Graph grammar representation for collaborative sample-based music creation

Gerard Roma
Music Technology Group
Universitat Pompeu Fabra
Barcelona, Spain
gerard.roma@upf.edu

Perfecto Herrera
Music Technology Group
Universitat Pompeu Fabra
Barcelona, Spain
perfecto.herrera@upf.edu

ABSTRACT

This paper proposes a music representation for collaborative music creation using shared repositories of audio samples. We explore the use of the graph grammar formalism to organize collective work on musical compositions stored as graphs of samples, and describe an experimental prototype that implements this concept. We then use the compositions created by different users with the prototype to show how this representation allows tracking and analyzing the music creation process. Potential applications of this include finding similarities between artists or suggesting sounds for a given compositional context.

Categories and Subject Descriptors

H.5.3 [**Group and Organization Interfaces**]: Web-based interaction; H.3.0 [**Information Storage and Retrieval**]: General; H.5.5 [**Sound and Music Computing**]: Methodologies and techniques

General Terms

Algorithms, Human Factors

Keywords

network music, collaborative composition, graph grammars

1. INTRODUCTION

There have been traditionally great expectations with respect to the possibilities of the internet to facilitate collaborative music creation. The two main problems can be defined as remote networked performance, which involves simultaneous presence of participants from different locations, and collaborative creation, which describes asynchronous creation of contents. While research continues on the former [14], it seems to advance at a slower rate on the latter. As generalized improvements of network bandwidth are stimulating new proposals in the market, most commercial offerings merely try to incorporate networking into the traditional multi-track audio sequencer interface that has become dominant for individual music creation. We believe that further research in the higher level aspects of interaction is necessary in order to exploit networked computers to create new forms of collective musical creativity. In this sense, two important questions must be asked that belong to different disciplines.
The first one is how to organize collaborative work from a practical point of view. Our observation in this respect is that collaboration already happens all the time in electronic music when producers re-use audio samples from other producers. The question is whether this practice can be generalized to more complex musical structures and, in that case, what strategies related to roles and teams can emerge. This can be investigated using human-computer interaction methodologies.
The second question is wether (and how), systems can exploit data generated by users to enhance both compositional and social aspects of music creation, for example by suggesting potential collaborators. This can be seen as a traditional information retrieval problem.
In this paper we propose using the graph grammar formalism as an initial step that allows both issues to be further investigated. Formal languages already facilitate high collaboration levels in music creation, for example in MUSIC-N style languages such as Max, Supercollider or Pd. User communities of these languages are very active and are continuously sharing and reusing their creations. Our aim is to allow for similarly re-using materials in sample-based composition. The rest of this paper is organized as follows. First, we review some related work in the field of networked music creation and briefly review the use of grammars in music. Then, we propose a general representation that embeds the graph grammar formalism in the process of music creation. We describe a prototype that implements the proposed representation to allow sample-based music creation on top of a large repository of audio samples. We use initial data created using this prototype to show how the proposed representation can be exploited in the analysis of users creations.

2. RELATED WORK

Many projects have researched internet-based collaborative creation. A recurring theme is that general availability of music creation tools allows to participate in music creation to a wider audience than traditional tools do [8][16]. In this sense, simple interfaces and large audio repositories can help exploring creation without the need of specialized musical training. Several projects have explored graph-based interfaces for web-based composition [5] [17]. However, these

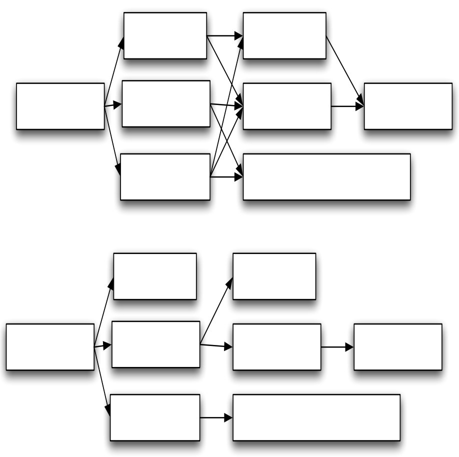

Figure 1: Multiple vs single (one antecedent per node) interpretations of a composition of audio samples (represented as rectangles) as a graph.

