Grammatical Music Composition with Dissimilarity Driven Hill Climbing

Róisín Loughran, James McDermott and Michael O'Neill

Natural Computing Research and Applications Group
University College Dublin, Ireland
roisin.loughran@ucd.ie

Abstract. An algorithmic compositional system that uses hill climbing to create short melodies is presented. A context free grammar maps each section of the resultant individual to a musical segment resulting in a series of MIDI notes described by pitch and duration. The dissimilarity between each pair of segments is measured using a metric based on the pitch contour of the segments. Using a GUI, the user decides how many segments to include and how they are to be distanced from each other. The system performs a hill-climbing search using several mutation operators to create a population of segments the desired distances from each other. A number of melodies composed by the system are presented that demonstrate the algorithm's ability to match the desired targets and the versatility created by the inclusion of the designed grammar.
Keywords Algorithmic composition, hill-climbing, grammar.

1 Introduction

This study introduces an algorithmic compositional system that creates short melodies from the combination of a number of melodic phrases or segments using a pre-defined context-free grammar. Each segment is related to each other according to a distance specified by the user. The user need not define any musical criteria for the compositions, merely how many segments there should be and how similar each segment should be to each other. In this way, the user can create compositions with no prior musical knowledge.

The proposed system is based on an evolutionary strategy. It is not dependent on the 'survival of the fittest' concept of traditional evolutionary algorithms as it does not measure fitness from individuals in a population but rather from the combination of segments of one single individual. Thus a segment has no merit on its own but only has importance according to how it is placed in relation to its neighbours. Once an operation is performed, a newly created segment only survives to the next generation if its inclusion improves the performance of the whole individual. This improvement is measured in relation to how the segments of the individual conform to the user-specified distance. The system employs a hill climbing evolutionary strategy with variable neighbourhood search.

The following section describes some relevant previous work in the fields of algorithmic composition and background to this work. Section 3 describes the workings of the system, specifically the Grammar, Fitness Function and Operators used. A description of the experiments undertaken is given in Section 4. The results are discussed in Section 5 along with a selection of compositions before conclusions and future work are proposed in Section 6.

2 Related Work

Algorithmic composition involves composing music according to a given set of instructions or rules. While this could refer to any hand-written set of rules used to compose, in recent years it has involved the use of computer code and in particular machine learning techniques. A comprehensive survey of computational and AI techniques applied to algorithmic composition is given in [6].

2.1 Composing with Evolutionary Computation

This system is based on an evolutionary strategy. Evolutionary techniques are well suited to creative tasks such as music composition as they are population based and inherently non-deterministic; a solution is not determined outright but found by stochastically combining and altering high-performing solutions. Various EC methods have been used previously for melodic composition. GenJam is a well-known system that uses a Genetic Algorithm (GA) to evolve jazz solos and has been used in live performances in mainstream venues [1]. A modified GA was used in GeNotator [19] to manipulate a musical composition using a hierarchical grammar. More recently, GAs were used to create four part harmonies without user-interaction or initial material according to rules from music theory [5]. Genetic Programming (GP) has been used to recursively described binary trees as genetic representation for the evolution of musical scores. The recursive mechanism of this representation allowed the generation of expressive performances and gestures along with musical notation [4]. Grammars were used with Grammatical Evolution (GE) [2] for composing short melodies in Elevated Pitch [16]. From four experimental setups of varying fitness functions and grammars they determined that users preferred melodies created with a structured grammar. GE was again employed for musical composition using the Wii remote for a generative, virtual system entitled Jive [17]. This system interactively modified a combination of sequences to create melodic pieces of musical interest.

The proposed system is a development of earlier experiments that used GE to create short novel music pieces. These experiments employed a grammar to create individuals consisting of notes, turns, chords and arpeggios and measured the fitness of each individual according to a statistical measure of the resultant tonality or the Zipf's distribution of a number of musical qualities [12, 11]. An interesting observation from these studies is that when similar (but not identical) individuals were concatenated together, certain themes or motifs emerged giving a new musicality to the composition. This was possible to exploit using GE as such evolutionary methods use a population of solutions, resulting in a number of highly fit individuals at the end of a run. For the current system, it was decided to exploit this relationship further, taking a measure of similarity between the musically mapped individuals as a fitness function for the entire population. This has been developed into a hill-climbing system with the individuals in GE replaced by segments and the population replaced by one complete individual.

2.2 Musical Fitness

If an EC system does not use an interactive fitness function, one of the most difficult aspects of using the system to compose music is in designing the fitness function; how can we attribute numerical merit to one melody over another? The given system exploits the aesthetic aspect of repetition within music. Repetition in music has been shown to have a profound affect on the enjoyment of music, both in the repetition of full pieces [7, 13] and in the analysis of form and meaning within music [15]. Repetition was also found to be an extremely useful aspect of using EC in previous experiments [12] leading to a focus on this quality for the given system.

