Composing with All Sound Using the FreeSound and Wordnik APIs

Evan X. Merz
University of California at Santa Cruz
Music Department
1156 High Street
Santa Cruz, CA. 95064.
emerz@ucsc.edu

Abstract

In order to create algorithmic art using the wealth of documents available on the internet, artists must discover strategies for organizing those documents. In this paper I demonstrate techniques for organizing and collaging sounds from the user-contributed database at freesound.org. Sounds can be organized in a graph structure by exploiting aural similarity relationships provided by freesound.org, and lexical relationships provided by wordnik.com. Music can then be generated from these graphs in a variety of ways. In the final section, I elaborate on three pieces I've generated from sound graphs using cellular automata, and swarm intelligence.

Introduction

One of my ambitions as a composer is to write music using all types of sounds, including instrumental sounds, recorded sounds, and synthesized sounds. With the wealth of sounds available on the internet, it is possible to create music by combining these sounds. However, this includes such a variety of material that it is a daunting task to select and organize sounds in a way that results in compelling music.

In the project described here, I organized sounds into a graph structure. A graph is a mathematical abstraction made up of symbols and connections between those symbols. Graphs are commonly used to model social networks, links between websites on the internet, and street maps.

The sounds are organized using two types of relationships: sound similarity relationships and lexical relationships. The sound similarity ratings are provided by freesound.org, which is the source of the sounds used in these compositions. The lexical relationships are provided by wordnik.com.

Related Work

Although Luigi Russolo dreamed of an expanded sound-palette in the early twentieth century, incorporating many sounds into music composition has been largely limited by the available technology. However, the web has allowed composers to access and include an ever more diverse array of samples in their music. For some composers, using many samples has become both a musical and a political imperative. In 2008, Johannes Kreidler created Product Placements, a piece containing 70,200 copyrighted samples. His goal was to mock the outdated structure of copyright law in the internet age. The culmination of his protest occurred when he delivered thousands of pages of paperwork to the German copyright authorities because he was not allowed to register his samples online.

Other artists have created live web streams that constantly combine and remix the media posted to the internet. Ben Baker-Smith's Infinite Glitch is a piece of software that combines videos from YouTube on the fly. It exists as a permanent web installation at infiniteglitch.com, where it generates an ever-changing, chaotic collage of glitchy video remixes intended to overwhelm the viewer's senses.

The works by Baker-Smith and Kreidler are extreme examples of the type of composition that is possible with the media on the internet. However, neither artist tried to combine the disparate media in a way that considered the content of the media files, and how they related to their new context.

Since the late 1990s, mashup artists have been combining parts of popular songs to form new compositions based on the recognition of some shared quality. These artists are relevant to this project specifically because they recombine samples created by other people, based on judgments that those samples work well together, which is precisely the process that is automated by my software. The usual formula for a

mashup is to combine the vocals from one recording with the instruments from another. These type of mashups are called A+B mashups, but mashups can get much more complex.

Mashups are also pertinent to this work because they transform listenership into a participatory experience. This is enabled by various message boards and websites devoted to creating mashups such as gybo5.com. Mashups are an extension of the growing participatory culture movement that has blossomed in the internet age.

The story of participatory culture starts with the trekkies. Initially scorned by the media as outcasts (Jenkins 2006), trekkies often meet to share their fan-fiction and re-enact their favorite scenes.

Websites such as FreeSound explicitly enable participatory culture. FreeSound allows users to upload, annotate and share audio files. Creating art with databases that are accessible or modifiable by users has been dubbed database art. Although database art is primarily a movement in the visual arts, my work shares one distinct characteristic with that movement. The software described here essentially acts as an agent for the artist, recontextualizing data from the web based on a process created by the artist.

> The data readymade in database art has two important characteristics: a resistance to artist-made content and the transformation of the conventional role of the artist. Although the artist of this genre must contribute the core concept to the artwork, he or she creates a process more than a final product. It is this process that causes a flow of meaning between the old and the new contexts. (Hsu 2013, 81)

It's also important to note that other composers have written software that generates music using online audio files. Indeed, that was the initial goal of the creators of the FreeSound website. Before FreeSound, one of the creators worked on a project called Public Sound Objects, which provided a shared musical space for real-time collaboration.

