SMUG: Scientific Music Generator

Marco Scirea, Gabriella A. B. Barros, Noor Shaker
Center for Computer Games Research
IT University of Copenhagen, Denmark
{msci,gbar,nosh}@itu.dk

Julian Togelius
Department of Computer Science and Engineering
New York University, NY, USA
julian@togelius.com

Abstract

Music is based on the real world. Composers use their day-to-day lives as inspiration to create rhythm and lyrics. Procedural music generators are capable of creating good quality pieces, and while some already use the world as inspiration, there is still much to be explored in this. We describe a system to generate lyrics and melodies from real-world data, in particular from academic papers. Through this we want to create a playful experience and establish a novel way of generating content (textual and musical) that could be applied to other domains, in particular to games. For melody generation, we present an approach to Markov chains evolution and briefly discuss the advantages and disadvantages of this approach.

Introduction

Some traditional works in music or lyrics generation already take into account real-world information. For instance, Colton et al.'s work creates a mood based on a newspaper article, and uses this to generate a poem (Colton, Goodwin, and Veale 2012). In general song composition process, the composer takes inspiration from his life experiences and perceptions of the world around him. This can enrich the final result, creating meaningful pieces of melody, harmony and/or stories.

Dynamic music generation in itself is not novel. Algorithmic music composition has been actively researched for the last several decades, using a large variety of approaches. Some examples include Mezzo's take into creating Renaissance style music through manipulation of *leitmotifs* (Brown 2012); the Cell-based approach (Houge 2012) used in Tom Clancy's EndWar; and the use of neural networks to create musical improvisations (Smith and Garnett 2012).

This work attempts to create lyrics from academic papers and appropriate melodies to go with them. We believe this system can also be modified to use different initial data sources, be it text sources for the lyrics or music sources for the music style. We chose academic papers as input due to their diversity and availability. Furthermore, due to their usual seriousness, it was our opinion that it would be amusing, not only for readers but also for authors, to see these works in a different light.

We believe that this system has value in being an interesting novel idea, and for creating a playful experience with something that, generally, very much lacks fun and playfulness.

We also see the proposed approach applicable in multiple areas. The most interesting for us would be in games: we think that our system (or a fork of it) could be used to improve player experience. For example, to create content for games where story is expressed through music (e.g. Karmaflow (Karmaflow) or Brutal Legend (Studio 2009)). Or by increasing re-playability and personalized content creation in games where music plays an important part in, either as ambiance or gameplay. Some adventure games even use music in small game sections to remind the player of the game's story or to provide a little comic relief moment (e.g. Deponia (**?**)).

This paper is divided in six sections. The following section (2nd) will describe background theories that we have adopted and the state of the art of research in those particular areas. The third and fourth section will present our approach for music and lyrics generations respectively, giving special attention to our algorithms' behaviours. Then we will present our results and, finally, section six will discuss these and expose our conclusions.

Background

Lyrics generation

Natural Language Generation, a sub-field of natural languages processing, has been the focus of several studies across the years. It includes creating text which is contextual, grammatical and lexical coherent, and is strongly related to poetry and lyrics generation.

One of the most important works in poetry generation uses a grammar-driven approach to create poetry, out of a given subject, that is metrically constrained. This work define three evaluation criteria to poetry generation: grammaticality, meaningfulness and poeticness(Manurung 2004). Grammaticality means that the poetry/lyrics must follow linguistic conventions dictated by a grammar; meaningfulness states that the work must convey a context or message that is understandable; and poeticness involves poetic aspects, such as rhyme and rhythm.

A different approach uses a corpus-based approach to

write lyrics about an user-specified theme (Toivanen et al. 2012; 2013). It copies a piece of text (in this case, a poem) and iteratively alters it, changing the words one by one. These words are extracted from a graph and are morphologically similar to the original. The novelty of the final piece is evaluated by calculating how many words were changed.

Oliveira's "PoeTryMe"(Oliveira 2012; Oliveira et al. 2014) uses semantic networks, generation grammars and sets of relation instances to create sentences. Nguyen and Sa generate rap lyrics, by extracting words from a database of real rap songs, and a rhyming database produces words that rhyme with the extracted ones (Hieu Nguyen 2009). Finally, they combine them into a fixed song structure.

There has been a great amount of work dedicated to create Tamil lyrics. Tamil is an old language spoken mainly in Tamil Nadu and Sri Lanka, with literature that goes back two thousand years(Suriyah et al. 2011). Sridhar et al(Sridhar et al. 2014) use the ontological meaning of a scene and a N-gram based approach to generate verses in this language. It identifies syllable patterns for the lyrics, and then create sentences that match said patterns.