For these compositions it is the variation on a theme — segments of a piece that are partly repeated or share similarities in some respect — rather than an exact repetition of a phrase that is desired. To develop such a fitness function a method of measuring the distance between two given melodies is required. Many studies have looked at musical similarities, most concentrating on similarities in the audio signal [10, 18]. Others, such as the current system, only consider the MIDI values of the notes. Some studies have used edit distances for such measurements. A probabilistic method on melodic similarity was used with a designed edit distance for query by humming tasks [8]. Measures of similarities between musical contours have also been considered [3].

The musical distance used in this system is measured from a step-wise list of the contour of the pitches in the segments as discussed in the following section.

3 Method

This section describes these three processes used in this system to compose music: the grammar, fitness measure and operators.

3.1 Grammar

Context free grammars can be used to map information from one domain into a more meaningful domain, allowing the user to develop a representation suitable for the problem at hand. Such a grammar is employed in these experiments similarly to how they are used in evolutionary methods such as GE, whereby the genotype — a variable length integer string known as codons, is mapped to the phenotype — a command language that is interpreted into MIDI notes. A grammar in Backus-Naur Form (BNF) is used, whereby a given expression is expanded according to a series of production rules. The choice of each rule is determined by the current integer codon:

$$\text{Rule} = (\text{Codon Integer Value}) \mathit{mod} (\#\text{ of choices}) \tag{1}$$

The creative capabilities of grammar-based methods come from the choices offered within the mapping of the grammar. The grammar used in this system expands the genotype into the description of a number of musical events — notes, chord, turns and arpeggios. The grammar and a brief description of its use is shown below. A more detailed description of the inner workings of this grammar can be found in [12].

```
<individual> ::= <piece>|<piece>|<piece><transpose>
<piece> ::= <note><note><note><note><note>
<note>::= 111,<style>,<oct>,<pitch>,<dur>,

<style>::= 100|100|100|100|100|100|100|100|50,<chord>|50,<chord>|
       50,<chord>|50,<chord>|70,<turn>,100|80,<arp>,100
<transpose> ::= 90,<dir>,<TrStep>,
<TrStep> ::= 0|1|2|2|3|3|4|5|5|5|6|7|7|7|8|8|9|9|10|10|11
<chord>::= <int>,0,0|<int>,<int>,0|12,0,0|<int>,0,0|<int>,0,0
       |<int>,0,0|<int>,<int>,<int>
<turn>::= <dir>,<len>,<dir>,<len>,<stepD>
<arp>::= <dir>,<int>,<dir>,<int>,<ArpDur>

<int>::= 3|4|5|7|5|5|7|7
<len>::= <step>|<step>,<step> |<step>,<step>,<step>
       |<step>,<step>,<step>,<step>|<step>,<step>,<step>
<dir>::= 45|55
<step>::= 1|1|1|1|1|2|2|2|2|2|2|2|2|3
<stepD>::= 1|2|2|2|2|2|2|4|4|4|4|4|4
<ArpDur>::= 2|2|2|4|4|4|4|4|8|8
<oct>::= 3|4|4|4|4|5|5|5|5|6|6
<pitch>::= 0|1|2|3|4|5|6|7|8|9|10|11
<dur>::= 1|1|1|2|2|2|4|4|4|8|8|16|16|32
```

This grammar results in an individual piece of music that may be transposed up or down a given interval. The piece is comprised of five note events each of which can either be a single note, a chord, a turn or an arpeggio. A single note is described by a given pitch, duration and octave value. A chord is given these values but also either one, two or three notes played above the given note at specified intervals. Turn results in a series of notes proceeding in the direction up or down or a combination of both. Each step in a turn is limited to either one, two or three semitones. An arpeggio is similar to a turn except it allows larger intervals and longer durations. The application of this grammar results in a series of notes each with a given pitch and duration. The inclusion of turns and arpeggios allows a variation in the number of notes that are played, depending on the production rules chosen by the grammar. The use of such grammars allows the introduction of a bias by including more instances of one choice over another. For example, a tone (value 2) is the most likely choice for the `<step>` rule above as there are more instances of that choice available.

3.2 Fitness Measurement

The fitness of the individual is calculated after the grammar has been employed to expand each segment into a series of notes. Fitness is taken as a measure of how close the segments fit to a pre-chosen pattern within a metric space.