> The Public Sound Objects (PSOs) project consists of... a networked musical system, which is an experimental framework to implement and test new concepts for online music communication. The PSOs project approaches the idea of collaborative musical performances over the Internet by aiming to go beyond the concept of using computer networks as a channel to connect performing spaces. This is achieved by exploring the internet's shared nature in order to provide a public musical space where anonymous users can meet and be found performing in collective sonic art pieces. (Barbosa 2005, 233)

This project was expanded after FreeSound was established in a project called FreeSound Radio.

> FreeSound Radio [is] an experimental environment that allows users to collectively explore the content in Freesound.org by listening to combinations of sounds represented using a graph data structure. Users can create new combinations from scratch or from existing ones. A continuous supply of potential combinations is provided by a genetic algorithm for the radio to play. (Romo et. al. 2009)

These earlier projects differ from this one in several ways. Significantly, my own work incorporates lexical relationships to connect otherwise unrelated sounds in the FreeSound database.

Constructing Sound Graphs

While both Google and Yahoo allow users to search for sound files, neither search engine is optimized to search for audio. When you search for sound files using a traditional search engine, your search returns audio files that occur near where your search terms appear. On properly named and documented audio files, this might lead to a file related to the search terms, but traditional search engines do not return any additional information. It is impossible to know the duration of the sound, or to find similar sounds, or to get a set of tags that might describe the sound. FreeSound provides a searchable database of user-contributed sounds that can work in place of a traditional search engine. FreeSound contains a database of over 66,000 sounds, as well as audio analysis information and user-submitted tags.

FreeSound provides access to their user-generated database of sounds through an application programming interface (API). An API is a protocol that programs can use to communicate with one another. In this case, my program makes requests to the FreeSound website, which returns the requested data. The FreeSound API provides multiple search mechanisms for retrieving sounds, including text search, content-based search, and sound similarity search. In my software, the text search is used to find sounds relating to the initial search term and sounds with related tags, while the similarity search is used to build networks around those sounds.

The FreeSound text search checks nearly all the text and numeric fields associated with a sound, including file name, file id, description and tags. According to the API documentation, "searching for '1234' will find you files with id 1234, files that have 1234 in the description etc" (FreeSound API Documentation). This leads to results where all interpretations of the search terms are included. For instance, when I searched for the term *metal*, the first

two pages of results contained sounds made by hitting metal objects, while the third page contained a sample of a heavy-metal band.

The FreeSound similarity search relies on a distance metric that combines features from several other analyses.

> the similarity measure used is a normalized Manhattan distance of audio features belonging to three different groups: a first group gathering spectral and temporal descriptors included in the MPEG-7 standard; a second one built on Bark Bands perceptual division of the acoustic spectrum, using the mean and variance of relative energies for each band; and, finally a third one, composed of Mel-Frequency Cepstral Coefficients and their corresponding variances. (Martinez et. al., 2009)

These search-based mechanisms are used to build graphs of related sounds. A graph is a mathematical abstraction consisting of vertices and edges. The vertices are symbols, while the edges represent connections between those symbols. In this project, the vertices sounds, while the edges represent an aural or lexical relationship between two sounds. The aural edges are weighted by their similarity rating. Sounds with a lower distance rating are connected by edges with a higher weight. The lexical edges are unweighted. The current version of the program does not consider edge weight in activation, but future versions of the program may do so. The graphs contain the raw materials for a piece, and the connections that might be explored in building a piece. But the graph is not a composition in itself. It is merely a static data structure. The sounds in the graph must be activated in order to generate a new piece.

The initial sound is obtained by using a text search on search terms provided by the composer. The first sound returned by that search is used as the first vertex.

Next a basic similarity search is used to attach sounds with a similarity distance less than a predefined threshold. The similarity search provided by the FreeSound API never returns fewer than fifteen sounds, but it may be the case that the sounds returned aren't very similar to the original sound. The composer can specify a distance threshold that tells the program to ignore sounds that are not very similar to the sound under consideration. Some of the sounds may have a similarity distance greater than the specified threshold. As of this writing, the FreeSound database contains over 66,000 sound files, so there are usually several sounds that are audibly similar to the starting sound. The program adds these sounds to the graph, and adds edges connecting the new sounds to the target sound.

