AudioQuilt: 2D Arrangements of Audio Samples using Metric Learning and Kernelized Sorting

Ohad Fried
Princeton University
ohad@cs.princeton.edu

Zeyu Jin
Princeton University
zjin@cs.princeton.edu

Reid Oda
Princeton University
roda@cs.princeton.edu

Adam Finkelstein
Princeton University
af@cs.princeton.edu

ABSTRACT

The modern musician enjoys access to a staggering number of audio samples. Composition software can ship with many gigabytes of data, and there are many more to be found online. However, conventional methods for navigating these libraries are still quite rudimentary, and often involve scrolling through alphabetical lists. We present AudioQuilt, a system for sample exploration that allows audio clips to be sorted according to user taste, and arranged in any desired 2D formation such that similar samples are located near each other. Our method relies on two advances in machine learning. First, *metric learning* allows the user to shape the audio feature space to match their own preferences. Second, *kernelized sorting* finds an optimal arrangement for the samples in 2D. We demonstrate our system with three new interfaces for exploring audio samples, and evaluate the technology qualitatively and quantitatively via a pair of user studies.

Keywords

Sound Exploration, Sample Library, Metric Learning, Kernelized Sorting

1. INTRODUCTION

The age of "big data" has brought upheaval in many aspects of our lives including photography, retail, social networks and music. The modern musician or composer has at her disposal a vast array of sound samples, and acquiring more requires little effort. Once the acquisition process becomes easy, the burden passes to the next stage, which is managing, searching, and exploring this data. A huge audio database is rendered nearly useless if it is impossible to navigate and find a desired clip.

Proper organization of audio samples is a challenging problem because similarity between samples is largely subjective. Audio features that are important to one user may be less important to others. Additionally, it's unclear how a sorted collection should be arranged on a computer screen in order to make best use of the space, and to maximize the browsing experience. In this paper we offer two techniques to address these problems. First, *metric learning* (Section 3.1) allows users to specify their own groupings of samples. The system then learns which features are important, by upweighting those which are most useful for classification, and ignoring those that are not important, and can then apply these weightings to new, as of yet unseen, samples. Next, we use *kernelized sorting* [18] (Section 3.2) in order to place samples onto target locations on a 2D plane, such that similar samples are near each other. This means that the samples can be arranged in any arbitrary pattern, such as a grid, a circle, a line, or a cluster. Kernelized sorting places the samples on the targets locations in a manner that respects the distances learned by metric learning. This approach provides particular flexibility when working with interfaces where real estate is scarce, or when building tangible controllers that have discrete buttons.

Our main contributions are (1) using metric learning to transform descriptors into a better space via user guidance and (2) using kernelized sorting to place the samples in a specific arrangement. While both techniques are known in other fields, their combination and usage for sound exploration is novel. Finally, we demonstrate two novel interfaces for audio navigation supported by our approach, and evaluate it both qualitatively and quantitatively via user studies.

2. BACKGROUND

In this section we discuss previous work and established methods for sound exploration and layout. We also provide background regarding metric learning and set matching, the two main components of our approach.

2.1 Sound Exploration and Layout

Sorting samples by timbral similarity, and visualizing them in a 2D plane is a well-studied field [9]. The process usually involves three steps. The first step is feature extraction. Commonly used features include bark-scale coefficients [15, 6], MPEG-7 [20, 15], and mel-frequency cepstral coefficients [8, 3]. Bark scale and mel-frequency cepstral coeffients are representations of the spectral envelope, compressed to smaller dimensionality, while MPEG-7 includes features which are relevant to perception of timbre [17].

Next, the similarity between each sample is expressed. Common approaches include embedding the sample in multi-dimensional space and expressing the similarity as a distance such as euclidean [7, 8], generalized minkowski[16], mahalanobis [8, 20]. Sometimes the dimensions are reduced using PCA, multi-dimensional scaling [7] or similar.