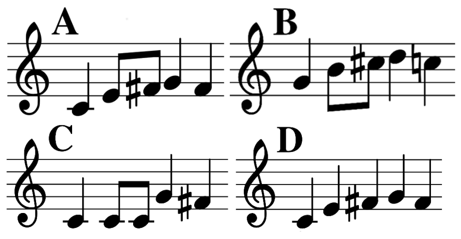

Fig. 1: Four variations on a melody

To consider the distance between two segments, the relationship between the pair is examined at every time-step. The pitch line is expanded for each segment to give a pitch value at each demisemiquaver (duration of 1 in the grammar). For example, if there is a crotchet (duration 8) played at D (pitch 2) this is represented as a list of 8 values of 2. This results in a list for each segment that indicates the pitch of that segment at every moment of duration. In the case of a chord, only the root note of the chord is considered. These experiments encourage the use of transpositions of melodies; melodies that are alike apart from a pitch shift are to be deemed equal. To achieve this, each pitch vector is normalised to start at 0. In this manner it is a *pitch contour* that is being examined, rather than the actual pitch values. The distance between two segments is then taken as the sum of the absolute distance between their pitch contours at each time step. If one contour is longer than the other, the difference in length between the two is multiplied by 5 and added to the distance between them. As there is a maximum distance of 11 semitones between two lines at any point, the value of 5 was chosen as the median of this pitch interval. This heavily penalises segments of different lengths to account for possible pitch differences as well as the length difference when notes are absent. To illustrate this, consider the four musical measures depicted in Figure 1. As described, these contours are calculated as:

ContourA	[0 0 0 0 4 4 6 6 7 7 7 7 6 6 6 6]
ContourB	[0 0 0 0 4 4 6 6 7 7 7 7 5 5 5 5]
ContourC	[0 0 0 0 0 0 0 0 7 7 7 7 6 6 6 6]
ContourD	[0 0 0 0 4 4 4 4 6 6 6 6 7 7 7 7 6 6 6 6]

The differences between each variation to the original melody A is given by:

Diff A-B	sum[0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1] = 4
Diff A-C	sum[0 0 0 0 4 4 6 6 0 0 0 0 0 0 0 0] = 20

Diff A-D sum[0 0 0 0 0 0 2 2 1 1 1 1 1 1 1 1] + (5x4) = 32

From these measurements, Melody B is most similar to Melody A, followed by C and finally D. Using this distance metric it can be ensured that melodies which stray in pitch but maintain a similar rhythm (such as Melody A & B) are not considered as different as those that differ in rhythm (such as Melody A & C). Two melodies which are identical other than an early timing difference will be measured as having a large dissimilarity between them: a 'ripple effect' of the timing difference will cause all following notes in one melody to be offset from their matching notes in the other. We acknowledge that this is only one possible measure of melodic similarity and are considering alternative methods. The use of dynamic programming algorithms such as the Levenshtein edit distance [9] were also considered for this purpose, although early experiments did not produce encouraging results with this system.

The distance between each pair of segments in the individual is measured and compared to an ideal list of measurements to determine fitness. We acknowledge that the space within which the distances are measured is an abstract metric space. However, it is useful for the user to visualise these distances in a two-dimensional geometric space. For example, consider five segments [a, b, c, d, e] equally spaced in a pentagonal shape as per the diagram in Figure 2. We know that in a pentagon the length of a diagonal e.g. (a,c) is approximately 1.6 times the length of a side e.g. (a,b). The spacing of the segments can be completely described by specifying a distance between each pair of points:

	a	**b**	**c**	**d**	**e**
a	0.0	1.0	1.6	1.6	1.0
b	1.0	0.0	1.0	1.6	1.6
c	1.6	1.0	0.0	1.0	1.6
d	1.6	1.6	1.0	0.0	1.0
e	1.0	1.6	1.6	1.0	0.0

As this is symmetrical about the diagonal, a unique description of the distances between all points can be given by the upper triangle of this matrix collapsed into the row vector. Hence the pentagon shown can be described by the distances [1, 1.6, 1.6, 1, 1, 1.6, 1.6, 1, 1.6, 1] or a scalar multiple of this vector. The fitness of an individual is measured by defining a list of ideal distances such as this and taking the absolute error from the actual measured distances to these ideal distances for each pair of segments. The fitness of the individual is calculated as the mean squared error (mse) of this list of errors.

Each individual is defined as an ordered list of segments. Due to the method of calculating fitness, the ordering of these segments is extremely important. Re-ordering the given segments is one of the simplest yet most powerful operations that can be implemented on the individual melody, as described in the following section.

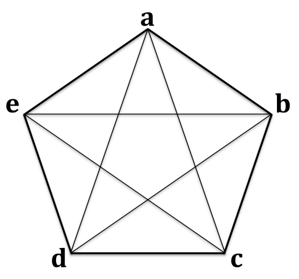

Fig. 2: Five segments equally spaced in a pentagonal arrangement

3.3 Operators

In each generation there are five possible operators performed on the individual: Switch, Permute, Crossover, Mutate and Copy. Each of these are performed in succession until an improvement on the individual is found. Once the fitness improves, the individual is updated and the next generation is started.

Switch The Switch operator switches the ordering of each pair of segments. A new individual fitness is measured for each switched pair and the best fitness obtained is recorded.

Permute The Permute operator makes one random permutation of all of the ordered segments. The ordering and fitness of the new individual is noted.