interfaces do not usually support re-using works to create higher level structures. This need emerged in the use of the previous version of our system [15], which supported the creation of small graphs of samples. Formal grammars are a common choice for representing this kind of hierarchies in music. The use of formal grammars for music creation has a long history, parallel to their use in music analysis. Early work explored their use to compose music using alphabets of sound objects [3]. Such approach required the definition of sonological rules, which in practice may involve a map of symbols to sounds that is private to the composer. Usage of grammars has further evolved largely on the choice of conventional vocabularies, such as western [7] [2] and indian [10] classical alphabets. In order to represent parallelism in music, string grammars are often adapted in different ways. Graph grammars can be used as a more general framework to represent parallelism. Graph grammars were introduced by Pfaltz and Rosenfeld in the late 1960s [13] as an extension of traditional grammars to languages of graphs. In [18] their application to music processing graphs is discussed at a theoretical level. On the other hand, in [11] [1], they are used in the analysis of classical scores. One problem of this kind of analysis is that traditional scores do not encode edge information. Thus, a given polyphonic fragment can be interpreted as many different graphs. For the creation of new music, we propose an interface that forces the user to define precedence relations among sounds to create music compositions, which can be collapsed into single nodes. This difference is depicted in Figure 1 for the traditional *brick wall* representation of an audio sequencer. Given an existing composition, many connections could be added between the different sounds. Our proposal is to design interfaces that allow authors to specify their ideas with respect to precedence and parallelism by defining the connections during the creation process.

3. REPRESENTATION

3.1 Facets

One important issue for using grammars in sample-based composition is mapping sounds to a discrete alphabet. Here our main assumption is that music can be better understood and represented through distinct facets or dimensions, and that musical rules operate in these dimensions. For example many rules of classic western music are defined independently for pitch, rhythm and timbre, while a lot of modern computer music is considered to be an exploration of timbre.

From a computational point of view, the use of different facets to define rules also solves the combinatorial explosion that would result from trying to find patterns of arbitrary descriptors. If, for example, we were to infer rules using social tags attached to sounds, for any graph with N sounds, and assuming M tags are attached to any sound, we would have M^N possible rules. By restricting our description to a single discretized facet, each sound is assigned one value (e.g. a pitch value) and one rule is extracted for each facet. Considering our music universe on top of a shared database of sounds, we define the grammar alphabet as a partition of the database. Given the current wealth of tools in audio description and data mining there is a wide number of possible descriptors and algorithms that can be used to define partitions. We describe our experiment in section 5.1 using a hierarchical clustering algorithm. The choice of a hierarchical algorithm allows us to define grammars at different levels. This granularity is important, because it allows us to choose an appropriate level given the application and the amount of data. For instance, a large network of users could be be described as a hierarchy of communities. Rules could be defined at a user level, team level or at different community levels. It can be expected that at lower levels of community structure, more specific labels can be used to find characteristic patterns. This level of generality in the labels of sounds can be named the *lexical level*. In contrast, the *syntactic level* refers to the levels of nesting found in a given composition.

3.2 Grammar framework

A musical fragment can be described as a graph where nodes are sound objects, and edges represent transitions between objects. In our proposed approach, transitions are fired when the source node finishes playing, and there is no specified temporal grid. This system is quite similar to the one proposed by Roads in the 70s [3], but there a syntax construct was used instead of the graph representation for parallelism. One advantage of this approach is that instead of focusing on time as a container, the representation stores the order of events. Given a large database, musical creation can be more based on exploration of the database, and the temporal structures that result from the found objects. In this version, we constrain music graphs to have a maximum in-degre of one for each node. The reason is that several transitions going from several nodes to a single one introduce artifacts to the playback model. A node could be triggered by a preceding node and then re-triggered by another one in ways that would not be obvious by looking at the representation. On the other hand, this forbids the creation of loops, which makes the patches contractable [13]. Repetitive musical patterns can be obtained simply by duplicating nodes. This restriction could be removed if transition probabilities are used for the edges. However, in the present work we stick to a deterministic representation and restrict musical graphs to trees.

We call this data structure a *sample patch*. Formally, a sam-

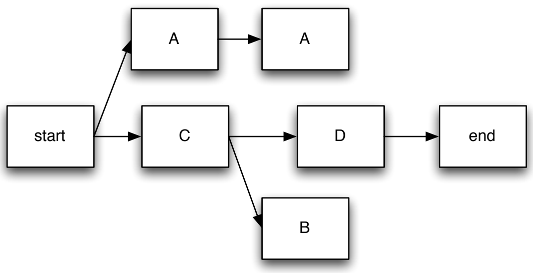

Figure 2: Example of a sample patch.

ple patch can be characterized as a directed tree with node labels: a tuple (V, E, f) where V is a set of nodes that represent audio clips, E is a set of edges that represent transitions between the clips, and f is a labeling function that assigns a label to each node at a given lexical level. An example patch is depicted in Figure 2. For simplicity, we consider this tree to be *ordered*, i.e. we preserve the position of each node with respect to its siblings, even when it has no musical meaning (all siblings start at the same time regardless of the order). This allows us to use a fast algorithm for computing the edit distance between trees. The tree edit distance can be described as the lowest cost sequence of edit operations (add a node, delete a node, rename a node) that will transform a given tree into another one. We use the Zhang-Shasha algorithm [19] with unit cost for each operation.

