COMPRESSED MULTIDIMENSIONAL TREES FOR EVOLUTIONARY MUSIC REPRESENTATION

Abbas Pirnia, Jon McCormack

Faculty of Information Technology
Monash University, Melbourne, Australia
abbas.pirnia@monash.edu jon.mccormack@monash.edu

ABSTRACT

This paper investigates the performance and suitability of evolutionary algorithms for music composition by enhancing representation schemes. First, we argue that genetic programming (GP) is well suited to capture higher order musical structures due to its hierarchical representation. Representational enhancements are proposed on the standard GP tree: considering different branches for different musical dimensions (pitch, duration, etc.), and making use of "Automatically Defined Functions" to define reusable patterns in the generated music. Each representation scheme is described, along with the role of genetic operators in evolving compact representations. Representations are compared for their ability to evolve a population over a range of different target melodies. The results illustrate performance improvements that result from the enhancements to the basic GP tree representation.

1. INTRODUCTION

Evolutionary algorithms (EAs) are a group of methods inspired by processes from biological evolution. They have been successfully applied to many problems in search, optimisation and learning, including in the field of algorithmic music composition (see, e.g. [13, 3, 16]). Music composition can be considered a process of creative exploration and search of a musical space [7]. Any non-trivial musical space is potentially vast and its structure often unknown, so it comes as no surprise that EAs have been a popular choice for algorithmic composition. But even the performance of EAs in searching such vast spaces is of limited success in many cases [11]. In this paper, we examine a number of different representations of musical structure and test them for suitability in evolutionary music composition. A good representation will compactly and succinctly represent the most common musical structures across a variety of genres, allowing fast and efficient evolvability of musical compositions.

When designing an EA-based system, there are three main issues to consider: genotype representation, evolutionary operators, and phenotype fitness evaluation. Representation – the scheme genotypes use to build a phenotypes (compositions) – effectively determine the type of compositions evolution can produce. For example, a linear representation of individual notes cannot produce a chord. Operators determine how genotypes change during the evolutionary process, affecting whether phenotypes change slowly and gradually, or can perform large "jumps" in the space of possible compositions. Finally, fitness provides a driving force by influencing the probabilities of survival and reproduction of individuals.

We have designed a series of representation schemes and genetic operators for music composition systems based on standard genetic programming (GP) techniques [8]. In order to compare each scheme's performance, we used a simple experiment: searching the musical space defined by the representation for a variety of pre-defined target melodies and comparing each representation's ability to evolve melodies similar to the target. The edit distance of a candidate melody to the target melody was used as a fitness measure. While not as accurate as perceptual-based musical distance measures, edit distance provides a reasonable and easily computable measure of similarity between two melodies.

The next section (2) briefly examines existing musical representations for evolutionary algorithmic composition. This is followed by a discussion on music characteristics and representation considerations in Section 3. Section 4 describes the representation schemes we have developed and summarises their performance, which is followed by a brief discussion of results and conclusions in the final section.

2. REPRESENTATIONS FOR EVOLUTIONARY ALGORITHMIC COMPOSITION

An important consideration for any algorithmic representation is that each musical note has many different attributes, including pitch, duration, timbre, volume, and articulations. This raises a critical design decision: should these different attributes be grouped together as a single unit of representation, or should they be kept separate, coming together only when the representation is converted to actual music?

Research in human music perception supports a distinction between pitch- and time-based relationships. Pitch intervals and melodic contours are processed in different regions of the brain than absolute pitch [15]. In contrast, many evolutionary composition systems tend to tie different attributes of the note together, probably because they are traditionally considered as a whole (e.g. as in traditional Western notation). Dahlstedt, for instance, uses a representation based on graphs (with custom edges controlling the type of traversal) in which each leaf node contains a note or a list of notes [4]. The information stored for each note includes onset time, pitch, amplitude, duration, and articulated duration. Fu et. al. employ a genetic algorithm to compose musical phrases consisting ultimately of notes represented as *(pitch, duration, intensity)* 3-tuples [6]. Povel's Melody Generator [14] generates hierarchically organised temporal sequences of notes. Once again, all the attributes (duration, timbre, etc.) have been tied together in each note.