Crossover The Crossover operator selects each pair of segments within the individual and performs typical crossover on them, resulting in a new pair of segments. Again the new individual fitness is measured and recorded for each crossed pair and the best fitness is noted.

Mutate The Mutate operator selects each segment in turn and performs a mutation on it. The mutation rate is initially set to 0.01 giving a 1% chance that any given codon will be mutated. The new individual and fitness is measured and recorded for each mutation.

Copy The Copy operator picks two segments at random and replaces one with a copy of the other. The new individual and fitness are noted.

Crossover and Mutate are typical operators in any evolutionary system. Switch and Permute were developed for this particular system as the ordering of the segments has been shown to be very important. Copy was included as repetition or variation of a segment may be very important in the system. These operators are implemented in order of how much effect they have on the individual. Switch merely switches the order of two segments and so has least effect on the the individual. Permute has more effect in that it can re-order all segments, but it still does not introduce any new material to the individual. Mutate and Crossover both introduce new material. Mutate replaces one segment, whereas Crossover combines two segments into two new segments. The degree to which each of these operators has an effect on the content of the individual is dependant on the mutation rate and the selection point for crossover. With a high mutation rate a single mutation can cause a large difference in a segment. Conversely if the crossover location is towards the end of the genome, much of the initial segments may remain unchanged. Hence Crossover is considered to have less effect on the individual, although this is dependent on the specific operation. Copy completely removes one segment from the individual without introducing any new material and is hence considered to have the largest effect on the individual.

With these operators applied in succession, the algorithm continually implements changes of a small step size until this small step is no longer effective. Only then can it make a larger change to the individual. Once any change is made, the algorithm returns to checking each Switch again for a small improvement. In this way, the algorithm systematically searches a small step — or the local search space — before searching further away for a better performance. This operates in the manner of a Variable Neighbourhood Search (VNS) whereby a local minimum is found before the search moves out to a wider neighbourhood [14]. The operators that perform multiple operations (Switch, Crossover, Mutate) systematically try each operational possibility and pick the best improvement as the outcome. Conversely Permute and Copy are only given one chance in each generation to improve the fitness. Permute is used this way as it is considered an extension of Switch. Similarly, Copy is only used once at the end of a generation when no other operator has improved the fitness.

4 Experiments

This section discusses the design choices considered in implementing the system.

4.1 Melody Shapes

For a user-friendly system, the user must be able to easily specify distances between segments. The simplest way to define such distances is graphically. A 'Music Geometry GUI' was created that allows the user to specify the number of segments and their placements in a two dimensional square. The user can place any number of points within the square and a numerical distance (integer) between each pair of points is returned. The target vector in the fitness function is this list of distances. The length of the composed melody is dependant on both the number and length of segments used. The number of segments in an individual melody is controlled by the user as the number of points she plots. The length of each segment can be controlled by adding more instances of `<note>` in the grammar. In either method of elongating the composition, more calculations are involved and hence a longer run time is required.

Four separate shapes were defined using the Music Geometry GUI. The design of these shapes is shown in Figure 3. Cluster is the simplest shape containing only six segments split into two distinct clusters. Circle contains nine segments that traverse in a circular shape before returning to where they started. Cross also contains nine segments, whereby each alternative segment returns to the original starting point, forming a cross shape. Line contains twelve segments at similar distances moving linearly away from the starting point.

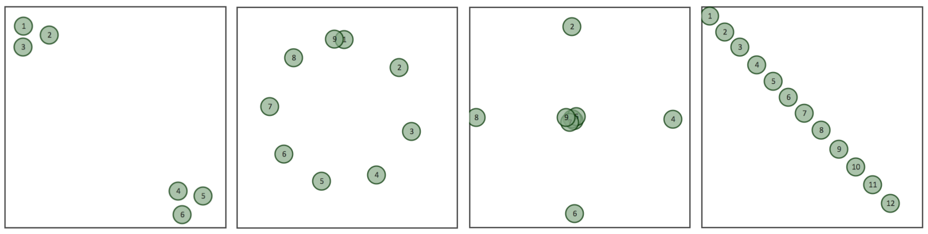

Fig. 3: Shape Targets for the Cluster, Circular, Cross and Line melodies as created by the Music Geometry GUI

4.2 Experimental Variations

Each experiment was run 30 times on all shapes with five set-ups.

Random Search To confirm the search operators were effective on the given problem, each experiment was compared against a random search for the desired shape. For this, a new individual was initialised and evaluated a given number of times, recording the best fitness achieved. To ensure a fair comparison, the number of evaluations was adjusted according to the number of segments in each shape. For the Cluster melody, six segments results in 15 evaluations for both the Switch and Crossover operators, six evaluations for the Mutate operator and Permute and Copy will make one evaluation each resulting in up to 38 fitness evaluations per generation. Likewise for the Circle and Cross individuals, both with nine segments, there can be up to 83 evaluations (36+36+9+1+1) and Line individuals could require up to 146 (66+66+12+1+1). As each experiment was run for 1,000 generations, individuals were initialised and evaluated in the random search for Cluster, Circle, Cross and Line 38,000, 83,000, 83,000 and 146,000 times respectively.