We model collaborative creation of sample patches as a context-free graph grammar. This is different to the traditional use of grammars in that we do not try to infer rules from preexisting data. Instead, by using the sample patch as a musical representation, we constrain users to work within the grammar framework. Each user creation can be considered as a rule that expands a node into a sample patch. This new node can be used in another patch.

Thus, the system can be formally characterized as a Node-Label Controlled (NLC) graph grammar [4]: a tuple $(\Sigma, \Delta, P, \sigma, E)$ where Σ is the alphabet of node labels, $\Delta \subset \Sigma$ is the set of terminal labels, P is a set of productions, σ is the starting graph, and $E \subseteq \Sigma \times \Sigma$ is an embedding function. We now describe the interpretation of these elements in the interactive system.

The alphabet of terminal node labels, is produced by the labeling function for a given lexical level and facet. At the lowest lexical level, the label simply identifies the audio clip, so for the audio document, the label is simply "filename#start:end" where start and end are indicated in samples. The sample patch can contain also silences which are labelled simply as "#0:end". Finally special labels are assigned to start and end nodes, which are described below. The labels for non-terminal nodes at the lowest lexical level are actually assigned by the user when a sample patch is created. Labeling terminals at higher lexical levels can result in two isomorphic patches being equal. This means that the two corresponding non-terminal variables that produce the patches are the same at that level. This can be checked by computing thet distance between patches: if the distance is zero at a given lexical level, a new variable is generated which includes both patches.

The starting graph has no direct meaning in the musical creation process. This variable can be designed to construct a grammar for a specific purpose. For example, if the initial graph is a finished piece, the grammar will consist of all the rules used to create that piece. To characterize a given author or team, the starting graph can be defined as a variable that can be replaced by any of the complete pieces by this author or team.

As mentioned, the set of productions is simply a collection of sample patches. Whenever a user creates a new composition, it is collapsed into a single node that can be used elsewhere. Hence, the patch is a context-free rule that defines the expansion of the node.

Finally, the embedding function indicates how a patch is embeded in another patch. We simply use two virtual endpoint nodes for each patch to indicate the embedding. Thus, for a given production that replaces a node n, with a sample patch s, $E = \{(i, start_s) \forall i \in In(n)\} \cup \{(o, end_s) \forall o \in Out(n)\}$, i.e. incoming edges of n are connected to the start node and outgoing edges to the end node. This process is depicted in Figure 3. Here, the host patch includes the example patch of Figure 2 collapsed as a node. The start and end nodes guide the embedding of the collapsed patch in the host patch.

Using this representation has two main advantages over traditional audio sequencing for collaborative creation. First, it allows to naturally share and re-use creations at different levels of complexity. As shown in Figure 4, the parse tree of a musical piece allows tracking all the authors that have participated in it. Second, by keeping the structure of the piece we can compute similarity measures that take this structure into account. While a full evaluation of these two aspects is outside of the scope of this work, we analyze the viability of this approach through an informal case study. In the next

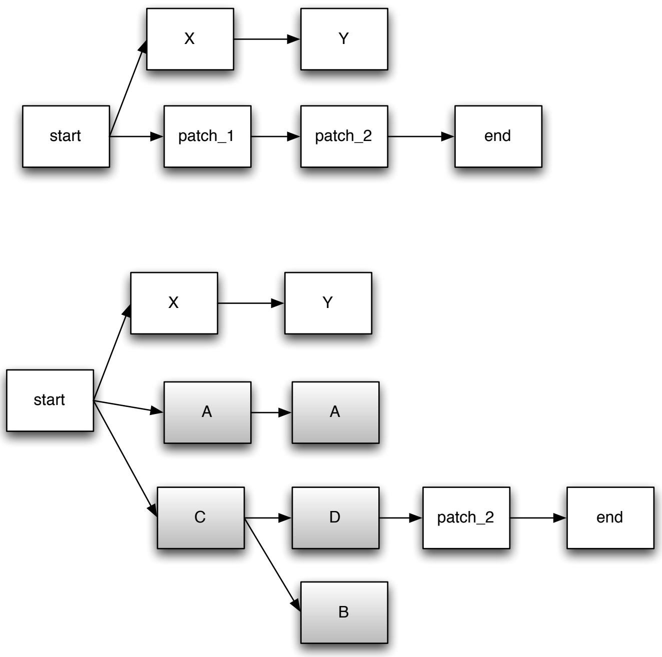

Figure 3: Replacement of a patch in a host patch: the node labeled as $patch\_1$ is replaced by its corresponding graph (in grey).

