Variator: A Creativity Support Tool for Music Composition

Avneesh Sarwate
Princeton University
P.O 3624 Frist Ctr
Princeton, NJ 08544
asarwate@princeton.edu

ABSTRACT

The Variator is a compositional assistance tool that allows users to quickly produce and experiment with variations on musical objects, such as chords, melodies and chord progressions. The transformations performed by the Variator can range from standard counterpoint transformations (inversion, retrograde, transposition) to more complicated custom transformations, and the system is built to encourage the writing of custom transformations.

This paper explores the design decisions involved in creating a Creativity Support Tool (CST), describes the Variator interface and a preliminary set of implemented transformation functions, analyzes the results of the evaluations of a prototype system, and lays out future plans for expanding upon that system, both as a stand-alone application and as the basis for an open source/collaborative community where users can implement and share their own transformation functions.

Keywords

Creativity Support Tools, Assisted Composition, Social Composition

1. INTRODUCTION

As computer based tools begin to permeate the landscape of many industries, one particular challenge has become to design tools that not only bolster efficiency, but creativity. The motivation for Variator was to create a system that could help assist human creativity by attempting to tackle the problem of musical writers block. The system would necessarily have to allow users to quickly insert, tinker with, and play the musical objects that they are working with in a relatively seamless manner. Ben Shneiderman provides an account of the many factors that go into creating "Creativity Support Tools" in his paper in Communications in the ACM [3]. Among them, he mentions the support for exploratory search and history keeping as key design principles in facilitating the creation of Creativity Support Tools. Variator attempts to provide an interface that gives users the ability to quickly and painlessly search through "musical space" and keep track of their progress, so they do not lose good ideas once they find them. But also, the aim of the Variator is to incorporate musical intelligence in order to generate new music not explicitly written by the user. The use of musical intelligence within the framework of a creativity support tool is what differentiates Variator from existing systems.

2. RELATED WORK

There exist popular musical notation/arrangement tools, such as Sibelius and Finale, and most of the popular Digital Audio Workstations, such as Pro Tools, Reason, and FL Studio, include piano roll type scores for each instrument track. These programs offer users a fairly quick and easy way to notate, modify, and play back music. However, none of these systems exhibit much "musical intelligence." The notations softwares show minimal musical knowledge, as they can exploit the basic relationships and rules that govern the diatonic key structure to expedite the process of entering notes into a score. For example, when entering music into a score where a specific key has been chosen, certain keys will map to overlaying certain intervals into the score. The software can also shift segments up or down along the staff, and automatically groups notes according to the chosen time signature. However, these features mostly amount to minimal arithmetic based on simple and explicitly chosen patterns. The piano roll layout for most DAW's offer even less, simply offering a way to shift selected notes up or down on the piano.

On the other end, there are systems that exhibit a high degree of musical intelligence, but do not allow for a high level of control of the type of music produced. The most famous example may be David Cope's *Experiments in Musical Intelligence,* which, after learning from a corpus of works, can produce entire pieces [1]. Cope's stated initial motivation for the project is actually very similar to the motivation behind Variator, as he says he intended the system to be a cure for composers' block. However, the end product was a system that, while exhibiting a very high degree of musical knowledge, did not provide "low level" control over the music produced, i.e, the ability to control the production of music conforming to similarities only at the level of individual melodies or chord progressions.

A similar system is Pachet's Continuator (from which Variator borrows its titular theme). The Continuator lies between the domains of probabilistic music generation systems and interactive music systems. In short, it learns a corpus of music, and then generates music based on a Markov model learnt from this corpus. However, the real time input of a playing musician also influences the what music generated, as the continuator defines a fitness function of how well the generated music matches the real musician's output. The music finally output by the generator is that which follows the Markov model and best fits the musicians output as defined by the fitness function [4].

While Continuator has much more "low level" control than Cope's system, it still produces music probabilistically. Therefore, a user could only produce "similar" music, but not explicitly explore along a certain direction or idea.

3. VARIATOR

3.1 System Details

Variator is implemented in Python and ChucK. The GUI is built using TKInter, Python's interface to the TK GUI Tool. All

of the transformation functions are implemented as Python modules. ChucK is used as the sound producing back end. All communication between Python and ChucK is done using Open Sound Control.