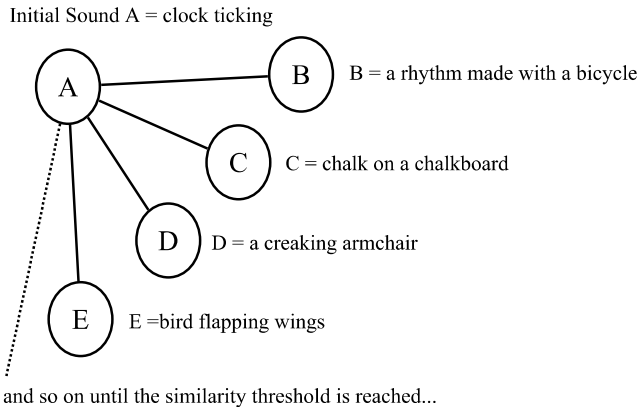

Figure 1. Basic Similarity Search

After the basic similarity search, a recursive similarity search is employed to get a list of sounds, each being the most similar sound to the previous sound. In other words, a similarity search is executed on the most similar sound to the original sound. The resulting most similar sound is connected to that sound. Then this process is repeated up to a composer-specified depth. This creates a chain of sounds of length equal to a composer-specified value. This allows a composition to move from a specified sound to other sounds that are less related by steps.

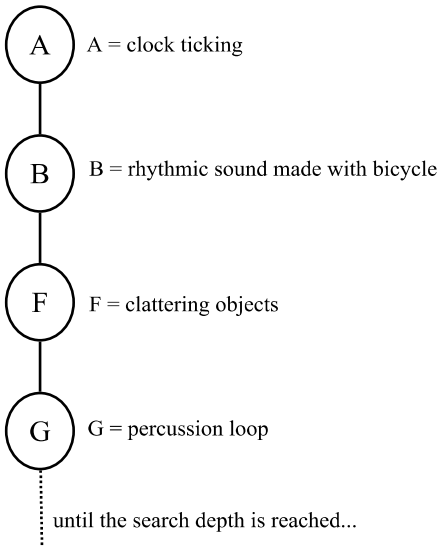

Figure 2. Recursive Similarity Search

So far all of the sounds in the graph other than the initial vertex are discovered via similarity search. However, a composer may want a piece to suddenly switch from one group of related sounds to another one entirely. Using only similarity connections, however, the program would have to go through many similar sounds before sufficiently different sounds are found. There is no way to connect one

group of sounds, to sounds that may be related but aren't audibly similar. Another search mechanism is required in order to allow a piece to have sections exploring other types of sounds. In this case, I use words related to the sounds in question to search for new sounds.

Wordnik.com provides an API that allows a program to find words that are related to a target word. The related words fall into categories that include synonyms, hypernyms, hyponyms, same-context, reverse dictionary, and tags. Provided the term *clock*, for example, Wordnik returns the synonym *timepiece,* the same-context word *lamp,* and the hyponym *atomic clocks.*

In my software, the tags on the original sound on FreeSound are used as search terms to the Wordnik API. The related words returned by Wordnik are then used as new search terms in FreeSound. This provides a mechanism through which the software can link aurally disparate groups of sounds. While the sounds returned in the first two mechanisms are related by aural similarity, the sounds returned using this final method are related lexically. Figure 3 shows two aurally-related sub-graphs that are connected by a lexical edge.

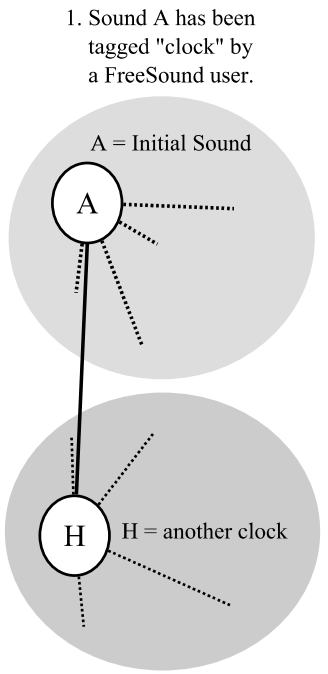

Figure 3. Lexically Related Sub-graphs

Once these three steps are completed on the initial vertex, they can be repeated on any other vertex in the graph. This allows the composer to build arbitrarily large sound graphs that contain however many sounds or sound-areas the composer desires.