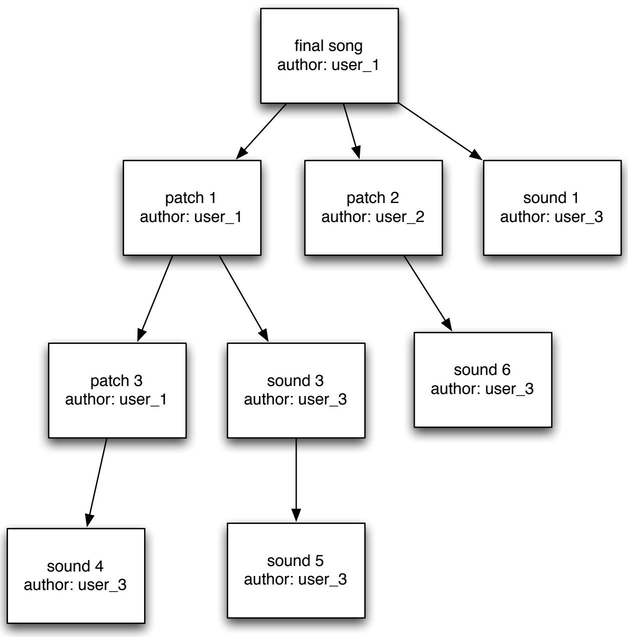

Figure 4: Authorship tree of a finished piece.

section, we describe an implementation of these ideas. We then illustrate their use through an analysis of initial pieces created with this prototype.

4. PROTOTYPE

We implemented a prototype that allows the creation of musical works using the described representation on top of a large database of sounds. This database contains sounds from Freesound[1], a popular sample exchange website currently containing more than 90000 samples. The interface consists of a flash application that connects to a pyhton back-end. The interface is based on three panels that describe a creative workflow. The three components share a tray that holds currently selected sounds. We now briefly describe the main components of the interface.

- Sample tray
  By default, the tray contains a blank node that represents silence. The tray allows duplicating any object and particularly silence objects of different durations can be created.

- Search panel
  The search panel allows to retrieve samples and patches from a database. Sounds can be searched by tag, file name or user name, and a sound duration limit is specified (by default 10 seconds). Patches can be searched by file name or user name. Selected objects are dragged to the tray.

- Edit panel
  The edit panel allows the user to modify the start and end points of a sample, thus creating a new clip. This operation produces a new entry in the global alphabet of terminal nodes. Since the user may be interested in adding several instances of this terminal to the patch, the edit settings modify a master copy represented by the visual element in the tray.

- Composition panel
  In the composition panel the user can edit a sample patch with the samples and patches in the tray. Two rectangles in this panel represent the start and end nodes. The user is asked to create a composition where a path exists from the start to the end. When saving the patch the user decides whether to share it with the community or to keep it for herself. Users can continue to edit their own patches as long as they are not shared. When loading a patch to the tray, all the sounds and sub-patches used in that patch are also loaded. Shared patches can no longer be modified (although new versions can be created through duplication). The reason is that modifying a shared patch could unexpectedly modify someone else's patch. Given more communication features, modification of shared patches could be enabled in some cases for faster collaborative creation.

[1]`http://www.freesound.org`

When the user saves a patch, the object structure is encoded in a JSON file, the audio is rendered to a waveform, and a thumbnail of the composition panel is generated. All files are sent and stored in the server.

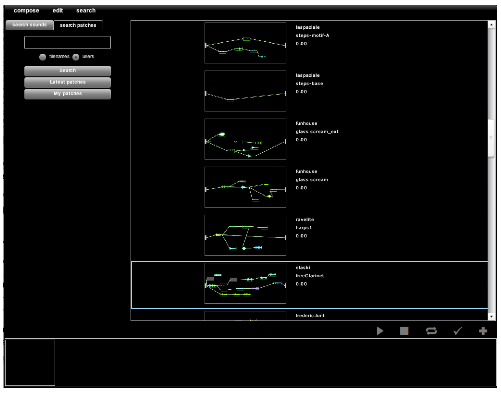

Figure 5: Screenshot of the search panel

5. USAGE ANALYSIS

We asked several people to try our initial prototype. During the initial trial period, we collected about 65 patches from 15 users. Of these users, most generated one or two patches, and two went on to create 16 and 21 patches respectively. We will call these users "A" and B". The first one recognized himself as expert using music production tools, but didn't have any programming or computer science background, while user B is a music technology graduate student. During this informal test, it was clear that people understood and used the possibility of nesting compositions at different levels. However we also found that more intensive use of this feature would require a pre-existing motivation for collaborating, for example an already established team. A more formal evaluation is needed in order to understand