3.2 Interface Elements

Variator can currently work with three different types of musical "objects." Phrases are single voice melodies, represented as corresponding lists of notes and times. Phrases can also optionally have a string denoting their key. Chords are represented as a list of notes.

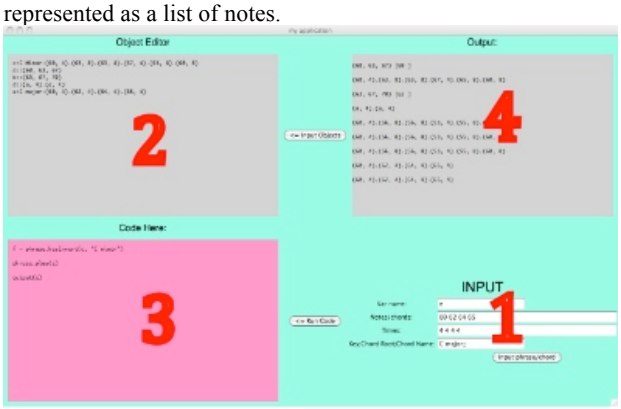

Figure 1. Variator interface: (1) The input area, (2) The Object editing window, (3) The coding area, (4) The output window

Chords also specify which of the notes is the "root" of the chord, which is by default set to the lowest note in the chord. Chords also have an optional string that gives the "name" of the chord. Progressions are chord progressions, and they are represented by parallel lists of chords and associated times. Currently, progressions can only be created from previously created chords.

3.2.1 Input

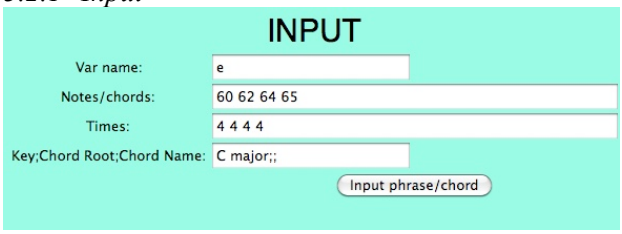

Figure 2. Variator object input method

All objects input to Variator must be given variable names. The type of object being input need not be declared. Inputting only notes will be recognized as a chord, notes and times as a phrase, and chords and times as a progression. Notes are currently input as integer values corresponding to the numbers that MIDI associates with notes. Optional attributes, such as associating a key with a phrase, or a non-default root or a name for the chord, can be specified in the last entry line in the input method.

3.2.2 Object Editor

Figure 3. Variator Object Editor

The object editor allows users to both see all of the objects they have currently input into Variator and edit them. The editor window itself is simply a text display, and users can go in and change objects at will (as long as they do not change the formatting of the text), saving their changes by pressing the "Input Objects" button at the right. The text formatting is "varName:KeyString:objectRepresentation." If users wish to save the set of objects they are working with, they can simply save the text from the object editor using any text editing program, and can similarly load a set of saved objects by copying them into the object editor and pressing the "Input Objects" button.

3.2.3 Coding Area

```
e = phrase.keyInvert(c, "C minor")

play(e)

output(e)
```

Figure 3. A simple code example. The "phrase" module contains a group of built in transformations. This example takes the phrase called "c", transforms it, plays it, and prints the string representation of "e" to the output

All of the objects in Variator are Python objects, and all of the transformations are Python functions that take Python/Variator objects as arguments and return other Python/Variator objects. All of this is executed in standard Python in the coding area. Once code has been typed out, it can be immediately executed by pressing the "Run Code" button to the right of the coding area.

3.2.4 Output window

The output window is a simple output terminal. The output() command can be passed anything that would normally be passed to the "print" command in Python, and instead of in the normal terminal window, the arguments are printed in the Variator output window. When Variator objects are printed, they are printed in the format such that they can be copied directly to the object editor window if the user wishes to save them for later use.

3.3 Transformations

The core novelty of Variator lies in its transformative approach to music generation. A transformation is implemented as a python function that takes in a Variator/Python object as one of its arguments, and then produces a different Variator/Python object. Variator comes with several built in transformation, and one of the goals for Variator is to allow users to code their own transformations to use.

