Expansion on Description-Based Design of Melodies

Avneesh Sarwate
Department of Computer Science,
Princeton University, Princeton, New Jersey
asarwate@princeton.edu

Rebecca Fiebrink
Department of Computer Science (also Music),
Princeton University, Princeton, New Jersey
fiebrink@cs.princeton.edu

Abstract

This work-in-progress paper describes attempted improvements on Pachet's Description-Based Design (DBD), a system that uses machine learning to generate melodies. We discuss in depth both Description-Based Design and our extensions to Pachet's original approach. We also present a user study in which users had some success in transforming melodies and describe the implications of these results for future work.

Introduction

There has been significant research into creating computer applications with "musical intelligence" by combining machine learning, Markov models, and other mathematical tools with music theory. However, very few tools exist that are "musically intelligent" and highly customizable by musicians.

In Pachet's 2009 paper "Description-Based Design of Melodies," he outlines a process that allows musicians to stochastically generate highly customized melodies without requiring any programming or knowledge of computer science. Pachet's implementation supports generation and modification of melodies according to a very specific set of criteria. We have implemented several modifications to Pachet's original approach, with the goal of supporting musically-intelligent melody generation and modification based on a wider array of criteria. We begin this paper by discussing some properties of computer-aided composition systems and providing a more detailed description of Description-Based Design (DBD). We then discuss our motivation for extending Pachet's work, describe our implementation, and present a user study for evaluating the effectiveness of our changes. Then, we discuss the results of the user study and their implications for future work.

Background

Existing music composition software may be considered with regard to two characteristics: "built-in musical intelligence" and "customizability." We say systems with in-built intelligence are those designed using knowledge of musical rules and patterns, and that have the ability to apply these patterns in the modification or generation of music. We say systems are customizable if they have functionality that can be flexibly re-configured by the user.

The Continuator, an interactive improvisation system that uses machine learning and Markov models, is an example of a music generation system with built-in musical intelligence (Pachet 2003). Cope's "Experiments in Musical Intelligence" is another such system; it uses augmented transition networks and pattern detection techniques to "learn" the styles of various classical composers and recreate music in their respective styles (Cope 1996).

By encoding specific rules or constraints for music generation, or by learning valid musical patterns from data, such types of musically intelligent systems can serve composers by generating new material (e.g., melodies) that is musically appropriate to a given context. Such content could be incorporated into a new composition or live performance as-is, or it could serve to inspire a composer by introducing examples of yet-unconsidered posssibilities.

A drawback of many prior approaches to musically-intelligent content generation is that they encode musical norms or constraints in a way that cannot easily be modified by a composer to suit his or her musicial style or purpose. A goal of our work is therefore to build a musically-intelligent melody generation system that is more highly customizable. Furthermore, we aim to allow customization without the use of programming, in order to support effective use by non-programmers.

Description-Based Design

Pachet's Description-Based Design bridges the gap between customizability and musical intelligence. It allows users to define a musical descriptor (e.g., "jazzy") and

transform existing melodies to be more like or less like the descriptor (e.g., more or less jazzy).

Users of Pachet's system can teach the computer a model of the relationship between properties ("features") of the music and user-defined descriptors using the following process: the user tags melodies output by the system, which are produced by a random melody generator, with whatever descriptor he or she desires (e.g., jazzy, funky, dissonant, smooth, etc.). For each user-defined descriptor "x," the system trains a binary classifier using melodies tagged as "x" as positive examples, and melodies without the tag "x" as negative examples.

After having learned the model of a descriptor, the system can be applied to transform a melody to be "more x" or "less x." The system generates a large number of variations of the source melody through the following process: Starting with a set that initially contains only the source melody, the system repeatedly removes a random melody from the set, creates a variation of it using a variation function, and then adds both the melody and its variation back to the set. Each generated variation is tested against the user's criterion (e.g., to test whether it is "more x"); of the variations that satisfy the criterion, the variation most similar to the original melody is returned to the user.

Pachet's Description-Based Design is a tool that merges customizability with musical intelligence. The choices made in the design of the random melody generator and the choice of musical features comprise the hardcoded intelligence of the system, and the support for user-defined tags provides a high degree of customizability.