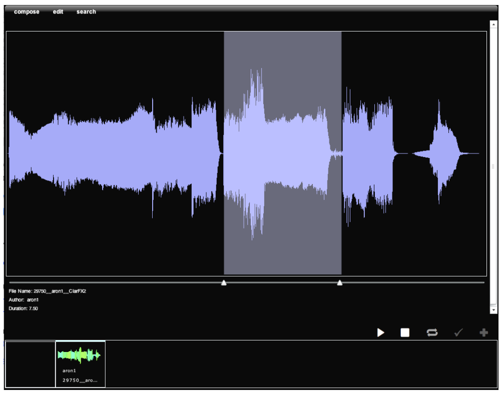

Figure 6: Screenshot of the editing panel

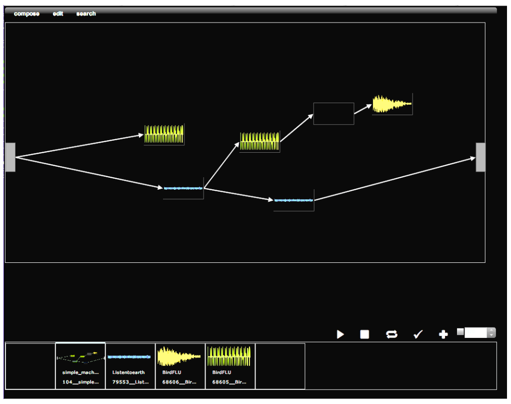

Figure 7: Screenshot of the composition interface

the role of a hierarchical representation, as well as the influence of different features of the user interface. For example, Figure 8 shows a patch created by user A, who reportedly "forgot" about the possibility of nesting patches. Editing this kind of patch quickly becomes tedious, although adding some features to the interface could help. As a comparison, the patch below exploited this feature conveniently, which allows concentrating on the higher level structure and, in the case of individual use, the modification of the repeated portions of the piece at once. It became apparent that this may require some practice. With the data collected during the test, 69% of patches contained no nested structures, 18% contained one level and the remaining 8% more than one level. Almost half of the patches with a syntactic level higher than zero were generated by user B. On the other hand, some users nested other patches in their first creations. In all, 54% of all patches participated in some nesting relation, either contained or as containers. These tend to have a lower number of nodes (6.9 mean, 4.8 standard deviation) than the others (10.5 mean,7.6 standard deviation).

5.1 Sound clusters

In order to find rules in users creations, we generate labels at a given lexical level by partitioning the database. We extracted a number of sounds from the database for our experiments. We restricted the base sounds to those shorter

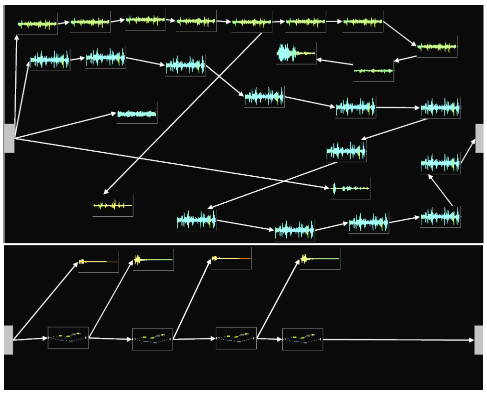

Figure 8: Using (upper figure) / not using (lower figure) nested structures.

than 10 seconds, in order to avoid long field recordings and also to minimize the variability of frame-level features. We resampled all sounds to 44100Hz/16bit, and converted them to wav. In total we analyzed 34164 samples from a wide variety of sources. This diversity required the use of a general musical facet. Initial experiments with chroma features[6] showed that the use of sounds with a well defined tonality was very low in the initial test period. Thus, we opted for using a general representation of the spectral energy, discretized in bark bands [21]. Our implementation splits the two lowest and the highest bands the original bark scale. This gives us a vector of 27 values per frame, which we average over time to obtain a vector of 54 features (mean and variance). We clustered these vectors to obtain a hierarchical partition of the database. Since the cost of traditional hierarchical clustering algorithms is prohibitive for the size of this dataset, we used bisecting k-means, which uses k-means to repeatedly split the data, producing a hierarchical partition. This algorithm has been shown to perform better than traditional hierarchical clustering approaches [20], especially when modifying the traditional k-means objective function. Also, it is well suited for some characteristics of our data, such as high dimensionality, large size and a considerable amount of noise typically associated with the web and its users. We use the implementation available in the CLUTO [9] clustering package.
One general problem with clustering is that a clustering solution is always found, regardless of wether the data is naturally clustered or not. Thus, without some objective measure of quality it becomes difficult to choose among different parameters. Especially critical is the choice of the internal validity function optimized by bisecting k-means. However, we do not have an objective classification of the data to assess the validity of the clusters. We have resorted to social tags (provided to describe the sound content by each sound's author and users) as a reliable cue to choose an appropriate function. While noisy and incomplete, tags provide valuable information regarding the content of the sounds that is independent of what we can automatically extract from the signal. We use a similar method to the one reported in [12] using the most generally used tags in the database. The rationale is testing the associations between clusters and assigned tags. If clusters are capturing some semantic regu-