Case-based reasoning has also been applied by the COLIBRI poetry generator to generate poetry from text provided by the user (Díaz-Agudo, Gervás, and González-Calero 2002). The quality of this approach results rely heavily on the quality of the original user-given text.

It is also possible to find applications online for this purpose. Country Western Song Machine[1] randomly creates country musics using a templates, and can output a very large amount of possible combinations. The Romantic Love Poetry Generator[2] uses pre-defined templates and user inputs to create poems. The words simply replace specific spaces in the template. Similarly, the Song Lyrics Generator[3] allows the user to select a style (e.g. "Freestyle" or "Love song") or an artist (e.g. "The Beatles" or "Katy Perry"), and to fill a form, that varies according to the style/artist. The form answers replace words in real music.

Our method differs from previous work in the sense that we extract structures from real songs, unlike (Oliveira 2012; Oliveira et al. 2014) extraction of words or the use of templates. Thus, we believe our system can allow for more diversity and expressiveness. Also, none of the cited works use the same input as we do (scientific papers), and very few try to parse information about the real-world into lyrics.

Music generation

Procedural generation of music content is an interesting field which has received much attention over the last decade. Examples of research on this topic range from creating simple sound effects, to avoid repeating the same clip over and over, to create even more complex harmonic and melodic structures (Shaker, Togelius, and Nelson 2014). While many games use some sort of procedural music structure, there are different approaches (or degrees), as suggested by Wooller *et al.*: *transformational* algorithms and *generative* algorithms (Wooller et al. 2005).

Transformational algorithms act upon an already prepared structure, for example by having the music recorded in layers that can be added or subtracted at a specific time to change the feel of the music (e.g., *The Legend of Zelda: Ocarina of Time* (Nintendo 1998) is one of the earliest games that used this approach). Note that this is only an example and there are a great number of transformational approaches (see GenJam (Biles 1994) and Music Sketcher (Abrams et al. 1999)), but we won't discuss them in this paper.

Generative algorithms instead create the musical structure themselves, which increases the difficulty in maintaining consistency between the music and the game events. This approach usually requires more computing power as the musical materials have to be created on the fly. An example of this approach can be found in *Spore* (Maxis 2008): the music written by Brian Eno was created with Pure Data, where many small samples created the soundtrack in real time. Also note that hybrid approaches are possible, see Experiments in Music Generation (Cope 1996)

In this project we adopt the generational approach, although limited to the generation of melodies. The motivation for us choosing this approach is that we believe we can create more novel content this way, instead of applying transformations to already existing content. Another pitfall of the generational approach is the amount of time necessary for generating the content; in our case, as the evolution of the Markov chains that will generate the melody is done *a priori*, we have a very fast (and inexpensive) generation of melodies.

Lyrics generation

The lyric generation process used in this approach takes as input an academical paper in PDF format, and output a series of verses. It has two main steps: pre-processing and lyric generation.

Pre-processing

Pre-processing involves populating databases of words (and their stems) and song structures. It needs to be executed only once, prior to the first lyric generation. Firstly, the word database was populated using Google searches for lists of word types (e.g. verbs, prepositions, pronouns). For each word in the database, its stem value was also extracted using SnowbalStemmer(Porter and Boulton 2001).

Afterwards, it was necessary to populate the structure database. By structure we define a group of word types in sequence that represent a sentence. For instance, the structure for "We see our big, blue sky" would be "Pronoun verb pronoun adjective comma adjective other". Possible values for the structure are: *verb*, *pronoun*, *preposition*, *adjective*, *adverb*, *conjunction*, *other*, *onomatopoeia*, *comma* and *dot*. "Other" represents both nouns and words that may not fall into other categories. We chose to use it, instead of "noun",

[1]Country Western Song Machine, 1998, http://www.outofservice.com/country/

[2]Romantic Love Poetry Generator: http://www.links2love.com/poem_generator.htm

[3]Song Lyrics Generator: http://www.song-lyrics-generator.org.uk/

because it allows a higher level of diversity while choosing the word. This way, not only can we choose a noun, but also any of the other categories previously mentioned as well. Onomatopoeia is an other value with less than three letters (e.g. "Po-po-poker face" would be represented as "onomatopoeia onomatopoeia other other"). These types are represented, in code, as integers.

To identify structures in real songs, a group of 50 songs were analysed. These songs were selected from famous artist (e.g. Rihanna, Michael Jackson), using as criteria that all songs need to be in English and there cannot be more than 3 songs per singer. For each sentence in the lyrics, the algorithm extracted its equivalent structure which is then inserted into the structure database.