Somewhat differently, Biles utilises a string representation in his GA-based *GenJam*, describing it as "a cooperating, two-level, position-based, binary representation scheme". Each chromosome represents a series of eight events, which could be a new note, a rest, or a hold; one for each eighth note duration of a 4/4 measure [2]. In other words, there is no explicit value indicating the duration of each note.

3. TOWARDS A MORE DEVELOPED REPRESENTATION SCHEME

Dahlstedt divides genetic representations into three categories: basic, structural, and generative [5]. In a basic representation, such as a list of pitches and durations, the genotype essentially is the phenotype. An improvement is to incorporate structural information into the representation. For instance, musical structures can be grouped into a hierarchy (e.g. each phrase consists of motifs, each motif consists of notes, and so forth). Generative representations draw on the ability of certain processes to generate complexity far greater than their specification, a principle known in computing as *database amplification* [12].

A number of authors have described the organisation of tonal music as a hierarchy (e.g. [1, 9]). Experimental work in psychology, neuroscience, and electrophysiology supports the hypothesis of a hierarchical and modular organisation of music perception in brain [15]. The context in which a musical note is set could be much further than just a few previous notes. Distant notes or phrases are often more related to a particular note semantically than immediate neighbours. In contrast, traditional music notation, event-based sound control codes (e.g. MIDI), representations for computer applications (e.g. ABC), and many algorithmic composition representations structure music chronologically.

In contrast, structural (e.g. Melody Generator [14]) and generative representations (e.g. NEvMuse [10]) capture information about the hierarchical structure of music, potentially making them more aligned with human music perception and composition. A generative representation provides an efficient framework for encoding complex, repetitive patterns [12].

4. EXPERIMENTAL RESULTS

We begin our experiments with a generative representation based on standard GP techniques. Notes (as a collection of attributes such as pitch, duration, etc.) and a set of musical functions form the nodes of a tree which is then traversed to generate a melody. We denote this the "standard GP-Rep". Further developing this representation, attributes of notes are separated and stored on different branches under a common root of the GP tree. In this case, which we refer to as "extended GP-Rep", each branch under the root node represents a different "dimension" of the composition (e.g. one branch for encoding the pitch sequence, one for the duration sequence, or rhythm, and so on). In a further modification, we utilised *automatically defined functions* (ADFs) [8] as a means of further compressing the information captured by this representation. This design, which we refer to as "extended GP-Rep with ADFs", permits definition of reusable musical patterns, which in turn results in a compressed form of information encoding. The evolutionary algorithm for each representation is based on standard GP [8], but is different in terms of how crossover is performed, in addition to some other details, which we describe shortly.

For all representations tested, each individual encodes a melody, i.e. a sequence of notes; and each note is considered to be a $(pitch, duration)$ tuple, for the sake of simplicity. The convergence of populations to pre-defined target melodies was compared for each of the three representations. The goal was to minimise the fitness value, defined as:

$$fitness(m) = \sum_{i}^{dimensions} \frac{\delta(dim_i(t), dim_i(m))}{length(t)}$$

where the function δ returns Levenshtein distance between two arguments, dim_i returns the i^{th} dimension of the melody (i.e. pitch or rhythm), t denotes the target melody, m is the phenotype melody to be evaluated, and $length(t)$ is the number of the notes in the target melody. The parameters used for each experiment are shown in Table 1. In order to assess the performance of each representation over a variety of styles, two melodies from each of five different genres (classical, jazz, pop, folk, and nursery rhymes) were used as target melodies (see Table 4). The results for each representation show the average of 100 independent runs, i.e. 10 runs for each target melody.

Table 1. Parameters used for running the programs

Parameter	Value
Population size	500
The number of generations	700
The maximum depth of tree	10
Crossover rate	0.65
Darwinian reproduction rate	0.20
Mutation rate	0.15

Table 2. Primitives for the standard GP-Rep. The argument S, is a string of notes i.e. (*pitch*,*duration*) tuples.