larities and group toghether coherent sounds with respect to certain features, then we should observe that certain clusters tend to concentrate certain tags. This hypothesis can be tested using the χ^2 statistic. For a given clustering, we compute a contingency matrix by counting the frequencies of each tag in each cluster. From this matrix we compute the χ^2 statistic:

$$\chi^2 = \sum \frac{(f_o - f_e)^2}{f_e}$$

where the observed frequencies f_o are the frequencies of tags in each cluster, and the expected frequencies f_e are the products of the totals for each row and column divided by the total number of elements in the matrix, $f_{e(i,j)} = \frac{T_i T_j}{N}$. A higher value of the statistic implies a lower probability that the difference between the actual and expected values is the product of chance, i.e., there are some robust associations or co-ocurrences between the clusters and the tags. Thus, a higher value of χ^2 indicates stronger support against the null hypothesis that the tags are equally distributed among the clusters. We use this value to hint towards better clusterings. Figure 9 shows the evolution of the statistic for increasing numbers of clusters given each of the k-means objective functions described in [20], labeled as $i1, i2$(internal criteria), $g1, g1p$ (graph-based) and $h1, h2$ (hybrid internal/external). Since the confidence on the statistic decreases with the number of clusters (as the number of observations in the contingency matrix approaches a minimum in some cases), we chose the function that works better with a lower number of clusters. This function maximizes the pairwise similarities of objects inside a cluster:

$$I_1 = max \sum_{r=1}^{k} n_r (\frac{1}{{n_r}^2} \sum_{d_i, d_j \in S_r} cos(d_i, d_j))$$

where S_r is a cluster, n_r is the number of documents in that cluster, and d_i, d_j are documents in that cluster

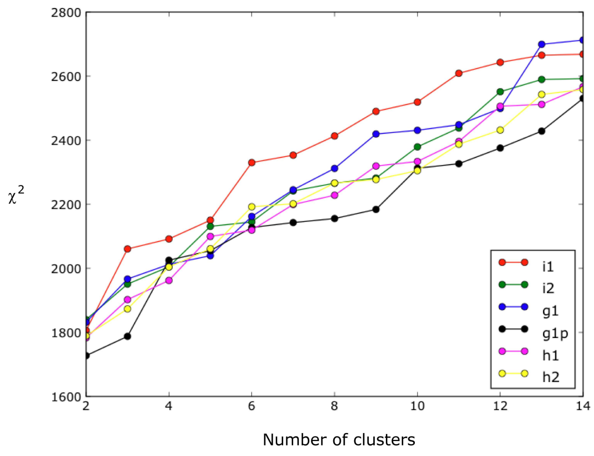

Figure 9: Performance of diferent clustering criteria along the number of clusters measured by the χ^2 statistic.

Figure 10 shows the dendrogram cut at 10 clusters and the corresponding centroids. Clusters tend to form around high values of specific or neighboring bands, both for mean and variance.

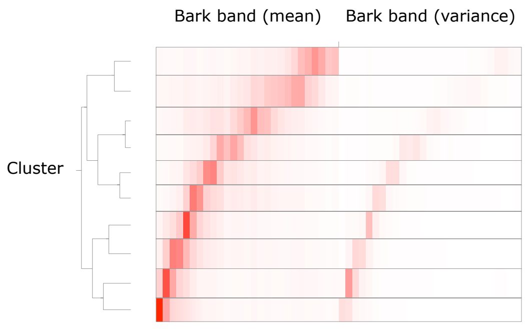

Figure 10: Dendrogram and centroids of 10 clusters. Color strength represents the value of the mean (left half) and variance (right half) of the spectral energy at each bark band. Centroids are sorted along the veritcal axis.

5.2 Label generation

Given a clustering of the sounds in the database, we can generate labels for all the nodes and patches in the database. Since users are allowed to clip sounds in the composition process, they are effectively generating new sounds. We analyzed all these sounds to extract their bark bands descriptor, and assigned them to the cluster with the closest centroid. Thus, for a partition of 10 clusters, all terminal nodes are assigned a cluster index from 0 to 9.

Since the clusters include many files, it is now possible that some of the generated patches are identical at this lexical level. As mentioned, this can be found by computing the distance between the trees, in ascending syntactic order. Because each patch must be saved before being reused in another patch, this can be done at creation time, as all nested patches will be already resolved for the current patch. For the present analysis we traversed the syntactic levels starting from the patches that contain no nested patches, up to the patches with maximum depth. At each level, we computed the distance with the other patches. Surprisingly, we didn't find any cases with zero distance using 10 cluster labels. In order to gain some flexibility, less strict rules could be used to cluster non-terminal nodes, for example using a certain distance threshold. This would allow identifying more general patterns at higher syntactic levels, and thus more general generative rules.