Mutation Coefficient The first experiment incorporates all operators with the Mutation Rate (μ) set to 0.01. This is a typical μ value in GE experiments employing a BNF grammar with an initial number of 100 codons. A higher μ typically results in random-like behaviour in GE experiments due to the ripple effect in which changes in the genome can have a very destructive effect on the phenome. Nevertheless, in preliminary studies it was found that using a higher μ in these experiments was leading to better final fitness. Mutation operates in a different manner in this system as regardless of the value of μ, only one segment is mutated, hence only part of the individual is affected. As a second experiment μ was increased to 0.1 and melodies for each shape were evolved again.

Limiting Operators The final operator used in any run is Copy. This operator removes one entire segment without introducing new material so it could be very destructive. Conversely, for patterns that require two segments to be very close or identical, Copy may be very beneficial. To examine the effectiveness of Copy within the system each experiment was run again without the use of Copy.

This results in a total of five experiments: Random, All Operators with μ=0.01 (AllMu01), All Operators with μ=0.1 (AllMu1), No Copy with μ=0.01 (NCMu01) and No Copy with μ=0.1 (NCMu1).

5 Results

A selection of melodies produced by this system can be listened to at `http://ncra.ucd.ie/Site/loughranr/EvoMUSART_2016.html`. The fitness results for each melody shape both for all four variations of the system and for random search are shown in Figure 4. The fitness results shown are the average of the best fitnesses achieved over 30 runs. These figures demonstrate a consistent performance for each experimental variation on each melody pattern. It is clear that each version of the system achieves better fitness than random search for each melody pattern. It is also evident that the AllMu01 and AllMu1 experiments converge faster than the methods that do not include the Copy operator. This demonstrates that the Copy operator is very important in converging to a good solution with this system. NCMu01 performs worst across each shape, implying that without the Copy operator a high μ is very important in traversing the search space to find a good solution.

From the fitness curves presented, it appears that AllMu01 and AllMu1 display a very similar fitness performance. To confirm their performances, the best final fitnesses achieved after 1,000 generations were examined. The average best fitness achieved at the end of each run is shown in Table 1. From this table it is evident that AllMu1 is the highest performer of each of the versions of the system. This indicates that unlike in standard GE experiments, a higher μ of 0.1 is more beneficial to the system than the standard 0.01.

It is evident from both Figure 4 and Table 1 that the Cluster melodies converge faster and achieve better final fitness than the Cross and Circles melodies which in turn perform better than the Line melodies. This is as to be expected as the fitness measure is calculated from the distance of each segment to an ideal, hence individuals with a larger number of segments will take longer to converge and have higher (worse) fitnesses. Even so, all methods do converge, demonstrating that the system can find solutions that approach the optimum.

5.1 Analysis of the Operators

As described in Section 3 the operators are applied according to a variable neighbourhood search leading to a difference in the number of fitness evaluations in each generation. Table 2 displays the average number of evaluations actually performed for each shape[1]. If after all operators have been performed there is no

[1] Note that the NC methods use less as each Copy in each of the 1,000 generations requires one evaluation