Motivation

The original DBD system uses a hard-coded set of musical features: the number of notes in a melody, the mean value of a pitch sequence, the mean value of a pitch interval sequence, the mean value of MIDI velocity, tonal weight (an indication of how tonal a melody is), pitch compressibility ratio (an indication of repetitiveness), and interval compressibility ratio. The system is only able to learn descriptors that can be modeled by these features (by a support vector machine classifier). Indeed, Pachet (2009) evaluates the system's ability to learn five simple descriptors with relatively clear relationships to these features (e.g., "tonal," "serial," "short"). Further, the original DBD system provided a unique random melody generator for each tag, making it easy to create melodies of a given style for users to tag.

The motivation of our work is to support a larger set of descriptors through use of a more general feature set and a single, more general random melody generator. Also, while Pachet's system included only a minimal user interface, we have aimed to allow more intuitive and flexible end-user control.

Implementation

Machine Learning Features

In order to accurately learn a more diverse set of descriptors from small training sets of hand-labeled melodies, our new implementation of DBD uses six new melody analysis features. Three of these features (nos. 1–3 below) pertain to high-level melodic properties described by Tymoczko (2011), who argues that there are five fundamental properties shared by all Western tonal music. For the properties used in the feature set, Tymoczko gives the following descriptions:

- *Conjunct melodic motion*: "Melodies tend to move by short distances from note to note…"
- *Limited macroharmony*: "I [Tymoczko] use the term 'macroharmony' to refer to the total collection of notes heard over moderate spans of musical time. Tonal music tends to use relatively small macroharmonies, often involving five to eight notes."
- *Centricity*: "Over moderate spans of musical time, one note is heard as being more prominent than the others, appearing more frequently and serving as a goal of musical motion." (Tymoczko 4)

The typically small size of the training sets in DBD necessitates a small feature set for reliable classification, so the fact that Tymoczko's properties are a small set with wide stylistic applicability and a single music-theoretic basis was attractive.

The other three of our features (nos. 4–6 below) are defined as rhythmic analogs to these melodic features. Specifically, our system calculates the following features:

1. **Melodic Conjunctness**: the average of the absolute sizes of all intervals between notes.
2. **Melodic Centricity**: The fraction of notes that are the first, third, or fifth scale degrees in the most likely key.
3. **Limited Macroharmony**: The percentage of notes in the most likely key divided by the variance of the MIDI note values. This is a measure of how "focused" the melody is. It takes into account the adherence to a diatonic scale and the variation in the set of pitches used by the melody.
4. **Rhythmic Conjunctness**: The geometric mean of the larger of the two ratios between consecutive durations. This is a measure of how much the rhythm changes.
5. **Rhythmic Centricity**: The fraction of metric accents at which note onsets fall exactly (beats 1 and 3 in 4/4 time; beat 1 in 3/4).
6. **Rhythmic Self-Similarity**: The average "difference" between measures, as measured by the Levenshtein distance (Levenshtein 1965) on the duration lists of the measures.

The music21 toolkit (Ariza 2000) was used to implement computation of these features from the melodies, which were represented in the system as pitch and duration lists.

Random Generators

Pachet (2009) implemented a unique random generator for each descriptor used in his evaluation. However, in a more general-purpose system, any descriptor could potentially be used. Therefore, we aimed to implement a single random generator capable of producing variations along all features. We implemented and evaluated two alternative generators, described below.

Generator 1:

This random melody generator relies solely on two variation functions. The melodic variation function applies a small variation to a melody either by randomly selecting a note and changing its pitch, or by adding or removing a note from the melody. The rhythmic variation function converts the rhythm into an "onset vector" (Post and Toussaint 2011) and shifts the onset times of notes in the melody. To generate a new melody, we start with a "flat" phrase (one where all notes have the same pitch and duration) and apply the variation functions repeatedly (specifically, the number of applications is equal to five times the number of notes in the melody). The probability that a variation will produce a note in the "most likely key" of the melody is 80%.

Generator 2:

First, a time signature (3/4 or 4/4) is randomly selected. Then, a one-measure-long random onset vector is created. This single measure is concatenated four times to create the rhythmic "frame" for the whole phrase. The rhythmic variation function is applied a random number of times (from zero to six). A "flat" melody (where all notes are the same pitch) is then added on top of the rhythmic frame, and the melodic variation function is called a random number of times (from two to five times the number of notes in the phrase). These variations aggregate into a distinct melody. The variation is applied so that, on average, 90% of the time the new note is in the key of the phrase. Finally, the melody is "flattened" by removing solitary jumps. Solitary jumps are single notes that are either much higher or lower than both their immediate neighbors. These are brought within a perfect fifth of the value of their closest neighbor.