One significant caveat to this approach that must be mentioned is that it assumes that the relationships culled from the FreeSound and Wordnik databases are salient to a potential listener. The music generated in this project draws out underlying relationships in those databases. It is a limitation of this work that the resulting music may only be successful to listeners who can comprehend those relationships. However, this is not a very severe limitation for several reasons. First, since the data is drawn from user-driven sites, I think it's fair to assume that the relationships represented in the databases are at least understood by the users of those sites. Second, if the relationships in the databases were useless, then the sites would cease to be popular resources on the web. Finally, it's important to note that these two data sources could easily be replaced by other sites, such as ccmixter and dictionary.com, or any databases that are deemed more rich in a potential artistic context. An expansion of this project might compare audio similarity algorithms, and evaluate the quality of word relationships returned by Wordnik.

Graphs created in this way capture the relationships between sounds and organize them in a way that a listener can understand. However, these graphs are simply static data structures. In order to create art, the networks must be activated.

Creating Music by Activating Sound Graphs

After a group of sounds has been organized in a graph structure, those sounds must be accessed and triggered in some way. Activating these sounds is the second part of music creation using the system described here. The sounds can be activated in a limitless number of ways. A program might randomly wander from vertex to vertex, triggering sounds as it moves. Or a search strategy might be used to wander through the graph in a directed way. The goal of the activation step is to create a collage of sounds. In early versions of my program it wrote collages directly to audio files; however, the current version writes to a format that can be subsequently manipulated in a digital audio workstation (DAW).

The collage generation step creates a Reaper Project file. The Reaper file format was created by Cockos for use in the Reaper DAW. Reaper files can contain all the track, media, midi and processing information that might be created in a music production project. In this case, the software creates many tracks in the output file, and organizes the audio collage onto those tracks. So the output file contains an editable Reaper project which gives me the opportunity to clean up or modify the algorithmic composition. Typically, several files in the collage will be

unreadable, and several will have audio errors. I usually remove these files from the project before rendering it. On several pieces where I wanted to explore a collaboration between myself and the software, I edited the output more heavily, but this is atypical.

I have explored the collage-creation step with several algorithms. In one of my first experiments, I coded an algorithm similar to a depth-first search which seeks to the end of the longest path in the graph. Specifically, I created a graph consisting of one long chain of sounds by using recursive similarity search to a depth of thirty. A depth first search activation strategy then starts from the initial vertex, and seeks to the final sound in the chain. The sounds along the path are activated in turn when 66% of the previous sound has been played. This graph creation and traversal strategy is aesthetically interesting, because it essentially sonifies the similarity algorithm employed by FreeSound. In pieces generated this way, the listener can hear how the similarity algorithm is similar and different from his own understanding of aural similarity.

My next attempt at sound graph activation employed cellular automata. Cellular automata are deterministic programs where many cells act simultaneously. Every cell has a state and a rule set. Each cell may have a different state, but all cells have the same rule set. The rules determine the state of a cell at any time *t,* based on the state of that cell's neighbors at time *t-1*. It's difficult to code a standard cellular automaton that works on a sound graph because in the standard model of cellular automata, all cells are connected in the same way. In other words, they all have the same number of neighbors. In the sound graphs created by my software, this isn't the case. So finding a rule system that leads to the emergence of interesting non-cyclical patterns is very difficult. In most of the rule sets I tried, the automaton either quickly moved to a steady state where all cells were off, or it quickly devolved into an oscillating pattern where the same cells were activated repeatedly. Neither of these outcomes are desirable in music composition where variation over time is a fundamental value of the composer. As a result of these early experiments with cellular automata, I turned to a variation on cellular automata suggested by a colleague.