Lyric generation

The process for generating lyrics is divided further into three steps: parsing and analysis of paper, creation of song structure, and lyrics word generation.

In the first step, the algorithm receives a PDF file containing the paper and extracts its words using the PDFBox library[4]. Then, the text is processed, removing everything before the abstract and after the references. This aims at avoiding inputting data that will not significantly improve the user's understanding of the paper. If the system cannot identify the abstract or the introduction (in the absence of the abstract), it will start at the very beginning.

In order to evaluate the importance of each word in the text, a word count is performed. It uses the stem value of the word, and is calculated as the sum of all occurrences of words derived from this stem, in the text. For instance, assume that "wait" appears once and "waiting" appears twice in text. The count would be 3 for both of them, because they have the same stem "wait". Also, each word was added to a collection of values types present in paper, according to their value type (see Section Pre-processing).

Secondly, the algorithm randomly selects a number of structures from the database. They will represent the total structure of the music, i.e. each structure will represent the structure of a line in the final lyrics. For the purposes of this paper, all songs have a total of 24 structures, divided into 6 groups of 4 structures each.

Finally, for each structure chosen, a sentence is created according to type values in the structure. *Comma* and *dot* values are translated directly into "," and ".". Types *verb*, *pronoun*, *preposition*, *adjective*, *adverb* and *conjunction* trigger a roulette selection among all words from that type that appeared in text. This selection uses the word count as probability. *Onomatopoeia* inserts either "aah", "ooh" or a random word from text with its start repeated (e.g. "ta-ta-taxonomy"). Lastly, *other* trigger a roulette selection with all words in text.

Music generation

To create a melody we decided to use two Markov chains. These are mathematical systems that undergo transitions from one state to another on a state space (Norris 1998). A Markov chain is a stochastic process with the Markov property: the next state to be selected only depends on the previous one.

Markov models can be trained using existing sequences of events (e.g., words in a book, or notes in a musical piece) and, once trained, be used to generate a new sequence of events statistically similar to the training data. It is highly unlikely for a Markov model to recreate an exact training sequence as it contains an intrinsic stochastic element. However this depends very much on the training data. An important limitation of Markov chains is that they capture statistical similarities only on a local scale, and not on a high level; this means that we lose information of structures like repetition of musical phrases in different part of the composition. Nevertheless, even with this disadvantages Markov chains have classically been extensively used for the purpose of melody generation, as they can be trained easily to create sequences of notes (Ames 1989).

Examples of research that use Markov chains and Evolutionary Algorithms are Manaris *et al.*'s *Monterey Mirror* (Manaris, Hughes, and Vassilandonakis 2011) and Bell's work (Bell 2011). Manaris' work focuses on evolving Markov chains to obtain the rare chains that will with high probability reproduce high-level structure (repetition of entire phrases or in general more structured music) while Bell's work uses interactive evolution to produce chains that create music pleasant to the listener. These are much more complex works that generate complete music and not just melody, as in our case.

There are some reasons why we have decided to approach the creation of these Markov chains through such an unorthodox method of using evolutionary algorithms (unorthodox only in this particular application of course). Using traditional (EM-based) training would have been faster and easier, yet it is in its nature to lead to an overfitting of the chain to the training set. What we hope to achieve through our approach is obtaining a chain that would reflect the characteristics of the training set while avoiding overfitting: in short obtaining a chain that reflects the characteristics of the training set while maintaining some diversity.

Another interesting feature that this approach gives us is introducing constraints through the fitness function. This gives us the option of tuning our chain in more interesting ways (of course this means in parts deviating from the training set, but that's exactly the point). These constraints could be musical rules, for example avoiding too large intervals between notes. We discuss these in the **fitness function** section.

Markov chains and Representation

In our approach we decided to use two Markov chains: one to determine the notes of our melody and another one to select the duration of these notes. Markov chains can be expanded to include some memory of the previous states by considering as state not only the current one but some of the previous ones. The amount of previous states we "remember" is called *order* of the Markov chain; if we consider a chain of order 2, it means that every state is a couple con-

[4]PDFBox is a Java open-source PDF library: https://pdfbox.apache.org/

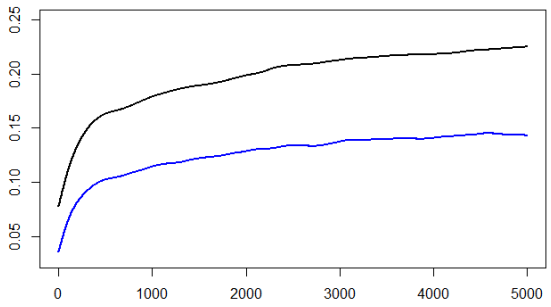

Figure 1: Fitness changes in the **notes** Markov chains population during 5000 generations. In black is represented the fitness of the best individual of the generation, while in blue is the average fitness of the population.