Finding the most likely key:

A deterministic algorithm is used to find the most likely key for a phrase. We rank the degrees of the major scale in descending order of their importance as follows: $\hat{1}, \hat{5}, \hat{3}, \hat{4}, \hat{7}, \hat{6}, \hat{2}$. We assign each degree a utility value according to its rank in this list; the most important degree ($\hat{1}$) has utility 7, the second most ($\hat{5}$) has utility 6, and so on. A utility value for a phrase can thus be computed for each key, by computing the sum of the utilities of the degree of each note in the phrase. (Notes not in the given key are given a utility of zero.) The most likely key is identified as the key with the highest overall utility.

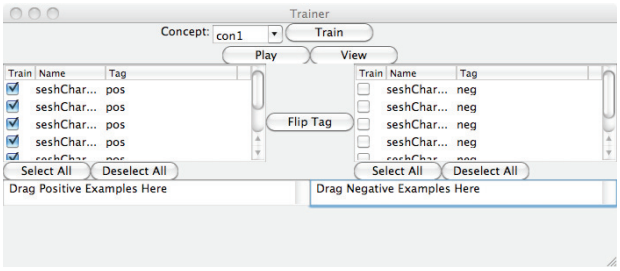

Figure 1: Training window

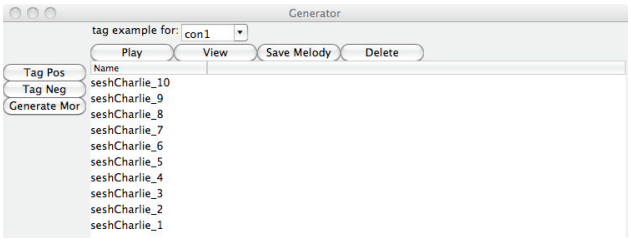

Figure 2. Tagging window

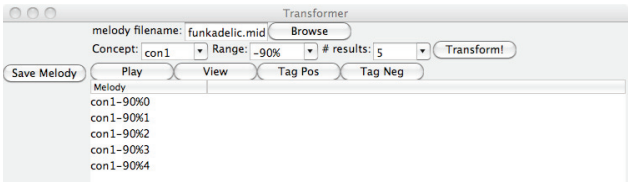

Figure 3. Transforming window

User Interface

Users interact with the system via a graphical user interface (Figures 1–3). In this interface, the user initiates random generation of a set of melodies, selects melodies to tag as positive or negative examples for any user-defined descriptor, and initiates training of a descriptor model from its current set of positive and negative examples. After taking these steps, the system can use the variation functions and descriptor models to create variations on melodies that are "more" or "less" like any descriptor.

Users can also drag and drop MIDI files into the system to serve as positive and negative training examples for a descriptor. This allows users to compose custom training melodies in an environment such as Ableton Live. We thought that users might be able to manually craft training melodies that were "stronger" positive and negative examples than those generated by the random generator, thus allowing descriptors to be learned more accurately and/or with fewer examples.

Descriptor Learning

Descriptors were learned from examples using decision trees boosted by AdaBoost (Freund and Schapire 1999), as implemented in the Weka learning library (Witten and Frank 2005).

Evaluation

We conducted a preliminary evaluation of the system to assess the ease of use and quality of music generated.

Procedure

Five subjects were recruited through mailing lists at our university. They varied in their level of programming experience (one had no experience, three had some, and one had substantial experience) and musical proficiency (one had no musical training, one had training but did not perform regularly, one actively performed but did not write music, and two both actively wrote and performed). All were undergraduate students, three female and two male. All respondents to the call for testers were taken.

Users were individually led through the following steps:

1. The goals of the system were explained verbally to the user, then all major functionality was demonstrated once.
2. The user was asked to spend 20 minutes teaching the system the concept of "funky," by generating random melodies and selecting positive and negative examples of funkiness.
3. The user was asked to spend another 20 minutes training another classifier or playing "freely" with the system.
4. The user completed a written questionnaire and was then interviewed about his experience.