Finally, the samples are represented in a 2D plane. The most popular approach is to use self-organizing maps [4, 6, 23, 24]. Our chosen method for embedding, kernelized sorting [18], differs from self-organizing maps in that it allows us to specify where the samples should be placed, and achieves a 1-to-1 mapping between sound samples and tar-

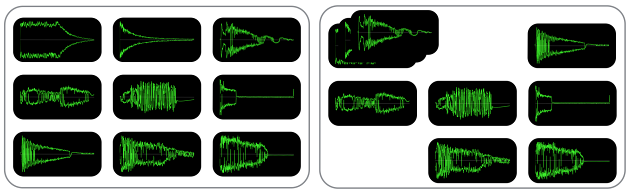


Figure 1: We produce a 1-to-1 matching between sound samples and grid locations (left), while methods such as Self Organizing Maps can match several or no samples to the same node (right).

get locations (i.e. all target locations are filled). This allows us to place the samples on dense structures such as grids. In comparison, in the training phase, self-organizing maps create a map by assigning values to a set of nodes (neurons), and in the mapping phase each element is mapped to its closest node. This means that many samples can be mapped to the same node, and some nodes might remain vacant (Figure 1).

2.2 Metric Learning

Embedding high dimensional features to 2D grid relies heavily on the measurement of similarity or distance. Many machine learning techniques such as K-means and KNN assume Euclidean space and L2 distance, where the dimensions are assumed to be independent and equally important. However, the Euclidean assumption is often not true. When measuring the similarities of snare drums samples based on spectrum and envelope features, the interactions between spectral dimensions and the difference scaling factors between the spectrum (energy) and the envelope (time) make the space non-Euclidean. Furthermore, when it comes to audio exploration, different users have different perceptions of the similarity. Metric learning properly adjusts the feature space to be Euclidean.

There are two types of metric learning, supervised and unsupervised. The unsupervised metric learning method transforms a vector x by a matrix M such that the elements in Mx are independent and normalized. Principal Component Analysis (PCA) and Multidimensional Scaling (MDS) both fall into this category. The main difference between the two is that PCA preserves the data variance while MDS preserves the inter-point distance [28]. However, unsupervised metric learning does not find the best presentation when the data is sampled from multiple subspace clusters. That is why we refine the feature space based on supervised metric learning.

Supervised metric learning requires that the data is labeled with pairwise constraints of either "similar" or "dissimilar." The simplest supervised metric learning method is linear discriminative analysis (LDA), which uses class labels to maximally separate two classes of data [12]. A more sophisticated method is devised by Xing et al. [26] for Mahalanobis distance metric [14]. It finds a new metric M, such that the new distance measure $d(x,y) = \sqrt{(x-y)^T M(x-y)}$ minimizes the distance of similar samples, and preserves the distances of dissimilar samples. For multiple data clusters (especially in the context of audio sample browsing) we extend this learning scheme to group-based constraints where we maximize the between-group distance, while preserving the upper-bound of the in-group distance.

2.3 Set Matching

Picking the best locations for audio samples can be formulated as a set matching problem, where the first set corresponds to the audio samples and the second to the 2D locations. Given two sets for which intra-set distances are known (but not inter-set distances), the problem of finding a permutation that best matches the two sets can be formulated as a Quadratic Assignment Problem (QAP) [11]. Since QAP is NP-Hard, an exact solution is generally out of reach and approximations or relaxations should be pursued. Such relaxations include kernelized sorting (KS) [18] and convex kernelized sorting (CKS) [5], both of which we use as part of our algorithm. Other options include *KS-NOCCO* and *LSOM* [27]. We choose KS due to its simplicity and speed, and find the performance to be adequate. We also implement CKS, which generally gives improved quality but at the cost of noticeably slower performance for large arrangements. (Future work might quantize the just-noticeable-differences and/or quantitatively compare different sorting approaches.) It is important to note that any of these methods can be used interchangeably in our algorithm, as we do not rely on any specific implementation details.

3. ALGORITHM

Our algorithm is formed as a two-part process, in which we aim to (1) automatically learn the correct relationships between sound samples based on user preferences and (2) automatically place the sound samples in a predefined arrangement that respects the previously learned relationships. We achieve these two tasks by using *metric learning* and *kernelized sorting*, respectively. In this section we describe these techniques and how we build on them in AudioQuilt.

