-Verlag, New York.

Ross, P. M. and Corne, D. (1995). Applications of Genetic Algorithms. *AISB Quarterly*, 89:23–30.

Russell, S. and Norvig, P. (1995). *Artificial Intelligence – a modern approach.* Prentice Hall, New Jersey.

Wolpert, D. and Macready, W. (1995). No free lunch theorems for search. Technical report, SFI-TR-95-02-010, Santa Fe Institute.