In this variation on cellular automata, a *king* vertex is selected that will act as the center or beginning of the activations. After the king is activated, activations radiate outward from the king, ignoring previously activated vertices. When 66% of the king sound has been heard, all vertices that are neighbors to the king are activated. When 66% of the longest sound in that group has been heard, all vertices that are neighbors to those sounds and haven't yet been activated, are activated. This continues until all sounds are heard. In other words, the king is activated first, followed by all sounds at distance one from the king, then all sounds at distance two, then all sounds at distance three, and so on until all sounds have been activated. This activation strategy is useful as a sonically unique alternative to the other two strategies explored here.

The third graph activation strategy I will describe employs swarm intelligence. This activation strategy is successful because it allows all the different relationships embodied in a graph to emerge in the resulting collage. This leads to novel juxtapositions of aurally and lexically related sounds. The program uses a variation on Craig Reynolds' boids algorithm. The boids algorithm has three rules: separation, alignment and cohesion. The separation rule says that boids shouldn't run into one another. The alignment rule says that a boid should move in roughly the same direction as the boids around it. The cohesion rule says that the boid should stay near the boids it can see. These rules must be slightly modified to work within the discrete space of a graph, as opposed to the continuous space of a typical boids simulation. The separation rule becomes an over-crowding rule. If too many boids are on a vertex, then that vertex becomes undesirable. The alignment rule is eliminated because in a graph of the type used by my software, direction doesn't exist. The cohesion rule stays the same. Boids try to move with the swarm as long as the space isn't too crowded.

The results of this swarm algorithm mean that the boids move as a loose swarm around the sound graph. They move around in roughly the same area of the graph without all being on exactly the same space. In other words, they explore the distinct neighborhoods created in the graph creation step. In the remainder of this essay, I am going to show some of the graphs created by my software, and the music that resulted from activating these graphs.

Compositions

Toll

Toll is a short piece created by using the king cellular automaton. In this activation strategy, the initial sound is activated, followed by all sounds at distance one, then all sounds at distance two, and so on until all sounds have been activated. The piece is only 100 seconds long because each sound in the graph occurs exactly once, with many sounds often beginning at the same time.

The graph was created by searching for the term *chimes*. There are many high quality recordings of wind chimes on FreeSound. Although a recording of wind chimes was the first sound returned by the FreeSound, the graph also diverged into sounds relating to music boxes. Hence, various recordings of music boxes, particularly ones playing Christmas songs, occur throughout *Toll*. Although this was not my original intent when I started the piece, this is an example of how my software has surprised me.

Sound Graph Details for *Toll*	
Vertices	100
Edges	272
Degree Distribution	
Degree	Occurrences
1	63
2	13
3	9
5	1
6	1
7	1
8	3
11	2
12	4
13	2
18	1

Table 1. Sound Graph Details for Toll

Machine Song 1

Machine Song 1 was composed specifically for the California State University at Fullerton New Music Festival with the theme *Voice in the 21st Century*. I began the piece by searching for the word *sing*. In the final piece, the sound of a singing wine glass is juxtaposed with the sound of a recorded reminder call. It's easy to understand that the sound named *SingGlas1.wav* and the sound of a recorded human voice are both related to the word *sing,* however, the software made this connection through an unexpected path.

After finding the initial sound, *SingGlas1*, the program retrieved a similar sound called *Hoot_1.wav*, which contains an owl hoot simulated with a whistle. The singing wine glass and the simulated owl hoot were rated as similar because they are both continuous tones at similar frequencies. One of the tags on the latter sound is the word *hoot*. My software then searched wordnik.com for words related to hoot, and one of the hypernyms returned was *call*. When freesound.org was then searched for the word *call*, the recorded reminder call was the third result.

So the initial two sounds are related by aural similarity, while the third sound is lexically related. The connection between the word *sing* with a singing wine glass and a recording of the human voice was created by exploiting both similarity and lexical relationships.