3.3.1 Built-In Transformations

- Interpolations between two objects: This function takes two phrases and a floating point value (0, 1) as arguments. The floating point value determines whether the output will be more like the first phrase argument (values closer to 0), or the second (values closer to 1). It is implemented using the Levenshtein distance algorithm [2].
- Melodicizing a chord progression: function takes a chord progression and a melody as arguments and inverts each chord in the progression to produce a new progression where the top notes of each chord follow the melody as closely as possible.
- Giving progressions a direction: This function takes a chord progression and revoices the chords so that all the top notes either descend or ascend by the smallest possible intervals.

- Reevaluating rhythm: This function takes a phrase object and an importance ranking over the scale of the phrase. Then, based on the importance ranking, it redistributes the time values of the phrase, assigning the longer time values to the more "important" notes.
- Key preserving transposition: This function transposes the notes in a phrase by scale degree rather than by absolute intervals, thus preserving the key of the original phrase.
- Key preserving inversion: Rather than reversing the intervals between each pair of notes, this function reverses the change in scale degree between each pair of notes, thus preserving the key of the original phrase.
- Fitting a melody to a chord progression: This function takes a phrase, a chord progression, and a mix factor [0, 1]. For the fraction of notes specified by the mix factor, the function changes them to chord tones from the chord in the progression the note is being played over.

3.3.2 Helper Objects

Variator's library comes with several "helper" objects and functions for those wishing to code their own transformations. It contains:

- A method to change the root key of a melody
- A method to change the mode of a melody (but preserve the root note)
- A method for detecting the key of a given melodic sequence
- Intervallic definitions of the modes of the major scale,
- An "importance ranking" for notes of the major scale
- A method, given a melodic sequence and a chord progression, returns the segments of the sequence over each chord in the progression.
- A method to determine the best triad over a given melodic sequence.

The helper objects are subjectively defined in terms of musical choices they make in their implementation, and are included as a template for users' own definitions.

4. EVALUATION

The evaluation process for Variator involved both an evaluation of the overall system, and questions individually concerning the interface and the built in transformations. The overall goals of the evaluation were to determine users' impressions of the effectiveness of Variator as a tool, and to learn the context in which users approached using the Variator (eg, whether assumed Variator would be a legitimate tool, an unproven system, an unhelpful "toy" system, etc).

4.1 Overall Evaluation

Participants were asked several questions regarding their background, rating from 1 to 10:

- Their knowledge of music theory
- Their experience with writing music
- Their programming experience

Real world examples were given for the value of "10," to attempt to give a more well-defined range. For example, for programming experience, 1 is defined as no experience, and 10 is defined as having equivalent experience to having received and undergraduate degree in Computer Science. Participants were also asked to qualitatively describe their songwriting process, and more specifically, what "parts" of the song they generally tended to write before others (rhythms before melodies before chords, etc).

The participants were given a 20-minute "tutorial" where the system and some of its functions were demonstrated. They were then given the code to run on their own machines. Participants then had one hour to experiment with the system themselves, both to write music and to tinker with the code and write original functions. Each participant was then given 5 minutes to recount his or her initial impressions of the system. The following questions were then asked:

- Overall, do you think you would experiment with his system when writing music? Why or Why not.
- If you answered yes to the previous question, would you be more likely to use this tool during composition of songs, or more likely to simply "experiment" with the system? Why?
- Would this system at all change your songwriting process?
- In a general context, how would computer aided composition be most useful to you?

4.2 Interface Evaluation

The questions asked about the interface were influenced by the metrics proposed in the Creativity Support index [5]. Specific questions regarding the interface were:

- Overall, how intuitive did you find the interface? 1 – 10. 1 being no instructions needed, 10 being needed lots of instruction.
- Overall, how flexible did you find this interface? 1-10
  Qualitative extension: was there anything you felt that could not be easily accomplished with this interface?
- How rapidly were you able to execute sequences of commands? (How quickly could you "flow" through different tasks while using Variator?) 1- 10

4.3 Built-in Transformation Evaluation

Specific questions regarding the transformations were:

- How rich did you find the representation of the musical objects?_1-10
  Qualitative extension: Would you add any parameters to them?
- How useful did you find the given set of built in transformations? 1- 10
- How helpful did you find the "helper" objects. 1 - 10
- What helper objects/functions would you propose be included?
- How likely would you be to write your own functions if you were to use this system outside of an evaluation setting?