3.1 Learning Correct Relationships

Metric learning allows us to incorporate user-supplied groupings of data. In the standard formulation, we suppose we observe a set of points $\{x_i\}_{1:m} \subseteq \mathbb{R}^n$ and are given the grouping information $\{y_i\}_{1:m} \subseteq \{1,2,...,K\}^n$ where y_i is the group label of point x_i. The supervised Mahalanobis metric learning problem is to learn the new distance metric $\|x-y\|_A \equiv d_A(x,y) = \sqrt{(x-y)^T A(x-y)}$, such that the pairwise distances of the points in the same group are kept small, while the distances of points between different groups are maximized. Note that the matrix A needs to be semi-definite such that the triangle property is satisfied for the new metric. Because $\sqrt{(x-y)^T A(x-y)} = \sqrt{(x-y)^T B^T B(x-y)}$ where $B = A^{1/2}$, this method essentially finds a new space Bx for vector x. According to Xing et al. [26], a plausible formulation for multiple group metric learning can be:

$$\max_A \sum_{(i,j)|y_i \neq y_j} \|x_i - x_j\|_A \qquad (1)$$

$$s.t. \sum_{(i,j)|y_i = y_j = k} \|x_i - x_j\|_A^2 \leq 1 \quad \text{for } k \in [K],\ A \succeq 0$$

Note that the objective is the sum of distances, and not the sum of squared distances, so semi-definite programming cannot be utilized. We do not want to change the objective to the sum of squared distances, because it leads to a rank-one solution of A, where all the data points are aligned onto a line [26]. But because of the convexity of this formula, we can still obtain the global maximum using gradient descent and iterative projection. However, the optimization is slow, which is unsuitable for a real-time application. To boost the speed and avoid low-rank solution, we propose the following optimization scheme for the metric learning:

$$\max_A \sum_{(i,j)|y_i \neq y_j} \|x_i - x_j\|_A^2 - \lambda \sum_{(i,j)|y_i = y_j} \|x_i - x_j\|_A^2 \qquad (2)$$

$$s.t. \quad \|x_i - x_j\|_A^2 \leq 1 \quad (i,j) \in \{(i,j)|y_i = y_j\},\ A \succeq 0$$

where the term λ is the weight of in-group distance penalty. Increasing this term yields more concentrated groups, which are closer to one another. In this work, we set this value to 1, meaning both in-group concentration and between-group divergence are important.

The new formula can be optimized using semi-definite programming and thus much faster to compute compared to the previous method.

3.2 Location Assignment

The inputs to the location assignment part of our algorithm are pairwise distances between n sound samples (i.e. d_{ij} is the distance between sound sample i and sound sample j, $d_{ii} = 0$) and target locations. Target locations $\mathcal{P}$ is an arbitrary set of 2D (or 3D) points in Euclidean space, such that $|\mathcal{P}| = n$. The goal is to find a bijection between the sound samples and the locations, such that pairwise distances are preserved as much as possible. Intuitively, we would like for similar sound samples to be placed close together and for dissimilar samples to be far apart. We achieve this goal by using two previously established algorithms: kernelized sorting [18] and convex kernelized sorting [5].

kernelized sorting. While we aim to solve a specific problem of matching sound samples to target locations, kernelized sorting is a general method to match any two equally sized sets, given an intra-set similarity measure (but not inter-set). More formally, we are given two sets $X = \{x_i\}_{i=1}^n$ and $Y = \{y_i\}_{i=1}^n$, which may belong to two different domains $\mathcal{X}$ and $\mathcal{Y}$. We are also given two kernel functions $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ and $l : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ which indicate the similarity between objects of $\mathcal{X}$ and $\mathcal{Y}$ respectively (i.e. a large value of $k(x_i, x_j)$ indicates that x_i and x_j are similar). We can represent the kernel values of our data in matrix notation as K and L, where $K_{ij} = k(x_i, x_j)$ and $L_{ij} = l(y_i, y_j)$ (we borrow the same notation as in KS [18]). Let us mark $\bar{K}$ and $\bar{L}$ as the centralized versions of K and L. The goal of KS is to find a permutation matrix Π such that $\bar{K}$ and $\Pi^T L \Pi$ are similar. The similarity criteria used is the Hilbert-Schmidt Independence Criterion [21]. An iterative approach is used to maximize an approximation of that criteria. We refer the reader to [18] for more details.

In our case, X are sound samples and Y are 2D locations as specified by the user. We compute similarity between sound samples by the Euclidean distance of the adjusted feature space learned from supervised metric learning. The similarity between two locations is the reciprocal of the Euclidean distance between them. KS allows us to interactively assign sound samples to user-specified locations.