Sound Graph Details for *Machine Song 1*	
Vertices	100
Edges	323
Degree Distribution	
Degree	Occurrences
1	66
2	6
3	8
4	4
7	1
8	1
9	3
10	2
12	1
13	1
14	1
16	2
17	1
18	2
19	1

Table 2. Sound Graph Details for Machine Song 1

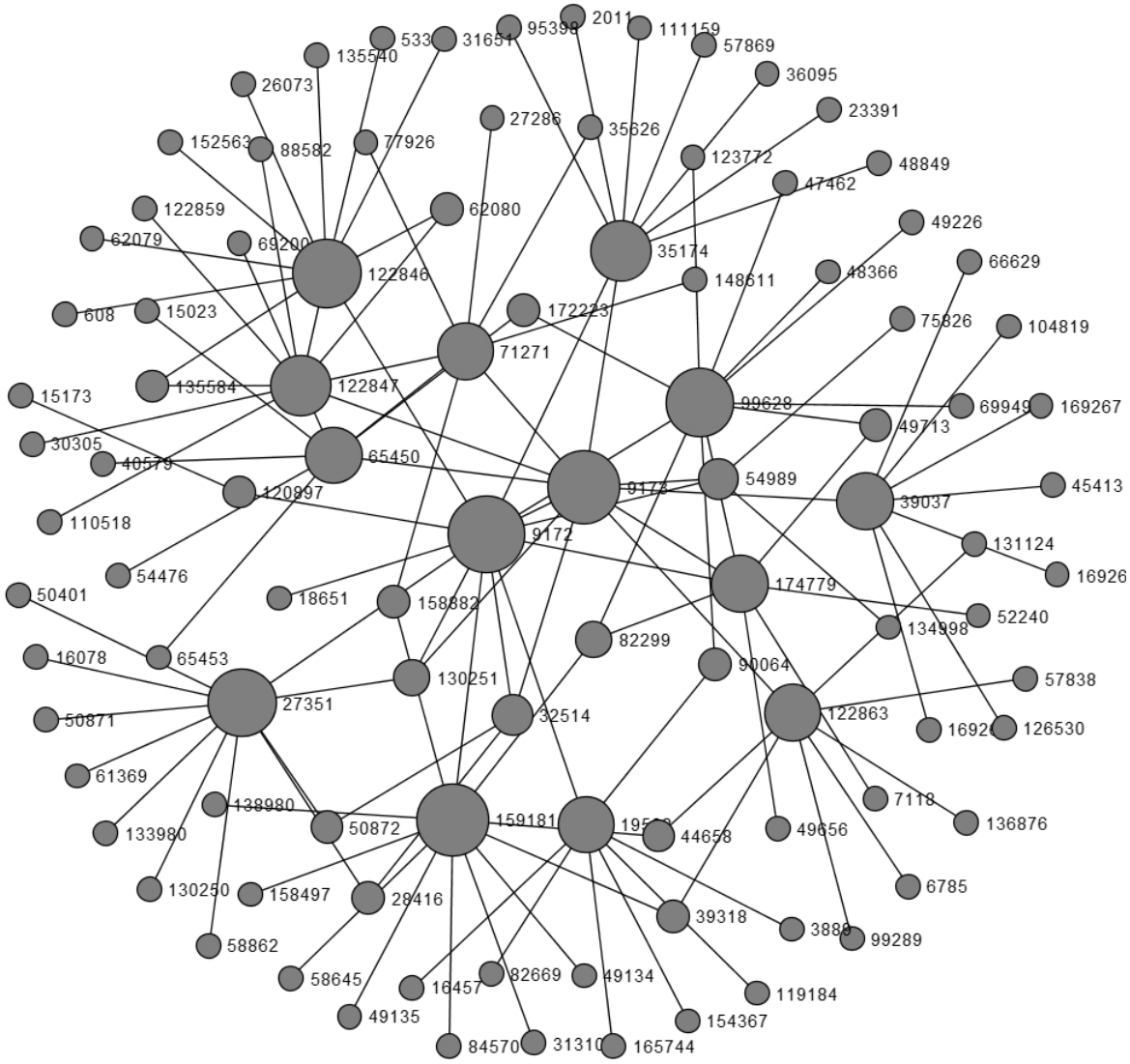

Figure 4. The Complete Graph for Glass Manufactory

Glass Manufactory

In *Glass Manufactory*, noisy sounds are reinterpreted by being placed in context with other wide-band noise sounds as well as glassy tones returned from the search term *glass*. The structure of the piece created by the swarm agents is clearly demonstrated by the stark contrast between these two sound areas.