4.4 Group Discussion

After the individual responses were collected, all of the participants were collected for a group discussion. Participants were asked to take notes on points that they found interesting from the discussion, and notes were taking by the evaluation conductor as well. All notes were then collected for examination.

5. RESULTS

Four subjects participated in the evaluation. Due to the small sample size, the relatively high variance in the numerical data, and the similarity of qualitative responses about questions involving numerical ratings, it was concluded that the numerical data was not useful in this particular set of evaluations. Of the four participants, all had a reasonable amount of programming experience. One had little to no music theory or songwriting experience, two had some music theory experience/songwriting experience, and one had a very strong background in both music theory and composition.

All participants agreed that the "Object Editor" window was an area where many elements could be improved. Firstly, it was recommended that the Object Editor should provide a graphical representation of the objects being stored. Participants found the process of trying to constantly interpret numbers when

looking at objects to be time consuming and difficult. Participants also found it very disorienting and counterintuitive that Object Editor did not dynamically update in response to objects produced in the code execution window. They found the distinction between objects displayed in the Object Editor and objects created programmatically (which are not displayed in the Object Editor unless specifically put there by the user) to be confusing when quickly trying to iterate through variations.

All participants also found the Input area to be quite confusing. The area was designed to present a uniform interface for entering all objects and automatically detect what object had been entered. Instead of simplifying the process of object input, the lack of explicit instructions/entries left users confused as to how to proceed. Also, users again expressed frustration with having to work with midi integer values.

Overall, users found the level of detail of the representation of objects usable. They also found the concept of a system allowing for fast variation quite useful, though they found the particular functions implemented for this iteration of Variator to not be flexible enough for general use. 2 users expressed a strong preference for randomized transformative functions, where the degree of randomness could be controlled.

All of the users also expressed the desire for a feature that would allow them to define transformation functions from within the interface itself to allow for more rapid prototyping. They also all supported the idea of in-interface documentation about functions, suggesting an option similar to the way Eclipse provides pop-up menus when accessing members of a class.

6. FUTURE WORK

Intended future work for Variator is split into several different categories: (1) Improvements on the interface. (2) Extensions on the live-playing functionality of the application. (3) Improvement of the helper objects/functions to facilitate faster developments of transformations functions. (4) Creation of a community to produce and share transformation functions. A good deal of the direction of future work is influenced by the resuls of the evaluation.

6.1 Interface Improvements

Currently, the only way to view and edit objects is through though the text of the object edit window. For many users, the lack of a true visual representation of the musical objects could make it quite difficult to effectively experiment with transformations. The loss of spatial information given by sheet music or a piano roll creates a lag in processing the information on the screen. However, graphical representations would also take up more screen real-estate, preventing the user from seeing as many objects at once. Future versions of Variator will seek to remedy this problem by allowing the user to switch between different views of the saved objects, including test, sheet music, and piano roll views. Efforts will also be made to streamline the process of object entry, which could be a very preventative lag in the use of the system.

Ways to allow users to define functions from the interface itself would also be explored, as this would both give users the ability to prototype faster during function development, and make the interface more expressive by allowing for immediate customization of the tools available to the user.

More long term plans for Variator include integration with notation software or DAWs. Variator could be much more closely tied to the compositional process if it were integrated with the tools musicians use most often to compose. In particular, allowing copy/pasting of musical objects between other software and Variator could virtually eliminate a huge time overhead in its use.

6.2 Live-Playing Functionality

The triggering of transformations by external controllers could provide interesting applications for the creation of real-time interactive music systems. In particular, the mapping of particular transformations to signals form a controller (such as a transformation for every button on a d-pad) would allow users to harness the musical intelligence of Variator in a live context. The use of triggered transformations would greatly speed up the process of exploratory search, and could provide beneficial in non-live use as well.