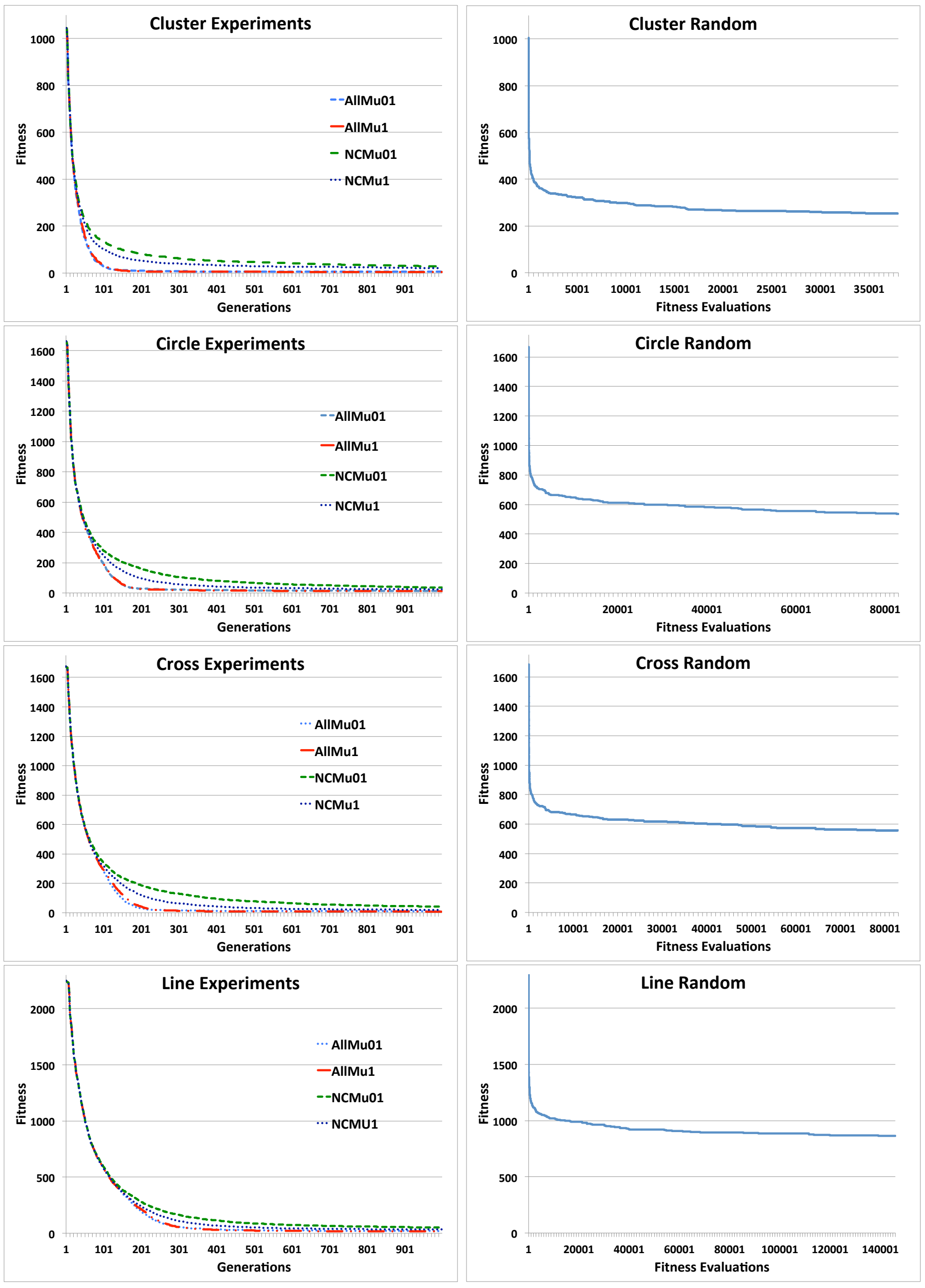

Fig. 4: Average best fitness for each experimental run and Random Initialisation for each melody shape. Note scales between shapes will differ due to different number of segments in each shape.

Method	AllMu01	AllMu1	NCMu01	NCMu1	Random
Cluster	5.37(1.9)	**4.49**(1.8)	28.79(11.4)	21.15(15.0)	252.52(26.6)
Circle	13.23(4.5)	**12.78**(3.15)	35.88(14.5)	23.48(8.26)	535.861(54.7)
Cross	10.54(5.7)	**7.55**(3.4)	40.08(18.7)	17.89(11.3)	556.21(54.3)
Line	19.14(5.9)	**15.44**(3.7)	50.45(18.8)	32.73(13.3)	864.48(65.7)

Table 1: Average (over 30 runs) best Fitness after 1000 generation achieved with each method for each pattern. Standard deviation is shown in parenthesis.

Method	Max	AllMu01	AllMu1	NCMu01	NCMu1
Cluster	38,000	36,986	36,911	35,666	35,626
Circle	83,000	78,777	78,677	76,765	77,086
Cross	83,000	77,630	76,973	75,292	75,169
Line	146,000	131,618	130,788	128,695	128,998

Table 2: Average number of fitness evaluations over 1000 generations actually performed by each method for each pattern

improvement, the current individual is kept until the next generation, resulting in the maximum number of fitness evaluations for that generation. The initial improvement in fitness shown during the evolution of all melodies demonstrates that the operators are successful and hence there are fewer fitness evaluations in early generations than in later ones. Certain operators were found to be more useful than others. The total number of times each operator produced a improvement in fitness is shown in Table 3.

It can be seen that for each shape Switch is the most successful operator. This is to be expected as it implements the smallest change on the individual and it is the first operator applied in each generation. Often Switch is applied many times in succession as when a good Switch is encountered, the individual is updated and then each Switch combination is applied again to see if further improvement can be found. This is seen to be particularly useful at the beginning of a run. Crossover and Mutate were found to be the next most successful. Copy was already shown to be beneficial in the experiments, however it is not found to be as successful as the previous operators. This is to be expected as it is quite destructive and also it is the last operator to be checked for a fitness improvement. Permute, however, is very rarely chosen as a successful operator. There are a number of possible explanations for this. Firstly, Permute is only given one random opportunity to produce a better individual whereas Switch, Cross and Mutate all have multiple opportunities depending on the number of segments in the individual. Furthermore, Permute is implemented directly after all possible Switch operators have been tested (i.e. the Switch arrangement is optimal). Hence a further re-ordering of the segments is unlikely to have an effect. Also it should be noted that Permute is used more for individuals with lower number of segments than with higher.