Convex Kernelized Sorting Over the years several improvements to kernelized sorting have be proposed. One such method is convex kernelized sorting (CKS) [5]. The objective function is changed so that the problem becomes a convex minimization problem, as shown in Equation 3. We use Π, $\bar{K}$ and $\bar{L}$ as before, and $\|\cdot\|_F$ denotes the Frobenius norm.

$$\min_{\Pi} \|\bar{K} \cdot \Pi^T - (\bar{L} \cdot \Pi)^T\|_F^2 \tag{3}$$

The authors of CKS [5] show that the new method achieves better results in tasks such as image matching. The improved performance comes at a cost of a slower algorithm, due to a non-linear optimization step. We have incorporated CKS into our system and can dynamically pick between KS and CKS, achieving a performance-vs-runtime balance.

It is important to note that while we had good reasons to choose these specific algorithms (as explained above), they can be easily swapped in and out. Any other algorithm that can create a bijection from a distance measure is suitable for our needs, and the rest of the pipeline does not depend on the specific implementation.

4. APPLICATIONS

In this section we present 3 interfaces that we prototyped using our system. We demonstrate applications for sound sample navigation, as well as audio synthesis.

4.1 Snare-Drum Navigator

Much like fabric for the clothes designer and wood for the carpenter, sound samples are essential building material for the modern day music composer. It is not unusual for an artist to collect hundreds and thousands of sound snippets, ranging diverse categories. All those samples are, at best, arranged by category (e.g. electric guitar, tires screeching, female vocals) and at worst just dumped into a single folder and sorted alphabetically according to file name. Despite a number of viable sample exploration systems [8, 25, 2, 19] developed by researchers in past years, it appears that commercial composition tools are only just now beginning to implement a "more like this" feature for sound samples [1], and we feel the interface can go even further.

Our snare drum navigator is simple to describe: the user is presented with a 2D grid of rectangles, each with a slightly different color. Proximity of rectangles implies similar sound qualities. The color can either emphasize similarity, or be used as a channel to convey an extra layer of information. In this application we applied two coloring scheme. The first one is generated by Isomap [22] which projects the original features (24-D) to 2D and use them as the first two channels in LAB color space (the thrid channel is fixed in this case). The second scheme is to apply k-means to find clusters, then use the 3 principal components of the features to create an initial coloring based LAB space, and finally move the color of each sample towards the center of its assigned cluster. The corresponding waveform is displayed inside each rectangle. Following feedback from early users, we also show the history of the exploration process by fading out visited cells and fading them back in over time.

The feature vector for this application is constructed as follows. First the samples are normalized; the onset time is detected and aligned via a threshold of -30dB. Next we use MIRtoolbox [13] to obtain the envelope and the log attack time; we compute the temporal centroid and two 11-coefficient MFCCs of window size 2048 at the onset and

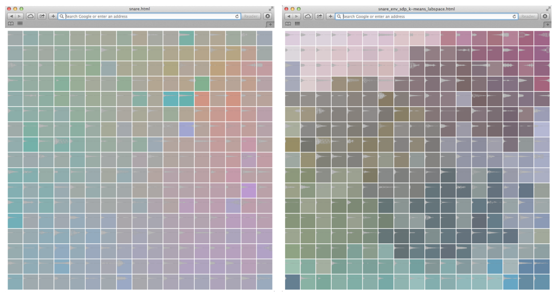


Figure 2: Snare drum navigator. The user is presented with a grid of colored rectangles, each corresponds to a sound sample. Hovering over samples produces sound; similar sounds are placed in proximity. The feature vector consists of MFCC descriptors, log attack time and temporal centroids. We support several coloring schemes, either according to the features used or according to other information we wish to convey. Here we show Isomap coloration (left) and k-means based coloring (right).

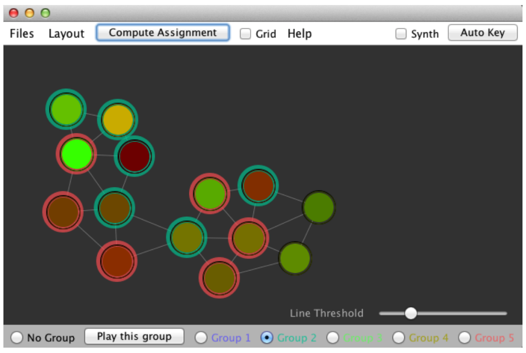


Figure 3: Our interface for metric adjustment. Each circle corresponds to a sound sample. The user can hover over samples to play them. Circles are color coded by the user according to their group by clicking (groups are shown at the bottom). Pressing "Compute Assignment" triggers a calculation of a new metric such that the user supplied labeling is respected. This GUI allows for interactive manipulation of the distance metric according to the user's needs, which can be applied to create new sound layouts (Figure 2).

after the attack (sustain). We concatenate these three attributes (24 dimensions in total) as the feature descriptor.