The questionnaire asked users about the simplicity of the interface, their satisfaction with the music produced by the system, and the overall usability/usefulness of the system. The design of the questionnaire was influenced by the work of Csikszentmihalyi (1990), which presented the idea of "flow," of moving from goal to goal and task to task without hesitation. The questionnaire asked about users' ability to navigate the interface without interruption. The questionnaire responses were used as a launching point for the subsequent interview.

Results

Pilot Test

First, a pilot test was run with one user, a person with no experience in either programming or music composition. The test revealed that the melodies created by the random generator (Generator 1) were too "random" sounding and too similar to each other, leaving the user unable to judge whether a given melody was at all funky. This user (and, later on, another informal tester) remarked that there did not seem to be any "melodicness" in the note patterns, that the rhythms were too unpredictable, and that there were surprisingly large intervallic jumps. These results prompted the design of Generator 2.

Subsequent Evaluation

Generator 2 was used in a subsequent evaluation with four people. In this evaluation, hardware malfunctions prevented users from successfully using the piano keyboard to write custom melodies during the tests. Thus, users were limited to selecting system-generated melodies to serve as "source melodies" to transform. All users followed the testing procedure as described above.

All users had difficulty classifying melodies as "funky," saying that the melodies still sounded too "random" to accurately judge. However, all users were able to distinguish between melodies that they liked and disliked, which was not the case with users who tested Generator 1. Overall, users found the interface easy to use after a brief tutorial, but two commented that they would have had difficulty knowing what to do without some initial prompting. The two users with songwriting experience spent much more time listening to the randomly generated melodies, often listening to the same melodies multiple times. One user trained a classifier on melodies he "liked" and "disliked" and was able to create variations that he claimed sometimes improved on the original melody.

Discussion

User Interactions

Users' level of concentration when playing with the system was strongly correlated with experience in writing music rather than music theory knowledge or performance experience. Users seemed bored by tagging training examples, and all used the full allotted time for training. Improvements to the tagging process are discussed later.

Random Melody Generator

Creating a "general-purpose" random melody generator was one of the design challenges we faced. An ideal generator would produce melodies of all types, so that for any descriptor, positive training examples could potentially be generated. Generator 1 enforced only the condition that, on average, 80% of the notes in the melody should fit into the most likely key. Unfortunately, most of the resulting melodies had few perceivable patterns in their rhythmic or melodic structure, sounding for the most part like random keys being hit on a piano instead of a coherent melody.

The users' criticisms of the random melodies produced by Generator 1 (that they were too rhythmically varied, dissonant, and had large jumps) indicated that the Generator 1 was not adhering strongly enough to Tymoczko's properties. In order to improve the quality of the melodies without introducing too many arbitrary limitations, our second melody generator only enforced those of Tymoczko's properties which Generator 1 had "violated." Most melodies in Western music have subsegments with similar rhythmic patterns. Therefore,

when designing Generator 2, generating rhythms based on variations of single-measure rhythms seemed justified. Since users found Generator 1's melodies too "atonal," the in-key percentage of Generator 2 was increased, with 90% being chosen through listening and tweaking. "Flattening," as described for Generator 2, was a heuristic that was implemented to more strongly enforce "Conjunct Melodic Motion," one of Tymoczko's properties of tonal music stating that melodies generally move in steps of small intervals. The interval of a fifth was arbitrarily chosen because it is fairly common and relatively large interval.

Machine Learning and User Interaction

During the initial design phase, we realized that the interface design would have a substantial effect on the quality of the machine-learned classifier. If the interface was clunky or confusing, it would take users longer to tag or write training melodies. As a result, there would be fewer tagged melodies, or those tagged could be of a lower quality. Because of the low number of examples in the training set, the difference in the training set caused by a bad interface could be significant. Based on the results of the evaluation, it seems reasonable to conclude that the quality of randomly generated melodies far outweighs the interface as the bottleneck for satisfactory training.

Future Work

Interface Improvements

By far the slowest task performed by users was the tagging process, moreso for users with less musical experience. One solution to speed up this process is to make the tagging process optional. Users could be given a handful of premade classifiers to use "out of the box." These premade classifiers could be "hand-built," or they could be generated from crowd-sourced tags. However, if users wanted to customize the classifiers, they could add their own training examples, weight their new training examples relative to the original examples used to train the classifier, and retrain the classifier using both old and new examples.