The initial sound returned from freesound.org after searching for *glass* was *glass7.aiff*, which is a granular resynthesis of a recording of rubbed glass, according to the description provided by its creator. It is a smooth tone that is given a slightly alien quality by the granular synthesis. Through the initial similarity searches, it was connected to various other tones, including guitar strumming and other synthesized sounds. Eventually, by exploiting the hyponym *hourglass*, it was connected to a recording of sand pouring through an hourglass that was made with a contact microphone. Because of the way it was recorded, the sand in the hourglass sounds like white noise. Using similarity searches, my program built a neighborhood of similar sounds around this white noise sound.

The drama of the final composition comes from contrasting the areas of the graph made up of smooth glassy tones with the areas of the graph made up of white-noise-like sounds. The swarm imposed an interesting formal outline on the piece by traveling between these two sub-graphs within the larger creativity network. The graph used in *Glass Manufactory* can be seen in figure 4, where the labels on the vertices represent the file id in the FreeSound database.

Sound Graph Details for *Glass Manufactory*	
Vertices	100
Edges	243
Degree Distribution	
Degree	Occurrences
1	71
2	11
3	2
4	2
8	6
9	2
11	3
12	2
13	1

Table 3. Sound Graph Details for Glass Manufactory

Conclusion and Future Work

From my initial goal of writing music with all types of sound, I showed that it is possible to organize the wealth of sounds available online by taking advantage of aural similarity relationships provided by FreeSound, and lexical relationships provided by Wordnik. After organizing the sounds into graphs based on these relationships, I demonstrated three ways the sounds in the graphs can be activated to create new electroacoustic music. By activating sounds along a chain of similar sounds, my program can sonify the similarity algorithm used by FreeSound. A modified cellular automaton allows my software to generate clangorous collages that activate sounds in waves radiating outward from the initial sound. Finally, a modified boids algorithm allows my program to release a virtual swarm on a sound graph. This activation strategy allows musical form to emerge based on how the swarm navigates the neighborhoods within a graph.

These activation strategies are only three out of a potentially infinite variety of ways that sound graphs might be used to generate music. I have spent some time exploring other strategies for graph activation, but none have been as useful as the three listed here. I am still working on ways to use this algorithm to generate vernacular style electronic music with a tonal center and a steady pulse. I am also working on ways to use the concepts here to generate visual art. In another article I will show how these graphs are rooted in theories of creativity, and how structures like these graphs can be used as a general base for generating art of any type using content from the internet.

All music discussed here can be streamed at soundcloud.com/evanxmerz

References

Baker-Smith, Ben. 2012. Infinite Glitch. Accessed July 8, 2013. http://bitsynthesis.com/infiniteglitch/.

Barbosa, Alvaro. 2005. Public Sound Objects: a shared environment for networked music practice on the Web. *Organised Sound* 10(3): 233-242.

FreeSound API Documentation. Accessed April 18, 2013. http://www.freesound.org/docs/api/resources.html#sound-search-resource.

Hsu, Wun-Ting, and Wen-Shu Lai. 2013. Readymade and Assemblage in Database Art. *Leonardo* 46 (1): 80-81.

Jenkins, Henry. 2006. Fans, Bloggers, and Gamers. New York: New York University Press. Kindle edition.

Katz, Mark. 2010. Capturing Sound: How Technology Has Changed Music. Berkeley, CA: University of California Press.

Kerne, Andruid. 2000. CollageMachine: An Interactive Agent of Web Recombination. *Leonardo* 33(5): 347-350.

Kreidler, Johannes. 2008. Product Placements (2008). Accessed July 8, 2013. http://kreidler-net.de/productplacements-e.html.

Martınez, E.; Celma, O.; Sordo, M.; De Jong, B.; and Serra, X. 2009. Extending the folksonomies of freesound. org using content-based audio analysis. In *Sound and Music Computing Conference.* Porto, Portugal.

Roma, G.; Herrera, P.; and Serra, X. 2009. Freesound Radio: supporting music creation by exploration of a sound database. In *Workshop on Computational Creativity Support.*

Roma, G.; Herrera, P.; Zanin, M.; Toral, S. L.; Font, F.; and Serra, X. 2012. Small world networks and creativity in audio clip sharing. *International Journal of Social Network Mining* 1(1): 112-127.