Figure 2 shows our snare drum navigator. We first learned the metric space using a training set of snares, manually labeling 176 samples out of a total of 839 in the training set into 12 categories. We note that this is a one time process for each data type (i.e. we can now sort snares without repeating this process). The total labeling time was about 30 minutes. We then applied the metric to the displayed test set. The result is an arrangement with different types of snares clustered in different parts of the interface. In the upper left are aggressive, punchy snares, while in the lower right are thin, snappy snares. In between there is a gradient of snare types ranging (moving top to bottom) from hip-hop vinyl snares, to woody acoustic snares, to thin and light acoustic snares. We refer the user to the accompanying video for a more audible experience. We also show the same sound layout, with a different color scheme.

We also allow the user to interactively adjust the learned metric by assigning group labels to examplars. Figure 3 shows the interface. Starting from a layout, a user can listen to the samples and indicate the proper group by changing the colors of the nodes. Then he/she presses "compute" and the system uses the grouping information to learn a new metric that enlarges the difference between different colors, and then apply it to the distance matrix for kernelized sorting to obtain a new layout.

4.2 Synth Explorer

We can also use our method to explore audio synthesizer parameters. The task of navigating synthesizer parameters can be challenging due to the diverse sounds and the nonlinear ways in which parameters interact. Furthermore, many novice users do not understand all the different buttons and knobs that are present on a synth. We aim to alleviate the problem by supplying an intuitive interface for exploring synth parameters and the resulting sounds.

In this system we use a simple synthesizer with 4 parameters: Pitch, FM amount, FM frequency, and Ring modulation frequency. The features used are MFCCs with a 8000 sample window and the log attack time.

The interface is shown in figure 4. Each node represents a sound created by a given synth parameter setting. To begin,

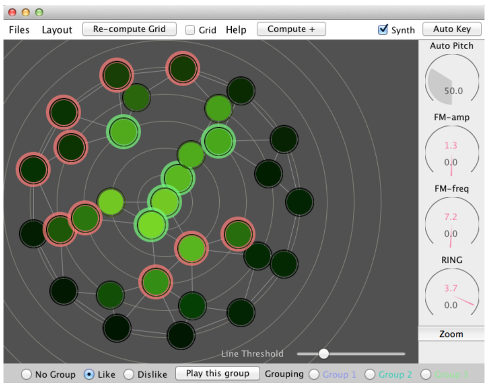


Figure 4: Synth sound explorer. Synthesizer parameters are sampled to create sound snippets and arranged by sound similarity. The user hovers over the circles to listen to the snippets, and can indicate likes/dislikes. The space is re-sampled and re-sorted interactively according to the user's selection. We refer the reader to the accompanying video for a more vivid demonstration.

we sample uniformly at random, from the parameter space of the synthesizer. For each parameter setting we record a 1-second sound clip, and use kernelized sorting to map it onto a node, so that timbrally similar sounds are near one another. Mousing over a node plays the sound. The layout of nodes is freely configurable. If the nodes are moved (the user can drag the circles), the sound-to-node mapping is recomputed to maintain the timbral relationships. The user then indicates which sounds they like, and some that they don't like. These two labels are used as classes for metric learning to recompute the space. Once the space has been recomputed, the system samples around the parameter space of each "liked" setting. This creates a hierarchical method for exploring the space. The new samples are arranged in the same manner described earlier. Note that sounds with very different parameters, but similar timbres, will be arranged next to one another, enabling the user to search by audio, not parameter. When they want to learn the parameters, they click the node, and the parameters of the synth are shown on the right side of the screen. They can also search by adjusting the parameters directly.

5. USER STUDIES

We evaluated our system using two methods – one in-person and the other online. The in-person study aims to get qualitative feedback from musicians on the usability of the system, while the online study was geared towards quantitative results, to help us understand which aspects of the system affected search time and accuracy. Both studies used the snare drum navigator shown in Figure 2.

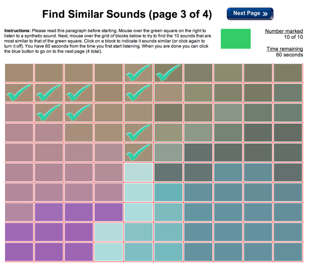
5.1 In-Person Qualitative Study

Our in-person study involved 5 musicians. Each of them were familiar with composition using a DAW, and the process of sample selection. They ranged from 20-33 years in age and consisted of 2 undergraduate students, 2 graduate students and one faculty member.