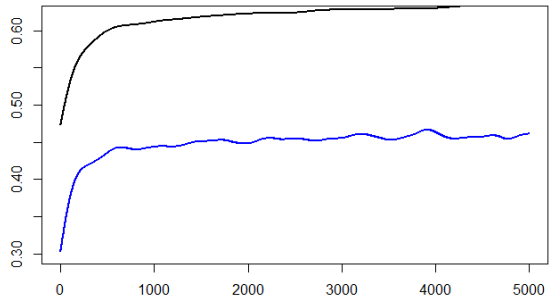

Figure 2: Fitness changes in the **durations** Markov chains population during 5000 generations. In black is represented the fitness of the best individual of the generation, while in blue is the average fitness of the population.

sisting of the previous note and the current one. In our final implementation we decided to use an order 2 chain.

We implemented a Markov chain as a hash-map. Labels (or keys) are the name of the state (the current note), and values are another hash-map containing the probabilities of choosing a note (transition) from the current state. We also adopt this hash-map as our **genotype**. You could visualize it as a labelled bi-dimensional matrix, with as labels states and transitions, the next state can be calculated as: (previous state - older note) + transition.

The state space can be calculated as $\frac{n!}{o!(n-o)!}$ where n is the amount of notes we consider and o is the order of the chain. In the case of our order 2 chain, where we consider 3 octaves (36 notes) it would be $\frac{36!}{2!(36-2)!} = 630$. To restrict this space we apply restrictions to remove states which we consider not to be good, in particular all the states that contain a transition between notes with intervals higher than an octave. To avoid leafs in our chains we do not allow for allowed states to have a 0 probability to move to any other (allowed) state.

To extract musical information without be restricted by the key of the song, we have our Markov chain for note generation work by *degrees*. In music *degree* is defined as the position of a note in a specific key's scale: for example a C can be considered differently depending what is the key of the song, in a C major song it will be a *I*st degree, while in a G major song it would be a *IV*th degree (as the scale of G would be [G A B C D E F♯]).

Evolving Markov Chains

We evolved our Markov chains using a genetic algorithm. The parameters used for our final chains are:

- Population size = 200
- Generation number = 5000
- Elitist factor = 1/4 (this means that we keep the best 1/4th of the population in the next generation)
- Mutation chance = 10%

The procedure to create the new generation is to copy to the new one the best individuals of the previous, then we fill the rest of the population with offspring of randomly selected individuals from the previous generation. Finally each individual has a chance of mutating.

To create offspring we use a one-point crossover approach: we select a random index of the states in our Markov chain (hashmap) and we create two new chains, the fist will contain the values of the first parent until the index and the values of the second parent for the following ones, while the second one the opposite. Because of the way we are representing the chains all of them always have the same amount of states, so the only thing changing while doing crossover are transition probabilities between states.

We are aware that using crossover will sometimes lead to broken Markov chains, with some orphan sub-chains that will result unreachable. This is an inherent issue with crossover, but we assume that a broken chain with high fitness will still present the characteristics that we desire and through our elitist strategy we will be able to preserve the individuals that presents good gene combination, be they broken or not. Vice versa, a broken chain with low fitness has a higher chance to be replaced.

To mutate a chain we consider a chance of $\frac{1}{numberOfStates}$ for each state to randomize it's transitions; this way we will statistically only have one state changing when mutating the chain, but still allowing for bigger mutations (or no mutation) to happen.

Fitness function

The fitness function we have chosen to apply for the evolution of the chains can be described as:

$$f = \sum_{S_i \in Songs} PredictRew(S_i) - ConstraintsPen$$

where $Songs$ is a set of melodies from existing songs, $PredictRew(S_i)$ is defined as the probability of the Markov