6.3 Improving Transformation-creating Tools

In order to facilitate the writing of functions by users, it would be worthwhile to include more “low level” objects and functions based on more rigorous music theory research. Functions that measured various definitions of distance between different types of objects, as well as different measures of “affinity” between objects would serve as useful building blocks. Also, intervallic definitions for keys not generally found in western music, as well as ways to “convert” melodies to keys with different numbers of notes (ex hexatonic to pentatonic) could provide users with a means of exploring new soundscapes with minimal time spent learning the requisite theory.

6.4 Creating a Community for Sharing of Transformations

Creating a forum where users could share their transformation functions could greatly expand the use of the system by allowing users access to tools that they could not produce themselves. A similar community exists based around the products of guitar software manufacturer Line 6. Line 6 specializes in the digital modeling of guitar amplifiers and effects, and has created a large online community where users can share their presets with others. A community for Variator “patches,” however, could prove even more useful. Allowing users to share patches could allow those users with minimal music theory knowledge to utilize transformations created by more proficient theorists. Also, allowing users to share patches could greatly increase the rate at which users generate their own transformations, as they could more easily learn the different ways to attack problems from reading others’ code, and could modify existing patches to reach their specific goals instead of starting from scratch.

Creating a means by which to share functions could also create a new model for collaborative composition. Teams of users could work together on more ambitious projects, such as fitness functions for hard-to-define adjectives like “funkiness” or “jazziness”.

7. DISCUSSION

Overall, the user studies suggested that Shneiderman’s two key features of Creativity Support Tools, rich history keeping and exploratory search support, are deficient in fundamental ways. Although history keeping exists, it must be user regulated to a degree higher than users are comfortable with, and the visual representation of the history does not quickly provide the desired information. Also, the programming approach to executing transformations renders the system all but unusable to those without coding experience. It was noted that the one user with no background in music theory seemed lost while experimenting in the system, and remarked that he didn’t know how to proceed without knowing how the transformations worked. With this in mind, it may provide useful to divide further development of functions by demographics. Functions for users with little music experience could involve a high

degree of intelligence, taking more of the decision making away from the user, but providing "normal" sounding music more often. Functions aimed towards more serious composers could incorporate more randomness, as suggested, or involve transformations that are more systematic and combinable into intricate systems.

The problem of non-coders not being able to use the system is an urgent one that requires addressing. Changes to the interface could allow non-programmers to select transformations and arguments via menus, but would still not address the problem that non-coders would have with designing their own transformations. One evaluation participant suggested that, if a community of transformation makers were to develop, non-coding users of the system could post requests of transformations to be made by those who can program.

Bugs in the interface and modules also prevented the evaluations from occurring as smoothly as possible, and an efficient method for testing combinations of user inputs to the interface will be necessary for the further development of features (especially live-playing functionality).

8. CONCLUSION

On the continuum of "interactivity" to "control" as defined by Lippe, the Variator seems to fall very far to the side of control [4]. The lack of publications on systems offering the same level of explicit low level control suggest that there may be a niche among composers for such a tool. The user studies suggest that, while insufficient in its current form, Variator introduces core design concepts and user features that that are worthwhile for composers to experiment with in their work.

9. REFERENCES

[1] Shneiderman, B. Creativity support tools: Accelerating discovery and innovation. *Communications of the ACM, 50*, 12 (Dec. 2007), 20–32.

[2] Cope, David. "Experiments in Musical Intelligence.". University of California, Santa Cruz. Web. 15 Jan 2013.

[3] Carroll, E., Latulipe, C., Fung R., and Terry, M.. Creativity factor evaluation: Towards a standardized survey metric for creativity support. *Proceedings of ACM Creativity & Cognition*, (2009), 127–136.

[4] Lippe, C., Real-Time Interaction Among Composers, Performers, and Computer Systems, *Information Processing Soceity of Japan, SIG Notes, 123*, (2002), 1–6.

[5] Pachet. Francois. "The Continuator: Musical Interaction with Style." *Journal of New Music Research, 32*, 3 (2003), 333-341.