We adopted a contextual observation style approach, giving the user only basic instructions and offering information only when asked. Before the user study, we pre-trained the system using metric learning. The users were instructed to use the system as if they were browsing for snare drums for a project. As they browsed they narrated their experience. After they had located a few snares that they liked, we conducted a semi-structured interview.

The most common feedback we received is that the system was very "fast". People liked the ability to audition a number of samples very rapidly. All of the users spontaneously identified clusters of similar snares in the interface. Some comments included "these here have scooped out mids" and "here are a bunch of hip-hop vinyl samples". However, they were sometimes surprised by certain snares that seemed not to fit their neighbors at all, "this snare should be over there", was a common comment. These unexpected snares appeared to diminish the user's confidence in their understanding of the system.

5.2 Online Quantitative Study



Find Similar Sounds (page 3 of 4)

Instructions: Please read this paragraph before starting. Mouse over the green square on the right to listen to a synthetic sound. Next, mouse over the grid of blocks below to try to find the 10 sounds that are most similar to that of the green square. Click on a block to indicate it sounds similar (or click again to turn it off). You have 60 seconds from the time you first start listening. When you are done you can click the blue button to go on to the next page (4 total).

Number marked
10 of 10

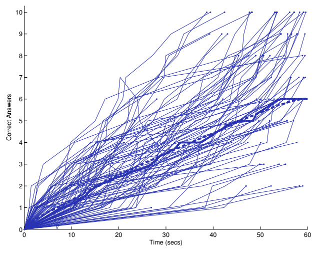
Time remaining
60 seconds

The goal of our second study is to evaluate our system quantitatively by testing a person's ability to find particular sounds among a collection. In a typical composition situation an artist might not be looking for a specific sound among their collection, just one that is "close enough." For each test, we generated 10 "close enough" sounds, which are similar to an exemplar sound, and 90 samples that are dissimilar. The goal of the task is to find as many of the 10 "close" samples as possible within 60 seconds. To create samples, we first randomly generate 2000 synthesized samples; then we extract the audio features and compute k-means to obtain 10 clusters; finally, we pick sample around the cluster centers.

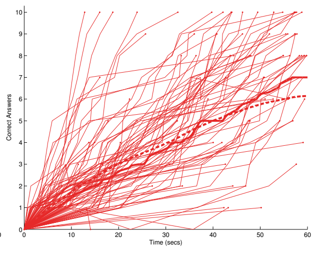
In the study, subjects were presented with our grid interface as well as the reference exemplar sound (shown inset). Before the test began, each subject read a short explanation of the task. Then they were allowed to listen to the exemplar as many times as they liked. Once they began browsing the grid of samples, they were given 60 seconds to mark up to 10 similar sounds. To mark a sound as similar, they clicked on the rectangle and a check mark appeared; an optional subsequent click would remove the mark. If subjects were satisfied with their selection, they could end the task early.

In our test, the independent variables were: (1) grid coloring enabled or disabled, crossed with: (2) Kernelized Sorting versus random arrangement. This yielded 4 different conditions. Each condition was paired with a different sample set (selected randomly from among 8 total sets) and every subject was presented with all four conditions in random order. We recruited 100 subjects using the Amazon Mechanical Turk (a microtask marketplace shown to be effective for online user studies [10]). Each subject spent approximately 5 minutes completing one "human intelligence task" (HIT). Each subject was randomly assigned a set-condition pair, and did exactly one HIT. In total we collected data from 100 HITs, and thus our experiments include a total of 400 trials (100 in each condition).

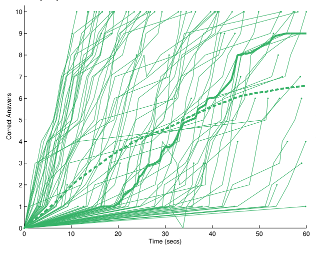
The 400 trials are plotted in Figure 5 as the correct number of marked samples vs. time. Notice there is a distinct shape to the spatially sorted trials: the subject tends to make little progress for a while, then the number of correct answers skyrockets. We hypothesize that this is the moment when the proper cluster is found. In contrast, the unsorted trials show a slower progress towards the maximum, with no great leaps. Overall, the bird's eye view is that the best performing condition (quickest to achieve high numbers of correct answers) is the use of sorted arrangements together with color.