One question that arises is how many of the labels are actually used in the generated patches. Since the clustering is done using the whole database, it could be that not all the labels would be present in the small set of sounds actually used in the compositions. We found that all clusters were similarly represented in patches submitted by users, except for one which was used three times more often than the rest (Figure 11). This cluster includes generally sounds with energy in the higher bands of the spectrum; in the training dataset it does not have a bigger size than the rest of clusters. After looking at the patches that use these sounds, this preference seems to indicate a certain compositional strategy for rhythmic patches, where a few low frequency sounds (e.g. bass drum) are repeated while a greater diversity of higher frequency noises is used. We also looked at the usage of different clusters by users A and B. While both seem to follow the general trend with respect to cluster 3, they otherwise exhibit different patterns. These patterns in the usage of each cluster may be used to represent the preferences of individual users or groups.

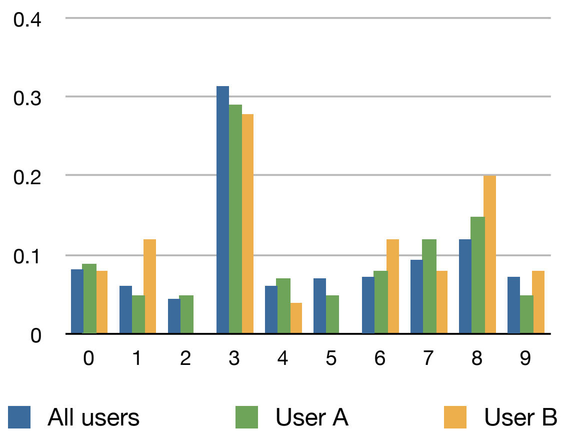

Figure 11: Fraction of sounds of each cluster used in sample patches by the whole community and users A and B.

5.3 Patch similarity

Having all patches labelled allows us now to compute the tree edit distance between patches, obtaining a similarity measure that takes their structure into account. Using a similarity measure can be used to enhance the creation of a patch, by suggesting suitable sounds from patterns found in other compositions. By using the current patch as a query, a similar patch can be used to find ideas not presently considered, for example the sounds that are most different from the ones in the current patch. We analyzed the tree edit distance between patches given 10 cluster labels. In general, the distance tends to be small for smaller patches, reaching the minimum for trivial ones, and large for more complex patches. Thus, it favors the preferred strategy of "nesting soon". Hence, with respect to authors, the average distance between patches was significantly lower than the global average for author B, who tended to use nesting and smaller patches, but higher for user A who did the opposite. Also, as expected, the average distance grows proportional to the number of clusters (i.e. the more diverse are the labels, the larger the average distance) due to the added renaming operations.

For small patches, this measure can be used to obtain insights on the different strategies employed by users. For example Figure 12 shows two pairs of patches with low edit distance, one from each user A and B in both cases. In the first one one sound is used as an initial section of the patch after which a number of sounds are triggered simultaneously. In the second, one "base" sound is used to bridge start and end. This helps getting started because the interface creates a loop by connecting the end with the start in the moment of creation. Both patches include a sequence of three sounds in parallel to the base loop. As can be seen clearly in both examples, the labels generated by cluster indices do not play an important role in patch similarity. We plan to adapt the edit distance to this application by taking into account node-to-node similarity based on the bark band vectors.

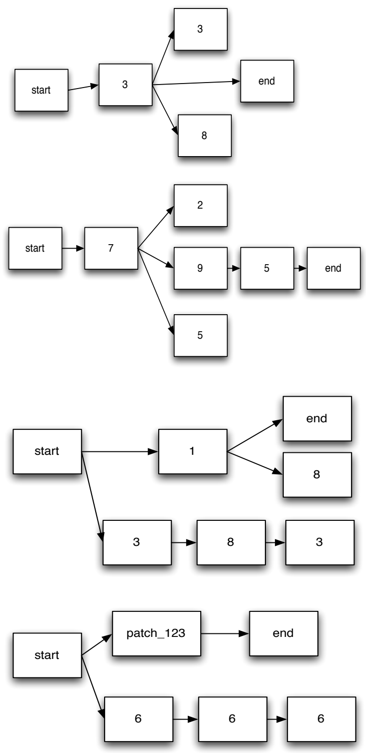

Figure 12: Two pairs of patches with high structural similarity.

