



Research on Music/Sound Databases at CREATE

Large-scale storage of sound and music has only become possible in the last decade. With this, and the new possibility for wide-area distribution of multimedia over the Internet, there arose a new requirement for flexible and powerful databases for musical and audio data. Since 1996, our work at CREATE has focused on database frameworks for multimedia applications, and on analysis and feature extraction techniques for music and sound databases. This white paper describes our results and presents several of our plans for future applications.

Background

Most prior work in sound or music databases has addressed a single kind of data (e.g., MIDI scores or sampled sound effects), and has pre-defined the types of queries that are to be supported (e.g., queries on fixed sound properties or musical features). Earlier systems also tended to address the needs of music librarians and musicologists, rather than composers, performers, and producers.

In our work at CREATE since 1996, we have built a suite of sound and music analysis tools that is integrated with an object-oriented persistency mechanism and a rapid application development environment. The goal is to provide a suite of analysis, storage, and query construction tools that scale to support large data volumes, complex queries, and on-the-fly analysis in several flavors. Our projects have produced several database analysis and interaction tools that can be used together or separately; they are called *Paleo*, *NOLib*, *P_Const*, and *FASTLab*. We will describe each of them below, and present several scenarios for their incorporation into innovative applications.

The Paleo Database Project

The *Paleo* database project at CREATE aims to develop and deploy a large-scale integrated sound and music database that supports several kinds of content and analysis data and several domains of queries. The basic components of the *Paleo* system are: (1) a scalable general-purpose object database system, (2) a comprehensive suite of sound/music analysis (feature extraction) tools, (3) a

distributed interface to the database, and (4) prototype end-user applications.

The *Paleo* system is based on a rich set of signal and event analysis programs for feature extraction from sound and music data. The premise is that, in order to support several kinds of queries, we need to extract a wide range of different kinds of features from the data as it is loaded into the database, and possibly to analyze still more in response to queries. The results of these analyses will be very long “feature vectors” (or multi-level indices) that describe the contents of the database. To be useful for a wide range of applications, the *Paleo* system must allow several different kinds of queries, i.e., it needs to manage large and changing feature vectors.

As data in the database is used, the feature vectors can be simplified. This might mean discarding spectral analysis data for speech sounds, or metrical grouping trees for unmetred music. This is what sets *Paleo* apart from most other media database projects—the use of complex and dynamic feature vectors and indices.

To be useful for a wide range of applications, the *Paleo* framework must support musical and non-musical sound, and must allow several different kinds of queries. The categories of data that are supported include:

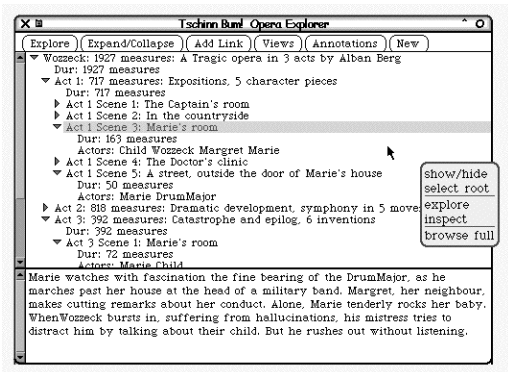
- musical sound (single instrumental notes, phrases, and entire pieces)
- non-musical sound (voice, sound effects)
- speech (single-voice utterances)
- musical scores (e.g., MIDI data or notated score information)
- expressive data (timing, amplitude, tim-

bre) from performance (“interpretation”)

Among the possible query types, we need to support the following:

- query by annotation keyword (bibliographical, analytical)
- query by sound similarity (many kinds of matching)
- query by musical content (phrases, harmony, formal structure)
- query by spoken phrase (assumes some speech recognition)
- query by performance technique (assumes capture of expressive data)
- query by expression (assumes keyword mapping of performance data)

The Paleo project progressed from the development of the basic analysis software (most of which has already been developed at CREATE in the NOLib project), to the population of test databases (already started as part of the Siren project), and then to the programming of the search engine to support the various kinds of queries required by the various Paleo applications.



Database browser exploring the structure of the opera *Wozzeck*

The GUI example in the figure above shows a simple database browser where the annotated structure of an opera can be used to browse the score or the actual recording of the various scenes.

Signal Analysis with NOLib

NOLib is a suite of signal analysis routines written in the MatLab programming language. These functions are called by scripts, which are started by a network-based “analysis server.” We use the public-domain “octave” implementation of MatLab running on UNIX or Linux servers.