Functions	
Concat(S1,S2)	concatenate S1 with S2
Repeat(S)	repeat S
ShiftUp(S)	transpose pitches up by one semitone
ShiftDown(S)	transpose pitches down by one semitone
Double(S)	double durations
Half(S)	half durations
RetroPitch(S)	Retrograde only pitches of S
RetroDuration(S)	Retrograde only durations of S
Terminals	
(*pitch*,*duration*)	*pitch*: an integer from 0 (C0) to 127 (G10). *duration*: an integer from 0 (dotted whole) to 15 (128^{th} note)

4.1. First representation: inspired by standard GP

As a first experiment, we used a representation from standard genetic programming [8], with a set of eight functions (Table 2) and notes as terminals. Figure 1 illustrates an example individual for this implementation. The three reproduction operators "Darwinian reproduction", "crossover", and "mutation" follow the standard definition from Koza [8]. This algorithm was run with the parameters in Table 1 to find melodies similar to the target melodies. Figure 4 illustrates the results averaged over 100 independent runs.

4.2. Second representation: multidimensional tree

In the second representation (extended GP-Rep), individuals have one branch under the root entry for each attribute dimension (here pitch and duration). This separation also enables us to have common functions for branches by abstracting their behaviour (e.g. *Retrograde* subsumes *RetroPitch* and *RetroDuration*, see Table 3).

Figure 2 illustrates an example of an individual in extended GP-Rep. For reproduction, standard crossover was modified to suit the multidimensional structure of the tree: for each selected pair of parents one crossover is performed for each dimension (i.e. child branches of the root node), so the number of offspring from one crossover is double the number of dimensions.

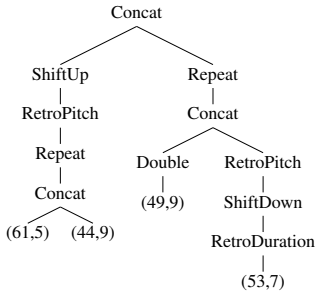

Figure 1. An example of an individual in the standard GP-Rep

Table 3. Primitives for the extended GP-Rep. The argument S, is a string of integer values, which could be either a *pitch* or *duration* sequence.

Functions	
Concat(S1,S2)	concatenate S1 with S2
Repeat(S)	repeat S
ShiftUp(S)	transpose pitches up by one semitone; halve durations
ShiftDown(S)	transpose pitches down by one semitone; double durations
Retro(S)	Retrograde S
Terminals	
pitch	an integer value (as per Table 2)
duration	an integer value (as per Table 2)

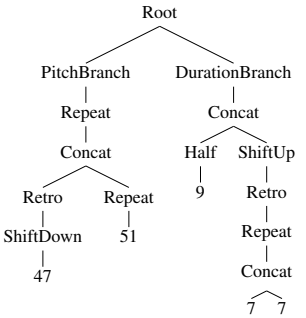

Figure 2. An example of an individual in the extended GP-Rep

As Figure 4 shows, separating different attributes of music into separate branches on the representation tree can result in a substantial improvement in phenotype proximity to the target melody.

4.3. Third representation: using ADFs

Musical information often contains repetitive patterns. As repetitions are not necessarily consecutive, the function *Repeat* cannot fully capture this feature efficiently. Furthermore, repeated patterns can be found at different levels of abstraction (e.g. intervals at a higher level abstraction from absolute pitches). Koza's automatically defined functions (ADFs) are used to evolve reusable components, which can be invoked repeatedly, typically with different inputs. In the third representation, we augmented the extended GP-Rep with ADFs as a means of capturing repetitive patterns and storing them as reusable components. In this representation, each dimension has one or more function-defining branches (ADFs), and one main result-producing branch (the RPB). An ADF is defined as a function which gets a terminal as its input and returns a sequence of terminals as output. Once defined, an ADF can be called repeatedly by the RPB with different arguments. Since one function can represent many musical phrases (which are not necessarily the same, though follow the same pattern) the representation is efficient in compressing repeated information. Figure 3 shows an example of an individual in the extended GP-Rep with ADFs. The maximum number of ADF branches on each dimension and the maximum depth of ADFs are set as program parameters. When initialising the population, ADFs are randomly generated independently (i.e. they do not call each other) for each individual from the primitives shown in Table 3. Next, the RPB is randomly generated from the union of the primitive and ADF sets. During evolution, the crossover operator is applied only to the RPB branch. If a crossover results in moving a branch which has a call to an ADF, then the ADF needs to be copied to the appropriate destination tree. The ADF is removed from the original tree if it is no longer used. Mutation operators may be applied to either of the branches, including ADFs. A mutation in a frequently invoked ADF usually results in an explorative change, whereas mutations on the RPB could be either explorative or exploitative depending on the branch undergoing the mutation.