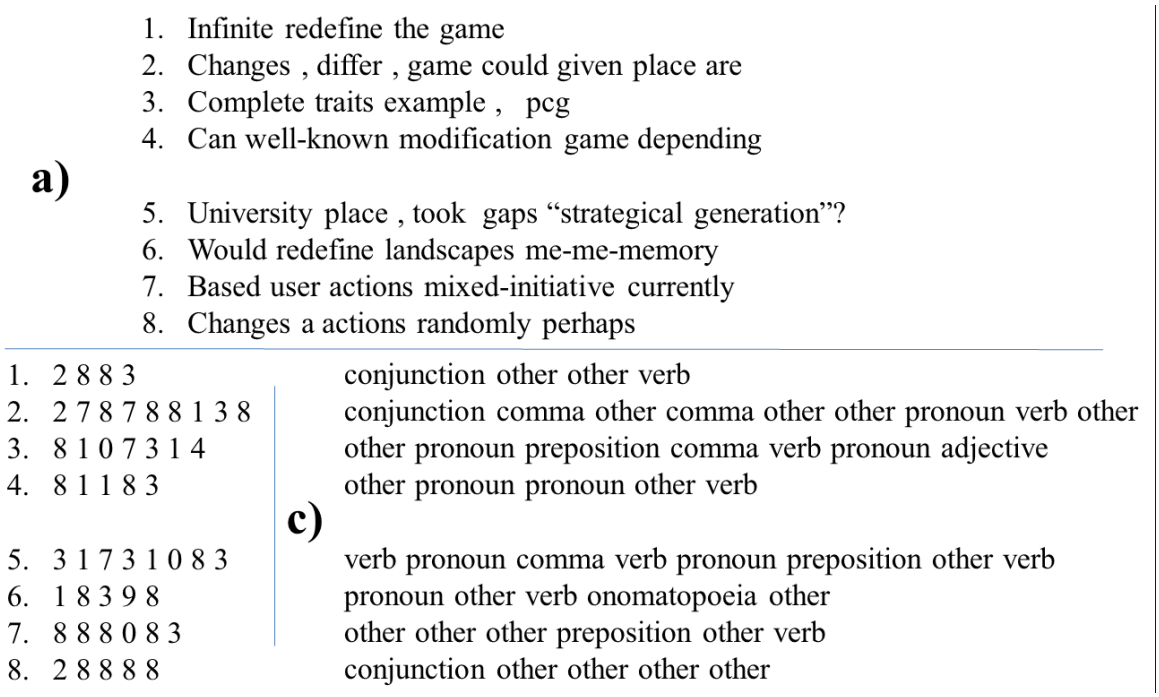

Figure 3: Output of program the program for a paper by Togelius et al.(Togelius et al. 2011). a) Actual lyric generated. b) Random structure used, represented in code. c) Same structure, in natural language.

chain we're currently evaluating to predict the melody in the song S_i and $ConstraintsPen$ is the penalty assigned to the chain according to the constraints we want to apply to it.

By considering $S_i = \{n_0, n_1, ..., n_k\}$, where n_i is the *i*-th note in the melody and $k + 1$ is the amount of notes in the melody, we calculate $PredictRew(S_i)$ as:

$$PredictRew(S_i) = \sum_{\{n_i, n_{i+1}, n_{i+2}\} \subset S_i} P(n_{i+2}|n_i, n_{i+1})$$

where $P(n_{i+2}|n_i, n_{i+1})$ is the probability that the Markov chain we are evaluating presents for the transition n_{i+2} from the state (n_i, n_{i+1}). To make a practical example, if the song presents a sequence of the type (C, D, E), the fitness of the chain will increase by the probability it has of making the transition E from the state (CD).

$ConstraintsPen$ is composed by two rules we introduced to eliminate cases we consider musically uninteresting:

$$ConstraintsPen = BigLeap + SameNoteLoop$$

where $BigLeap$ is defined as:

$$BigLeap = \sum_{n_k|(n_i,n_j)\in Chain} P(n_k|(n_i, n_j))$$
$$\text{if } |(n_k - n_j)| > 12$$

So it will increase for every transition that appears in the chain that presents a voice movement bigger than an octave (e.g. $(C1, C1) \rightarrow D2$). $SameNoteLoop$ is instead defined as:

$$SameNoteLoop = \sum_{n_i|(n_i,n_i)\in Chain} P(n_i|(n_i, n_i))$$

This way we will have a higher penalty for transitions that keep us in the same state when the state is comprised of a couple of identical note (e.g. $(C1, C1) \rightarrow C1$).

The fitness function for the chain that will determine the duration of the notes (instead than the notes themselves) is evaluated the same way, but without $ConstraintsPen$, as these constraints are pitch specific.

Our $Songs$ set consists of 20 songs taken from a list of most popular pop songs. It presents a variety of styles, but all the songs are in a major mode. This limits our generation to melodies in major mode, while for minor melodies we would have to evolve a new chain using a set of songs in minor key. This is necessary because the intervals between the notes in a major and minor scale differ, making us hypothesize that our chain will only be able to produce melodies appropriate for the key of the songs used to calculate the fitness.

The elements of the set are the voice track from the songs; we have isolated the voice melody and stored it in a MIDI file, from this file we extract the degrees of the notes of the melody (by considering in which key the song is) and the duration of these notes. We will then use these values in the evaluation of our chains (remember that to abstract the key our chain work by *degrees*).