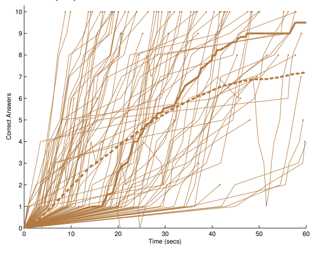
(a) not sorted, no color



(b) not sorted, color

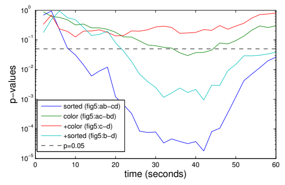


(c) sorted, no color

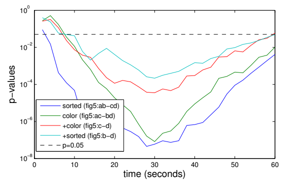


(d) sorted, color

Figure 5: Individual and aggregate results from the online user study. Each thin lines is the performance of a single user in the 60 second trial, Y-axis is number of correct answers. Wide dashed line indicates mean over all 100 trials; wide solid indicates median. Steeper is better, and in aggregate the sorted arrangement with color (d) is best.



(a) 100 users, 100 HITs



(b) 45 Users, 100 HITs

Figure 6: P-value plots against time. Four ANOVA tests are shown; the response variable is the number of correct answers at time *t*. Tests are "sorted": sorted versus unsorted; "color": color versus no color; "+color": sorted with versus without color; "+sorted": colored with versus without sorting. Legend labels also refer to Figure 5. Left: 100 unique users. Right: 45 users, each allowed to repeat the HIT up to 5 times (100 HITs in total).

We also evaluate these trajectories numerically. Our dependent variable is the number of correct answers at time t. We constructed 4 tests and evaluated their p-values every 2 seconds. The results can be seen in Figure 6(a). The data shows that spatial sorting has significant effects early on (p-value $< 10^{-3}$ at around 20 seconds). Spatial sorting also has an effect when only considering colored data (p-value $\approx 10^{-3}$ in the 30–40 second time-frame). The U-shape of the curves hints that given enough time, all conditions will converge (since the user can inspect all samples when time is abundant). Color is mostly insignificant. We hypothesize that our users did not have a chance to learn the meaning of color. Thus, in another experiment (Figure 6(b)) we allow the same user to perform several HITs (45 users, 100 HITs), in which case color becomes significant. In both versions of the experiment, arrangement was the most important factor in user success.

We logged all the events in the study, so we were able to see exactly how a subject interacted with the system. We found that people used different strategies depending on the interface. These different strategies give us insight into why users find the system "fast" to use. When using the uncolored, unsorted task first, the user was likely to adopt a linear approach, auditioning every square row-by-row. With the colored, sorted interface, they would begin with the linear approach, but as they became aware that the samples were spatially sorted, they changed their strategy, and moved from cluster to cluster, rapidly exploring homogeneously colored regions before moving onto the next. We expect even better performance on a real world system which is familiar to the user.

6. CONCLUSIONS AND FUTURE WORK

In this work we have shown AudioQuilt, a way to harness metric learning and kernelized sorting in order to provide superior layouts of sound samples. We demonstrated how these layouts can be used for sample navigation and as an interface for musical instrument creation. We have shown several applications using snare-drum samples and synth sounds. A promising direction for future work is to experiment with other families of sound samples, such as different instruments, vocals and movie effects. Another exciting direction for future work will allow metric learning with less or no user supplied input (using, for example, inductive transfer to propagate information between sound families). We would also like to investigate, in more depth, the effect that different coloring schemes have on search quality. We feel that sound sample exploration is at a stage where much more exciting work can be done.

7. ACKNOWLEDGMENTS

We thank our study participants for the informative discussion and our anonymous reviewers for their helpful comments and suggestions. This work was supported in part by generous gifts from Adobe, Google, and the Project X Fund at Princeton.

8. REFERENCES

[1] iZotope BreakTweaker, Jan. 2014. http://www.izotope.com/products/audio/breaktweaker/.

[2] V. Akkermans, F. Font, J. Funollet, B. de Jong, G. Roma, S. Togias, and X. Serra. Freesound 2: An improved platform for sharing audio clips. In *Late-breaking demo abstract of the Int. Soc. for Music Information Retrieval Conf*, 2011.

[3] G. Coleman. Mused: Navigating the personal sample library. *Proc. ICMC, Copenhagen, Denmark*, 2007.