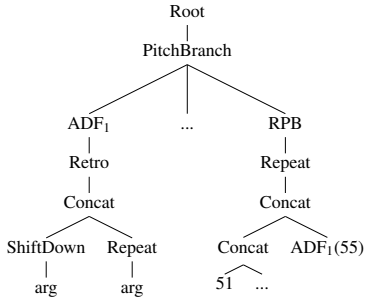

Figure 3. An example of an individual in the extended GP-Rep with ADFs which shows only the pitch branch

To compare the performance of the extended GP with-ADFs versus without-ADFs, the same set of target melodies with the same algorithm parameters (shown in Table 1) were applied. Additionally, there were two new parameters: the maximum number of ADF branches for each individual (set to 3), and the maximum depth of the ADF branches (5). Figure 4 shows how adding ADFs improves performance.

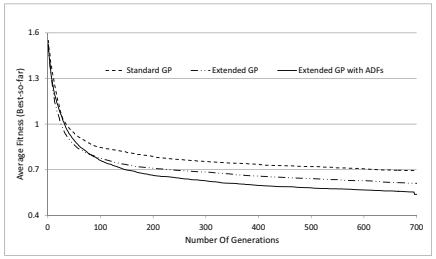

Figure 4. A comparison between "standard GP-Rep", "extended GP-Rep", and "extended GP-Rep with ADFs" on converging to the target melodies. This figure shows an improvement for extended GP-Rep over standard GP-Rep as a result of separating out different dimensions into different branches on the representation tree. Further improvement using ADFs shows up after generation 100. The results are the average of 100 independent runs for each representation (i.e. 10 runs for each target melody).

Table 4. Best individual found for each target melody. S-GP, E-GP, and E-GP* stand for standard GP-Rep, Extended GP-Rep, and Extended GP-Rep with ADFs respectively. The first value in each cell shows the average of the 10 independent runs, and the second value shows the standard deviation. The best and the worst values for each algorithm are shown in bold.

Melody Name	**S-GP**	**E-GP**	**E-GP***
For Elise (Beethoven)	0.65	0.62	0.53
	0.27	0.25	0.34
Symphony No. 40 (Mozart)	0.68	0.60	0.50
	0.37	0.16	0.29
West End Blues (Armstrong)	0.89	0.78	0.71
	0.30	0.21	0.26
Wonderful World (Armstrong)	0.73	0.59	0.47
	0.40	0.40	0.49
Hey Jude (Beatles)	**0.91**	**0.80**	**0.80**
	0.15	0.13	0.23
A Man After Midnight (ABBA)	0.73	0.65	0.64
	0.27	0.26	0.29
Turkish Folk	0.84	0.70	0.56
	0.51	0.42	0.64
Persian Folk	0.55	**0.44**	0.48
	0.19	0.17	0.27
Twinkle Twinkle Little Star	**0.47**	0.45	**0.36**
	0.35	0.28	0.39
The Farmer in the Dell	0.50	0.47	0.45
	0.27	0.27	0.32

5. DISCUSSION AND CONCLUSIONS

The standard representation suffers from the problem of its pitch and duration being tied together in one structure, the "Note". This implies there may be cases where one aspect of the music, say the duration sequence, gets quite close to the duration sequence of the target melody, while leaving much room for improvement in another aspect (i.e. pitch). For the target melody shown in Figure 5 for example, the individual in Figure 6 gets a relatively good fitness value because its rhythm closely matches the target melody, although it does not sound similar because the pitches are quite distant. In this situation, most attempts to fix the pitches would result in worse fitness, due to a single mutation or crossover effecting pitch and duration simultaneously. This situation is akin to being trapped in a local optimum. We can avoid increases in rhythm distance if we can modify pitches independently of the durations. We attempted to minimise this problem by separating out duration-related functions from pitch-related functions (e.g. *RetroPitch* and *RetroDuration*), but we found the problem inevitable because of the function "Repeat", which could not be split into separate functions.