From text to melody

To create a melody to go with some particular lyrics we our method is:

1. Find the total amount of “syllables”. In this case we con-

sider a simplistic concept of syllable: we consider a syllable for every time we encounter a vowel (groups of vowels are considered as part of the same syllable).

2. Create as many notes as the syllables in the lyrics using the notes chain
3. Define the duration of the notes using the durations chain
4. Add rests after each word (with a 30% chance that there is going to be no rest)

Finally, for easy usage and visualization of the melody we produce a midi file representing our melody.

Results

Lyrics

Figure 3 shows two verses of lyrics generated using Togelius et al. paper (Togelius et al. 2011), and it's basic structure. It is possible to notice some degree of understanding in the sentences, and diversity in word choices.

Figure 4 shows a small part from lyrics generated by the system using Darwin's paper (Darwin 1991), with melody. Another verse from the same work goes as follows:

Natural who in, throw all selection
Re-re-related nature relations law on any
Cl-cl-class that false but it inhabitants generic
It natural origin its, species to sp-sp-special and on its
.
That more be all, – reflecting
Each relations on these – natural
Each dr reflecting gr-gr-grouping circumstances
Selection introduction

Figure 5 show some verses from a song generated with this paper. In a different iteration, the following verses were generated:

Parent chain mutating
Another should, we pre-processing musical be with pr-pr-pre-processing states
That songs structures that, sridhar, other possibility use
Im-im-improve, pr-pr-priori, ad-ad-add, im-im-improved, rh-rh-rhyming, on-on-on, ooh
,
Papers to music lyrics generation, figure
Generation that, consider, generate, correct, with we is using
Restrict input and note statistical final as music
Create it, approaches, create, be, into it chains generated

Music

In this section we'll try to analyse some of the melodies our generator produced.

In figure 4 we can see an example of a melody generated by our system from Darwin's paper *On the origin of species by means of natural selection*, the generation of melodies is very fast, as the training is done *a priori*. Interesting to note is how our generator doesn't create melodies that strictly stick with the diatonic scale but introduces alterations.

In the figure we can see how in the fifth bar it lowers the VII degree to a B♭, and more interestingly how it presents the note again on a different octave. Looking at the other notes played in the chord we can recognize how the chord underlying the measure could well be a C7 with the omission of the V degree [C E B♭]. While this chord goes out of the normal key it is not uncommon to use it in this key and it doesn't necessarily signify a change of key.

Another example generated from this paper can be observed in figure 5. This score shows even more alterations than the other one with a more dissonant and almost jazz-like feel. Interesting to note how musical passages seem to emerge and be repeated with alterations: for example the succession E-D-C (bars 1, 2 and 3, with a rest in the latter) and the succession C-C-B[-C] (repeated two times in bar 4 and inverted and transposed just afterwards becoming C-C-D[-C].

Nonetheless, we haven't conducted an evaluation study on the melodies produced so we cannot make any statement on how interesting or musically pleasing the melodies are to the listener. Also we believe that to achieve a more interesting result we would need a harmonic framework to give more musical context to the produced melody; as we discussed in this section we can see some passages that seem to present some chord, but that is a purely emergent behaviour.

Discussion

This section will discuss our main findings in this project, and final considerations about them and the work in general.

Lyrics

Regarding lyric generation, although our approach may be perceived by some as simplistic, we believe it is capable of creating relatively fluid and interesting lyrics. The sentences structure seem somewhat sensible, although there are definitely space for improvement. Rhyming also happens in some moments, however it does occasionally, as the current version of the system cannot guarantee rhyming. We intend to correct it in further implementations, perhaps using a rhyming library or accessing a service online to check possible words. This would permit to create musical rhyme patterns (e.g. ABAB or AABB, where A and B represent rhyming endings of sentences). The size of sentences, too, varies, and a syllable measure constraint could help improve it.

Furthermore, it is possible to understand, to some extent, basic ideas transposed from the paper to lyrics. Some words that are clearly significant in the paper also appear in the lyrics. But there is no perceptible line of thought. It would be interesting to take the structure of the paper into account in the generation, by changing the probability value of words according to the current verse number. For instance, in the first verse, words from the paper's introduction would be more likely to be chosen than others. We would also like to try different techniques, such as an evolution strategy, to see if the outcome presents higher or lower semantic meaning in comparison to this approach. Further mechanisms for dealing with the meaningfulness of lyrics need to be applied.