6. CONCLUSIONS AND FUTURE WORK

We have proposed a formal representation of music aimed at sample-based, collaborative composition, and described a graphical interface that exposes this representation to internet users. Initial tests with this prototype showed that the framework extends on the common practice of reusing materials in computer music and allows bottom-up, distributed creation of sample-based music. The information about co-authorship of such compositions is retained at all levels. In addition, we have shown how this representation allows the analysis of user-submitted compositions.

As suggested in the introduction, we plan to extend this research in two directions. On one hand, the framework provides a basic way to organize collaboration, but we want to explore what kind of strategies could be implemented on top of that, such as the formation and growing of teams. A deeper study from the point of view of user interaction would also allow refining the different interface elements to improve the understanding and usefulness of the representation. On the other hand, we hope to gather more information and to further evaluate the possibilities of mining common patterns in user-submitted compositions, while adding other musical facets to the analysis. With more data it would be possible to evaluate different similarity measures and also analyze how they can help in defining similarity between authors. The prototype can be used online at `http://radio.freesound.org`.

7. REFERENCES

[1] D. J. Cook and L. B. Holder. Graph-based data mining. *IEEE Intelligent Systems*, 15(2):32–41, 2000.

[2] D. Cope. *Computer Models of Musical Creativity*. The MIT Press, 2005.

[3] C.Roads. Composing grammars. In *Proceedings of the International Computer Music Conference (ICMC)*, San Francisco, 1978.

[4] J. Engelfriet and G. Rozenberg. Graph grammars based on node rewriting: An introduction to nlc graph grammars. In *Proceedings of the 4th International Workshop on Graph-Grammars and Their Application to Computer Science*, pages 12–23, London, UK, 1991. Springer-Verlag.

[5] J. Freeman. Graph theory: Linking online musical exploration to concert hall performance. *Leonardo*, 4(1), 2008.

[6] E. Gómez and P. Herrera. Estimating the tonality of polyphonic audio files: Cognitive versus machine learning modelling strategies. In *Proceedings of the 5th International Conference on Music Information Retrieval (ISMIR04)*, 2004.

[7] S. R. Holtzman. A generative grammar definition language for music. *Journal of New Music Research*, 9(1):1–48, June 1980.

[8] S. Jordà and O. Wüst. Fmol: A system for collaborative music composition over the web. In *Proceedings of Web Based Collaboration DEXA 2001*, Munich, Germany, 2001.

[9] G. Karypis. CLUTO - a clustering toolkit. Technical Report #02-017, Nov. 2003.

[10] J. Kippen and B. Bel. Modelling music with grammars. In A. Marsden and A. Pople, editors, *Computer Representations and Models in Music*, pages 207–238. Academic Press, London, 1992.

[11] S. T. Madsen. Automatic discovery of parallelisms and hierarchy in music. Master's thesis, University of Aarhus. Department of computer Science, Denmark, May 2003.

[12] G. Pallis, L. Angelis, and A. Vakali. A probabilistic validation algorithm for web users' clusters. In *In Proceedings of the IEEE international conference on systems, man and cybernetics (SMC*, pages 4129–4134. IEEE, 2004.

[13] J. L. Pfaltz and A. Rosenfeld. Web grammars. In *IJCAI'69: Proceedings of the 1st international joint conference on Artificial intelligence*, pages 609–619, San Francisco, CA, USA, 1969. Morgan Kaufmann Publishers Inc.

[14] A. Renaud, A. Carôt, and P. Rebelo. Networked music performance: State of the art. In *Proceedings of the AES 30th International Conference*, 2007.

[15] G. Roma, P. Herrera, and X. Serra. Freesound radio: supporting music creation by exploration of a sound database. In *Workshop on Computational Creativity Support (CHI2009) (accepted)*, 2009.

[16] A. Tanaka, N. Tokui, and A. Momeni. Facilitating collective musical creativity. In *MULTIMEDIA 05: Proceedings of the 13th annual ACM international conference on Multimedia*, pages 191–198. ACM, 2005.

[17] N. Tokui. Massh!: a web-based collective music mashup system. In *DIMEA '08: Proceedings of the 3rd international conference on Digital Interactive Media in Entertainment and Arts*, pages 526–527, New York, NY, USA, 2008. ACM.

[18] F. Wankmüller. Application of graph grammars in music composing systems. In *Proceedings of the 3rd International Workshop on Graph-Grammars and Their Application to Computer Science*, pages 580–592, London, UK, 1987. Springer-Verlag.

[19] K. Zhang and D. Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM Journal on Computing*, 18(6):1245–1262, 1989.

[20] Y. Zhao and G. Karypis. Evaluation of hierarchical clustering algorithms for document datasets. In *CIKM '02: Proceedings of the eleventh international conference on Information and knowledge management*, pages 515–524, New York, NY, USA, 2002. ACM.

[21] E. Zwicker and H. Fastl. *Psychoacoustics, Facts and Models*. Springer-Verlag, 1990.