Method	Switch	Permute	Cross	Mutate	Copy
Cluster	1122	18	664	687	280
Circle	1353	7	709	786	231
Cross	3500	2	1285	864	436
Line	5300	0	1987	1472	427

Table 3: Number of times each operator improved fitness with the AllMu1 experiment for each shape

It was observed that more operators were successfully used at the beginning of a run. To test this observation the number of fitness evaluations made were compared against the highest number possible for the Line melodies. As each Line melody has 12 segments, the maximum number of fitness evaluations per generation is 146. Figure 5 shows this maximum against the number of evaluations actually implemented for each of the four methods on the Line melody. This figure shows that for each method the number of evaluations increases more slowly than the maximum up until approximately generation 300 at which point it increases in line with the maximum. As expected, this change in frequency of implementation of an operator aligns with the point of convergence of these runs, at which point little further improvement is seen in the fitness.

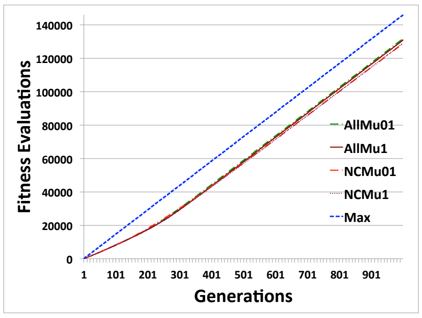

Fig. 5: Average number of fitness evaluations per generation for each of the four methods on the Line melody compared the the maximum possible number

5.2 Melodies

From produced melodies, it is clear that some shapes are easier to hear than others. Of the four shapes, we found Cluster and Cross to be easiest to identify. These melodies are repeated twice to emphasise their corresponding shapes. The similarities in the segments within the clusters in the Cluster melodies are evident to the ear. Even when these are transposed, as in Cluster2, the transposed phrases are easy for the ear to recognise. Likewise with the Cross melodies, e.g. Cross1, the return to the central segment results in a cyclical aspect to the composition. The pattern in the Circle melodies are slightly harder to discern. Each Circle melody is repeated three times to encourage the pattern to emerge. Each segment steps away from the previous but circles around and returns to the starting position. As the piece gradually returns we can hear consistencies throughout the melody. Circle3 for instance is a slow moving piece with a semibreve in each segment. Although the segments go through many changes in the cycle, this long note anchors the piece into a sense of continuity. The combination of quick runs followed by a long chord results in a similar effect in Circle1. In the Line melodies the segments can be heard to move gradually away from where they started. However, without a return to a specific point such as with the Cross and Circle melodies, or a closer variation within a group such as with the Cluster melodies, the musical quality of these Line melodies is less evident than those produced from other shapes.

When the shape is known, the authors can generally 'hear' this shape. To determine if this is recognisable to a naive listener, a series of listening tests would need to be conducted. The purpose of the system is not to exactly match such patterns however, but to create an easy to use compositional system that can create interesting melodies with a specified or implied level of repetition. The grammar used and the measurement on a pitch contour lead to a transposed meaning of what constitutes the 'same' melody, but this is purposefully introduced into the system to maintain variety and diversity in the music produced.

5.3 Discussion

Evolutionary methods such as this hill-climbing algorithm are very suitable to this type of creative problem. The stochastic element inherent to EC combined with the specified grammatical mapping can result in a variety of musical features. It is difficult to formulate a linear or deterministic algorithm to create music that offers true surprise or novelty. EC offers a rich search space that can be traversed in a variety of ways depending on the fitness function selection methods used. This particular method uses the relationship between elements of the composition as a selection criteria rather than any specific musical content, thus the user can have no concrete expectations as to what content the system will produce. Such systems offer a balance between randomness and determinism in which computational creativity may have the opportunity to flourish.

As discussed, the above experiments were run over 1,000 generations resulting in a maximum of 38,000 to 146,000 fitness evaluations per run. This may be considered small in comparison to some EC experiments and yet the converged fitnesses in Figure 4 show that this is sufficient. The lack of improvement in the Random plots however, prove that this is not a simple problem. This demonstrates that the system is efficient at finding good solutions and the operators chosen are adept at traversing the search space effectively.

The way in which the user controls the composition is somewhat abstract. The user has no control of the melodic content within the compositions. For the

Line melodies the linear decrease in the shape does not imply a decrease in pitch, merely that the segments are linearly separated in the metric space. Likewise a circular pattern doesn't imply a circle of pitches. Nevertheless, we found that controlling this placement of segments can result in interesting compositions.