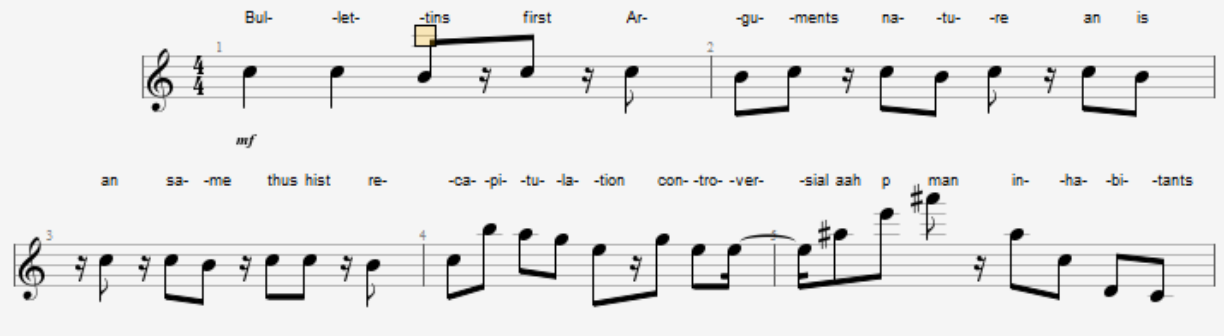

Figure 4: Excerpt from the score generated from Darwin's paper *On the origin of species by means of natural selection* (Darwin 1991) C major.

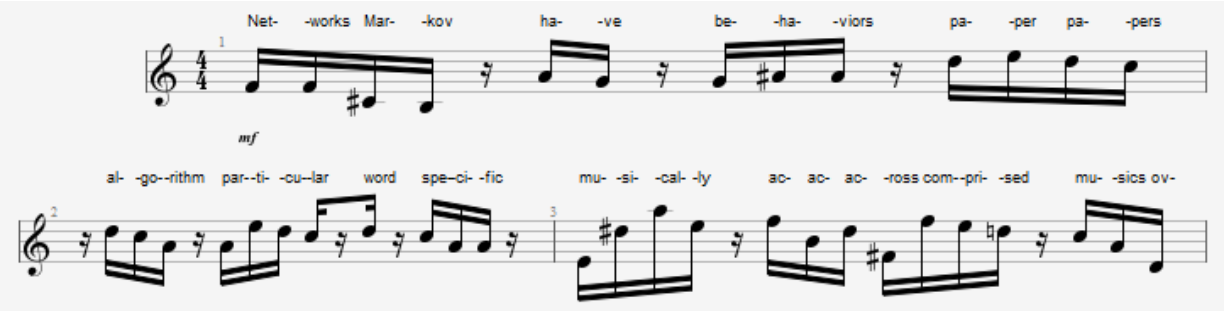

Figure 5: Excerpt from the score generated from this paper in C major.

Music

The main point we have to discuss is our choice of adopting evolution of Markov chains instead of the more common method of training them. While this method is more time consuming, we believe it is interesting. There is an argument of novelty, because the method of evolving Markov chains for music production, while not completely new, is not very explored.

As we stated at the beginning of the **Music Generation** section, we believe that this method results in lower dependency on the training set than traditional training. We think that, this way, our chains should be able to express a greater music space while maintaining some structure from the training set. One cost we expect to have to pay is a smaller rate of emulation of the training set style. Sadly, at the moment we don't have enough data to support this statement, but an evaluation study is already planned. Another pitfall is the possibility of getting in a part of the melody space where there is not enough information to create musically interesting melodies, degenerating in the worst case scenario to a random search.

As seen in section , we see some interesting emergent behaviour (like the almost key changes and the jazzier sections) which might hint to how the Markov model might not be very effective at producing a coherent whole.

Still, we believe that our approach will be able to capture the style of a specific genre/style of music with a large enough corpus of songs to use in our fitness function. We have to recognize how we might have achieved better results by having a bigger training set, but we believe we have already achieved some very interesting results.

Finally by observing the increase of the fitness function of our evolved population in Figures 1 and 2 we notice how the duration chain evolves much faster and with higher fitness. This is due to the smaller space we consider for this chain, which is less than half of the notes chain's one.

Conclusions

We have presented a method for creating melody and lyrics using real-world data. To do so, we developed a musical generator that evolves Markov chains to create melodies, and a lyric generator, that extracts content from academical papers and transforms them into songs. We have a fully functional system that complete both tasks, taking an academic paper in PDF format and outputting a melody and the according lyrics. Our generator seems to produce interesting music/lyrics combinations, but we still have to conduct further studies to prove their interestingness. The generator also still shows much room for improvement, as discussed previously, and future work will be in both fine-tuning the evolutionary approach and introducing more features in the lyrics generation, such as rhyming, stricter metric structure and improved semantic content transfer from the original paper. Still, we need to recognize that there might be issues inherent to using Markov chains for melody production that

might not be resolved, like insuring the production coherent whole.