Next, we evolved pitches and durations on different branches for each individual. Providing functions that can be applied to different aspects of music independently, allowed the extended GP-Rep to find melodies closer to the targets. Figure 4 shows how the standard GP-Rep fails to get close to the target melody, whereas the population in the extended GP-Rep converged, on average, closer to the target. The problem of standard GP-Rep sometimes getting trapped in local optima no longer exists for the extended GP-Rep. This is why the standard deviation values for the standard GP-Rep, in most cases, are greater than extended GP-Rep (refer to Table 4).

But the extended GP still did not take advantage of repetitive musical patterns, so we modified the representation again to make use of ADFs as a means of capturing reusable patterns. We use the term "pattern", not "motif" or "phrase", because, unlike using the *Repeat* function, calling the same ADF with different arguments generates different sequences, which share the same pattern. Adding ADFs allowed the representation to get closer to the target melody.

In the previous section, we noted that a mutation in a frequently invoked ADF usually results in an explorative change in the phenotype. One could say the effect of the mutation operator has been enhanced in the extended GP-Rep with ADFs. This can favour diversity, but may slow convergence. This is why the version with ADFs is behind the version without in the first generations, and needs

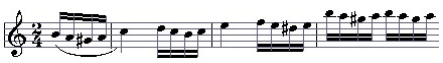

Figure 5. The beginning of the Turkish March by Mozart

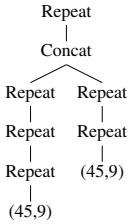

Figure 6. An example of an individual in the standard GP-Rep which gets a high fitness because of its rhythm closeness to the target melody shown in Figure 5, although it does not sound similar due to pitch differences.

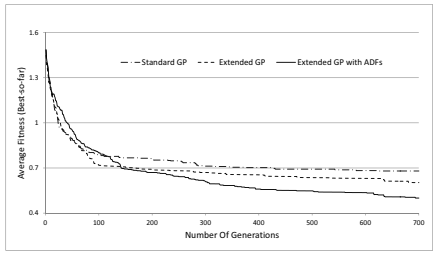

Figure 7. A comparison between Standard GP-Rep, extended GP-Rep, and extended GP-Rep with ADFs using a section of "Symphony No. 40" (Mozart) as the target melody. The figure shows how effectivly the extended GP-Rep with ADFs can represent melodies with repetitive patterns. The results are the average of 10 independent runs for each algorithm.

around 100 generations before overall improvement is observed.

As would be expected, extended GP-Rep with ADFs demonstrated the best performance for target melodies with a large number of repetitive patterns. As Figure 7 shows, when a target melody with a large number of repetitive patterns (such as "Symphony No. 40") is used, the addition of ADFs improves performance. The best individuals for this genre of music sounded very similar to the target melodies, being easily recognisable to a human listener. Conversely, the compositions without, or with fewer repeated patterns remained hard to find. Using a part of the pop song "Hey Jude" (The Beatles) as the target melody, as shown in Figure 8, ADFs did not result in significant improvement. The best individuals did not sound similar to the target melodies, and in many cases, they were not recognisable to a human listener.

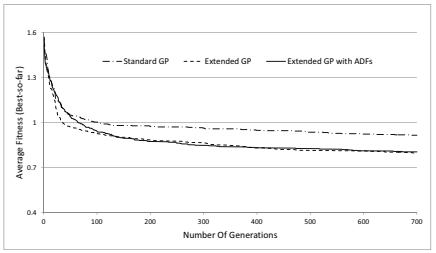

Figure 8. A comparison between Standard GP-Rep, extended GP-Rep, and extended GP-Rep with ADFs using a section of "Hey Jude" (The Beatles) as the target melody. This figure shows that ADFs do not significantly improve on the efficiency of the extended GP-Rep for melodies without repetitive patterns. The results are the average of 10 independent runs for each algorithm.