[4] P. Cosi, G. D. Poli, and G. Lauzzana. Auditory modelling and self-organizing neural networks for timbre classification. *Journal of New Music Research*, 23:71–98, 1994.

[5] N. Djuric, M. Grbovic, and S. Vucetic. Convex Kernelized Sorting. pages 893–899, 2011.

[6] A. Eigenfeldt and P. Pasquier. Realtime Sample Selection Based Upon Timbral Similarity: A Comparison Between Two Methods. 2004.

[7] J. M. Grey. Multidimensional perceptual scaling of musical timbres. *The Journal of the Acoustical Society of America*, 61:1270, 1977.

[8] S. Heise, M. Hlatky, and J. Loviscach. Soundtorch: Quick browsing in large audio collections. 2008.

[9] P. Herrera-Boyer, G. Peeters, and S. Dubnov. Automatic classification of musical instrument sounds. *Journal of New Music Research*, 32(1):3–21, 2003.

[10] A. Kittur, E. H. Chi, and B. Suh. Crowdsourcing user studies with mechanical turk. In *Proceedings of the SIGCHI conference on human factors in computing systems*, pages 453–456. ACM, 2008.

[11] T. C. Koopmans and M. Beckmann. Assignment problems and the location of economic activities. *Econometrica*, 25(1):pp. 53–76, 1957.

[12] B. Kulis. Metric Learning: A Survey. *Foundations and Trends in Machine Learning*, 5:287–364, 2012.

[13] O. Lartillot, P. Toiviainen, and T. Eerola. A matlab toolbox for music information retrieval. In *Data analysis, machine learning and applications*, pages 261–268. Springer, 2008.

[14] P. C. Mahalanobis. On the generalized distance in statistics. *Proceedings of the National Institute of Sciences of India*, 2:49–55, 1936.

[15] E. Pampalk, P. Herrera, and M. Goto. Computational Models of Similarity for Drum Samples. *IEEE Transactions on Audio, Speech & Language Processing*, 16:408–423, 2008.

[16] E. Pampalk, P. Hlavac, and P. Herrera. Hierarchical organization and visualization of drum sample libraries. In *International Conference on Digital Audio Effects*, 2004.

[17] G. Peeters, S. McAdams, and P. Herrera. Instrument sound description in the context of mpeg-7. In *Proceedings of the 2000 International Computer Music Conference*, pages 166–169. Citeseer, 2000.

[18] N. Quadrianto, A. J. Smola, L. Song, and T. Tuytelaars. Kernelized sorting. *IEEE transactions on pattern analysis and machine intelligence*, 32(10):1809–21, Oct. 2010.

[19] S. V. Rice and S. M. Bailey. Searching for sounds: A demonstration of findsounds.com and findsounds palette. In *Proc. of the International Computer Music Conference*, pages 215–218, 2004.

[20] D. Schwarz, R. Cahen, and S. Britton. Principles and applications of interactive corpus-based concatenative synthesis. *Journées d'Informatique Musicale, GMEA, Albi, France*, 2008.

[21] A. Smola, A. Gretton, L. Song, and B. Schölkopf. A Hilbert space embedding for distributions. *Algorithmic Learning Theory*, pages 1–20, 2007.

[22] J. B. Tenenbaum, V. de Silva, and J. C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, 2000.

[23] P. Toiviainen. Optimizing auditory images and distance metrics for self-organizing timbre maps. *Journal of New Music Research*, 25:1–30, 1996.

[24] G. Tzanetakis, M. S. Benning, S. R. Ness, D. Minifie, and N. Livingston. Assistive music browsing using self-organizing maps. In *International Conference on Pervasive Technologies Related to Assistive Environments*, pages 1–7, 2009.

[25] E. Wold, T. Blum, D. Keislar, and J. Wheaten. Content-based classification, search, and retrieval of audio. *MultiMedia, IEEE*, 3(3):27–36, 1996.

[26] E. P. Xing, A. Y. Ng, M. I. Jordan, and S. J. Russell. Distance Metric Learning with Application to Clustering with Side-Information. In *Neural Information Processing Systems*, pages 505–512, 2002.

[27] M. Yamada and M. Sugiyama. Cross-Domain Object Matching with Model Selection. *arXiv preprint arXiv:1012.1416*, 15:807–815, 2010.

[28] L. Yang. Distance metric learning: A comprehensive survey, 2006.