Ultimately, we believe think these techniques might be used in music-based games to add and customize content.

References

Abrams, S.; Oppenheim, D. V.; Pazel, D.; Wright, J.; et al. 1999. Higher-level composition control in music sketcher: Modifiers and smart harmony. In *Proceedings of the ICMC*.

Ames, C. 1989. The markov process as a compositional model: a survey and tutorial. *Leonardo* 175–187.

Bell, C. 2011. Algorithmic music composition using dynamic markov chains and genetic algorithms. *Journal of Computing Sciences in Colleges* 27(2):99–107.

Biles, J. 1994. Genjam: A genetic algorithm for generating jazz solos. In *Proceedings of the International Computer Music Conference*, 131–131. International Computer Music Association.

Brown, D. 2012. Mezzo: An adaptive, real-time composition program for game soundtracks. In *Proceedings of the AIIDE 2012 Workshop on Musical Metacreation*, 68–72.

Colton, S.; Goodwin, J.; and Veale, T. 2012. Full face poetry generation. In *Proceedings of the Third International Conference on Computational Creativity*, 95–102.

Cope, D. 1996. *Experiments in musical intelligence*, volume 12. AR editions Madison, WI.

Darwin, C. 1991. On the origin of species by means of natural selection, 1859. *Murray, London*.

Díaz-Agudo, B.; Gervás, P.; and González-Calero, P. A. 2002. Poetry generation in colibri. In *Advances in Case-Based Reasoning*. Springer. 73–87.

Hieu Nguyen, B. 2009. Rap lyric generator.

Houge, B. 2012. Cell-based music organization in Tom Clancy's EndWar. Demo at the AIIDE 2012 Workshop on Musical Metacreation.

Karmaflow. Karmaflow: The rock opera videogame.

Manaris, B.; Hughes, D.; and Vassilandonakis, Y. 2011. Monterey mirror: Combining markov models, genetic algorithms, and power laws. In *Proceedings of 1st Workshop in Evolutionary Music, 2011 IEEE Congress on Evolutionary Computation (CEC 2011)*, 33–40.

Manurung, H. 2004. An evolutionary algorithm approach to poetry generation.

Maxis. 2008. Spore.

Nintendo. 1998. The legend of zelda: Ocarina of time.

Norris, J. R. 1998. *Markov chains*. Number 2008. Cambridge university press.

Oliveira, H. G.; Hervás, R.; Díaz, A.; and Gervás, P. 2014. Adapting a generic platform for poetry generation to produce spanish poems. In *5th International Conference on Computational Creativity, ICCC*.

Oliveira, H. G. 2012. Poetryme: a versatile platform for poetry generation. *Computational Creativity, Concept Invention, and General Intelligence* 1:21.

Porter, M., and Boulton, R. 2001. Snowball stemmer.

Shaker, N.; Togelius, J.; and Nelson, M. J. 2014. *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*. Springer. (To appear).

Smith, B. D., and Garnett, G. E. 2012. Improvising musical structure with hierarchical neural nets. In *Proceedings of the AIIDE 2012 Workshop on Musical Metacreation*, 63–67.

Sridhar, R.; GANGA, K.; PRABHA, G. D.; et al. 2014. Automatic tamil lyric generation based on ontological interpretation for semantics. *Sadhana* 39(1):97–121.

Studio, D. F. 2009. Brutal legend.

Suriyah, M.; Karky, M.; Geetha, T.; and Parthasarathi, R. 2011. Special indices for laalalaa lyric analysis & generation framework. In *Proc. Internat. Tamil Internet Conf*, 287–292.

Togelius, J.; Kastbjerg, E.; Schedl, D.; and Yannakakis, G. N. 2011. What is procedural content generation?: Mario on the borderline. In *Proceedings of the 2nd International Workshop on Procedural Content Generation in Games*, 3. ACM.

Toivanen, J.; Toivonen, H.; Valitutti, A.; Gross, O.; et al. 2012. Corpus-based generation of content and form in poetry. In *Proceedings of the Third International Conference on Computational Creativity*.

Toivanen, J. M.; Toivonen, H.; Valitutti, A.; et al. 2013. Automatical composition of lyrical songs. In *The Fourth International Conference on Computational Creativity*.

Wooller, R.; Brown, A. R.; Miranda, E.; Diederich, J.; and Berry, R. 2005. A framework for comparison of process in algorithmic music systems. In *Generative Arts Practice 2005 — A Creativity & Cognition Symposium*.