The purpose of the system is not to make musical shapes, as such shapes do not have particular meaning in a musical sense. The specific shapes described in this experiment are used to demonstrate the operation of the system. The system does however offer the user a simple way of controlling, albeit in an indirect manner, the approximate length of the compositions and how much repetition or similarity is contained within the melody. Combining the distance metric with the grammar results in a one-to-many mapping; the distance metric from one shape can create a wide variety of melodies. The system can be seen as either a potential compositional tool to assist in the creation of musical ideas or a fun way for people with no musical experience to create musical excerpts without having to specify any musical qualities in relation to timing or pitch.

6 Conclusion and Future Work

A system is presented that composes short melodies using a hill-climbing algorithm with a context-free grammar. The system is driven by a fitness function based on the distances between user-placed segments. No key or time signature is incorporated into the system and the user does not require any musical knowledge. A selection of melodies created by the system are presented demonstrating that the system can create a variety of melodies that correspond to the metric distances specified by the user. The success of the evolutionary runs of the system was confirmed against random search confirming that the defined operators were extremely adept at traversing the search space to find a good solution. By employing the operators in succession a variable neighbourhood search was adopted in which the Switch operator was found to be most successful in improving fitness followed by Crossover and Mutate.

In future work, we plan to consider alternative distance metrics for comparing segments and to consider larger-scale segments that can refer to form in larger compositions. We are planning a series of listening experiments on melodies created by the system to determine if there is a correlation between the similarity of melodic segments and musical preference. The music produced by the system is dependent on the designed grammar, the implemented operators, the similarity measure used in the fitness function and the shapes chosen by the user. In a future experiment we would like to isolate these various aspects to determine their individual effects of the produced melodies. In doing so, we hope to learn more about the perception and understanding of melodic expectation.

Acknowledgments

This work is part of the App'Ed (Applications of Evolutionary Design) project funded by Science Foundation Ireland under grant 13/IA/1850.

References

1. Biles, J.A.: Straight-ahead jazz with GenJam: A quick demonstration. In: MUME 2013 Workshop (2013)
2. Brabazon, A., O'Neill, M., McGarraghy, S.: Grammatical evolution. In: Natural Computing Algorithms, pp. 357–373. Springer (2015)
3. Chen, A.L., Chang, M., Chen, J., Hsu, J.L., Hsu, C.H., Hua, S.: Query by music segments: An efficient approach for song retrieval. In: ICME 2000. vol. 2, pp. 873–876. IEEE (2000)
4. Dahlstedt, P.: Autonomous evolution of complete piano pieces and performances. In: Proceedings of Music AL Workshop. Citeseer (2007)
5. Donnelly, P., Sheppard, J.: Evolving four-part harmony using genetic algorithms. In: Applications of Evolutionary Computation, pp. 273–282. Springer (2011)
6. Fernández, J.D., Vico, F.: AI methods in algorithmic composition: A comprehensive survey. Journal of Artificial Intelligence Research pp. 513–582 (2013)
7. Hargreaves, D.J.: The effects of repetition on liking for music. Journal of Research in Music Education 32(1), 35–47 (1984)
8. Hu, N., Dannenberg, R.B., Lewis, A.L.: A probabilistic model of melodic similarity. In: International Computer Music Conference (ICMC), Goteborg, Sweden,. International Computer Music Society (2002)
9. Lemström, K., Ukkonen, E.: Including interval encoding into edit distance based music comparison and retrieval. In: Proc. AISB. pp. 53–60 (2000)
10. Logan, B., Salomon, A.: A music similarity function based on signal analysis. In: ICME, Tokyo, Japan. p. 190. IEEE (2001)
11. Loughran, R., McDermott, J., O'Neill, M.: Grammatical evolution with zipf's law based fitness for melodic composition. In: Sound and Music Computing, Maynooth (2015)
12. Loughran, R., McDermott, J., O'Neill, M.: Tonality driven piano compositions with grammatical evolution. In: CEC. pp. 2168–2175. IEEE (2015)
13. Middleton, R.: 'Play It Again Sam': Some notes on the productivity of repetition in popular music. Popular Music 3, 235–270 (1983)
14. Mladenović, N., Hansen, P.: Variable neighborhood search. Computers &amp; Operations Research 24(11), 1097–1100 (1997)
15. Ockelford, A.: Repetition in music: Theoretical and metatheoretical perspectives. Ashgate (2005)
16. Reddin, J., McDermott, J., O'Neill, M.: Elevated Pitch: Automated grammatical evolution of short compositions. In: Applications of Evolutionary Computing, pp. 579–584. Springer (2009)
17. Shao, J., McDermott, J., O'Neill, M., Brabazon, A.: Jive: A generative, interactive, virtual, evolutionary music system. In: Applications of Evolutionary Computation, pp. 341–350. Springer (2010)
18. Slaney, M., Weinberger, K., White, W.: Learning a metric for music similarity. In: International Symposium on Music Information Retrieval (ISMIR) (2008)
19. Thywissen, K.: GeNotator: an environment for exploring the application of evolutionary techniques in computer-assisted composition. Organised Sound 4(02), 127–133 (1999)