6. FUTURE WORK

The current design of the extended GP-Rep with ADFs suffers from a lack of domain knowledge. For instance, a pattern extracted from a perfectly valid phrase in a tonal composition, can generate a phrase some of the pitches of which fall outside the scale. In this case, this problem could be avoided if the representation took care of tonality. So, one possible improvement is to encode the basics of domain knowledge in the representation, such as key and time signatures. Evolution will allow these features to change as required.

Further improvements could be made to the fitness measure. In this article, we proposed a simple fitness function based on the edit distance for examining the performance of representations in finding target melodies. A more sophisticated measure based on perceptual similarity was considered too difficult to use for these experiments. This edit-distance fitness function, in its current form, can not be used extensively for music composition. A more sophisticated fitness measure would focus on subjective evaluation of compositions according to preferences, or be capable of measuring a set of well-defined, musically important features.

7. REFERENCES

[1] Bamberger, J.: Developing Musical Intuitions. Oxford University Press, Oxford, (2000)

[2] Biles, J. A.: GenJam: a genetic algorithm for generating jazz solos. In: Proceedings of the International Computer Music Conference, Denmark, (1994)

[3] Burton, A. R., Vladimirova, T.: Generation of musical sequences with genetic techniques. In: Computer Music Journal, 23(4), pp. 59-73, (1999)

[4] Dahlstedt, P.: Autonomous evolution of complete piano pieces and performances. In: Proceedings of Music AL Workshop. Fernando Almeida e Costa et al (eds.), Advances in Artificial Life, 9th European Conference, Lisbon, Portugal, (2007)

[5] Dahlstedt, P., Thoughts on creative evolution: a meta-generative approach to composition. In: Contemporary Music Review, vol. 28(1), pp: 43-55, (2009)

[6] Fu, Tao-yang, Wu, Tsu-yu, Chen, Chin-te, Wu, Kai-chu, Chen, Ying-ping: Evolutionary interactive music composition. In: Proceedings of the 8th annual conference on Genetic and evolutionary computation, Seattle, Washington, USA, (2006)

[7] Gartland-Jones, A., Copley, P.: The Suitability of Genetic Algorithms for Musical Composition. In: Contemporary Music Review, vol.22(3), pp.33-55, (2003)

[8] Koza, J. R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge, MA, USA, (1992)

[9] Lerdahl, F., Jackendoff, R. :A Generative Theory of Tonal Music. MIT Press, Cambridge, (1983)

[10] Manaris, B., Roos, P., Machado, P., Krehbiel, D., Pellicoro, L., Romero, J.: A Corpus-Based Hybrid Approach to Music Analysis and Composition. In: Proceedings of 22nd Conference on Artificial Intelligence, Vancouver, BC, pp. 839-845, (2007)

[11] McCormack, J.:Facing the Future: Evolutionary Possibilities for Human-Machine Creativity. In: The Art of Artificial Evolution: A Handbook on Evolutionary Art and Music, editor: Romero, J., Machado, P., pp:417-451, Springer, Berlin Heidelberg, (2007)

[12] McCormack, J., Eldridge, A.C., Dorin, A., McIlwain, P.: Generative Algorithms for Making Music: Emergence, Evolution, and Ecosystems. In: The Oxford Handbook of Computer Music, editor: R. Dean, pp:354-379, Oxford U.P., New York; Oxford, (2009)

[13] Miranda, E. R., Biles, J. A., editors.: Evolutionary Computer Music. Springer, Mar. (2007)

[14] Povel, D. J.: Melody Generator: A Device for Algorithmic Music Construction. Journal of Software Engineering & Applications, vol. 3, pp: 683-695, (2010)

[15] Purwins, H., Herrera, P., Grachten, M., Hazan, A., Marxer, R., Serra, X.: Computational Models of Music Perception and Cognition I: The Perceptual and Cognitive Processing Chain. In: Physics of Life Reviews, Elsevier Science Publishers B. V., vol. 5, pp: 151-168, (2008)

[16] Wiggins, G. A., Pearce, M. T., Mullensiefen, D.: Computational Modelling of Music Cognition and Musical Creativity. In: The Oxford Handbook of Computer Music, editor: Dean, R. T., Oxford University Press, pp.383–420, (2009)