Tagging could also be replaced with pair-wise comparisons between two melodies, where a user is given two melodies and asked to judge which one is "more x." These pair-wise comparisons could be used to train a ranking SVM (Joachims 2002). From an interaction perspective, pair-wise comparisons may be a more comfortable task for users than assigning positive or negative tags to melodies, as users would only have to consider two melodies instead of trying to remember an "average" against which to compare.

Data collection

Pachet advocates for the collection of crowd-sourced tagging data in building classifiers. Such data could be very helpful both in creating premade classifiers and in feature selection.

User tags could either be collected directly from users' systems, or by farming the tagging to Amazon Mechanical Turk (AMT). AMT could be used in a manner similar to the work of Chaudhary and Koltun (2010), who collected tags relevant to 3D modeling. Here, AMT users might be given two melodies, and asked to rank one of them as more "x" than the other, with "x" being any adjective descriptor of their choosing. Sets of melodies tagged with synonyms could be combined. The pairs for each descriptor set could then be used to train a ranking SVM.

Building up a large set of tagged or ranked examples could also be useful in feature selection. With a set of baseline data, we could evaluate the effectiveness of a wider set of music analysis features with respect to their relevance in classifying melodies according to descriptors.

Random Melody Generation

Rather than relying on a random melody generator to produce melodies for the user to tag, we could start with a large set of pre-made, "satisfactory" melodies—ones that have been determined (by some criteria) to be "melodic" enough for a general user to work with. Once a user starts tagging these melodies to train a concept classifier, the system could take an active learning approach, prompting the user to tag the pre-made melodies that are likely to be most useful in improving the classifier.

Currently, we are researching new random generation approaches that allow users different modes of control over the melodic or rhythmic structure. For example, given a user-provided rhythm and a set of notes, melodies could be produced using only those notes and similar rhythms. This approach could support production of more appropriately structured melodies without sacrificing generality. Current challenges include determining the choice of structural elements and the design of an effective user interface.

Conclusion

We have described a new system that builds on Pachet's Description-Based Design. Our system attempts to combine musical intelligence with a higher degree of end-user customizability. An evaluation of our system has identified a few critical avenues for improvement. First, the end-user tagging process can be arduous and unpleasant. Second, the creation of a good general-purpose random melody generator—capable of generating melodies that are both diverse enough and "melodic" enough to serve as appropriate positive or negative examples of arbitrary descriptors—remains challenging. Our current and future work aims to address these challenges through the use of crowd-sourced data collection and new mechanisms for user guidance of melody generation algorithms.

References

Ariza, C., and Cuthbert, M. 2000. Music21: A Toolkit for Computer-Aided Musicology and Symbolic Music Data. *Proc. ISMIR.*

Csikszentmihalyi, M. 1990. *Flow: The Psychology of Optimal Experience*. New York: Harper and Row.

Chaudhuri, S. and Koltun, V. 2010. Data-driven Suggestions for Creativity Support in 3D Modeling. *ACM Trans. Graph*. 29(6), Article 183 (December 2010).

Cope, D. 1996. *Experiments in Musical Intelligence*. Madison: A-R Editions.

Freund, Y., and Schapire, R. 1999. A Short Introduction to Boosting. *Journal of Japanese Society for Artificial Intelligence* 14(5): 771–780.

Joachims, T. 2002. Optimizing Search Engines Using Clickthrough Data. *Proc. ACM Conference on Knowledge Discovery and Data Mining*.

Levenshtein, V. I. 1965. Binary Codes Capable of Correcting Deletions, Insertions, and Reversals. *Cybernetics and Control Theory* 10(8): 707–710.

Pachet. F. 2003. The Continuator: Musical Interaction with Style. *Journal of New Music Research* 32(3): 333–341.

Pachet, Francois. 2009. Description-Based Design of Melodies. *Computer Music Journal* 33(4): 56–68.

Post, O., and Toussaint, G. 2011. The Edit Distance as a Measure of Perceived Rhythmic Similarity. *Empirical Musicology Review* 6(3): 164–179.

Tymoczko, D. 2011. *A Geometry of Music: Harmony and Counterpoint in the Extended Common Practice*. Oxford: Oxford University Press.

Witten, I., and Frank, E. 2005. *Data Mining: Practical Machine Learning Tools and Techniques*. San Francisco: Morgan Kaufmann Publishers.