Rather than list the functions NOLib implements (see its reference documentation), we present an annotated example of a NOLib analysis script that reads a directory of sound files and derives several relevant features. We used this script for 100 recorded guitar notes to build a database object set for query based on performance technique. (Note that MatLab functions routinely return more than one result, as implied by the syntax

```
[a b c] = fcn(x, y, z))
```

```
# Set the sampling rate & window dimension
Fs = 48000;
windan = 2048;
# Path to the files
fpath = '/snd/stp/samples/';
# Name of the group of files plus their
# file name extension
fnames = [...list_of_file_names...];
extens = '.aiff';
# Number of files
[filnum,void] = size(fnames);
# Number of analysis coefficients
numcoef = 20;
# Initialize the coefficients table
lfc = zeros(20,filnum);
```

```
# Loop for the analysis
for n = 1:filnum,
    # Load the file
    filename = [fpath,deblank(fnames(n,:)),extens];
    [head x] = ldaiff(filename);
    # Find portion of file after the attack
    [inpos endpos] = findport(x,Fs);
    # Check size of “steady state”
    # up to 0.1sec max
    if (endpos-inpos > Fs * 0.1)
        endpos = inpos + Fs * 0.1;
    endif
    # Analyze a frame to get the linear
    # frequency cepstral coefficients
    tmp_lfc = linceps_f(x(inpos:endpos), Fs,
        numcoef, windan, windan/2);
    # Compute the mean value of the
    # cepstral coefficients
    lfc(1:numcoef,n) = mean(tmp_lfc);
    # End of the file loop
endfor
# Perform principal component analysis on the
# mean value vector
[projection eigvec weig] = pca(lfc);

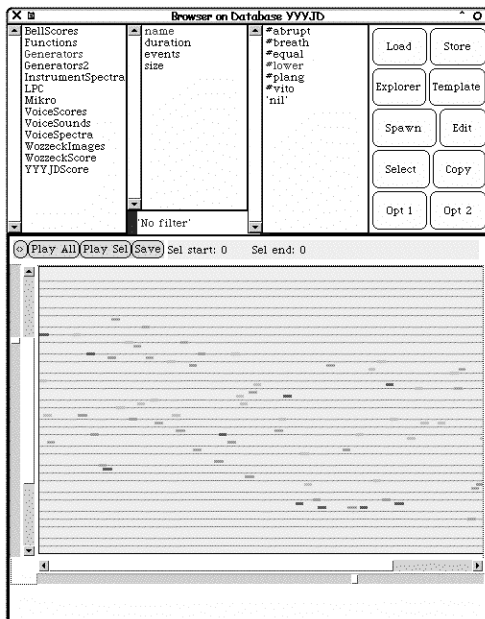
# At this point, the projection vector “proj” can be
# used to classify the plucking technique for the
# guitar sample, and the spectral variance can be
# derived from the linear transformation matrix
# “eigvec.”
```

Other NOLib scripts have been used for audio file segmentation, speaker identification, flute performance analysis, and sound effect feature extraction.

Constraints for Music Analysis

A large amount of digitalized music is available as MIDI files, for instance from the many MIDI archives on the Internet. The MIDI format however, provides only low-level musical information; it is a performance-oriented rather than an analysis-oriented representation. Thus, we need to analyze MIDI files to derive additional musical features, such as pitch-classes, voice leading, keys, and harmonies; this is the domain of the Paleo constraint-based analysis package *P_Const*.

One of the main applications of *P_Const* is to search MIDI files based on musical content, and to compare multiple MIDI files of the same piece and tell which are “correct” and possibly which are “dead-pan” performances (i.e., uninterpreted scores).



Database browser showing the navigation options and a MIDI-derived score

The different tasks of analysis—enharmonic, melodic, tonal, and harmonic analysis—are not independent. For instance, the enharmonic analysis depends on tonal analysis, and conversely, the computation of local keys is based on the frequency of the different pitch-classes. Therefore, we need a global

strategy in which the different tasks are performed simultaneously.

In our context, we often need a partial analysis because many queries only involve a few specific elements or incomplete information. Consider the following queries: “How many sonatas by Scarlatti end with a perfect cadenza?” or “Are there more minor than major chords in the preludes of Bach’s Well-Tempered Clavichord?” In such cases, it is useless to perform a complete harmonic analysis of the 555 sonatas by Scarlatti, or of the 48 preludes of the *WTC*. This speaks for a scheme allowing partial and incomplete analysis.

This situation led us to an approach based on *constraint satisfaction*, instead of using specific algorithms for the different tasks on analysis. Another advantage of constraint resolution is that it can be partial and incomplete. More precisely, the query “How many sonatas by Scarlatti end with a perfect cadence?” will only require the computation of elements related to the last two chords of each sonata. Finally, constraint resolution is a global process, in which the different elements are progressively computed, thus, interdependent tasks are interlaced in the resolution.

A *constraint satisfaction problem* or *CSP* consists of a set of *variables* (each one associated with a set of possible values, its *domain*), representing the unknown values of the problem, and a set of *constraints*, expressing relationships between them. Solving a CSP consists in instantiating each variable with a value in its domain so that the constraints are satisfied.

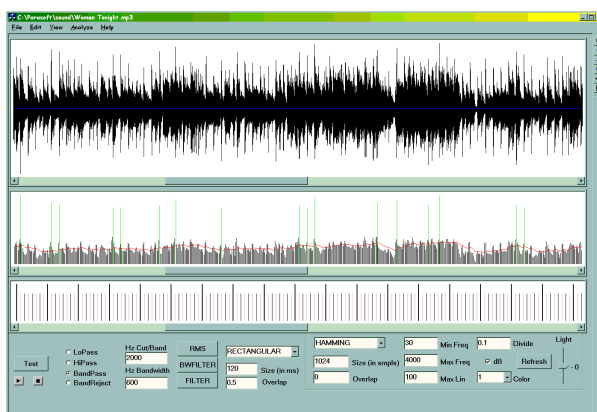
Our approach to analyzing a MIDI file consists in the following steps. First, we quantify the MIDI file in order to get rid of slight tempo fluctuations, and we segment it into a series of positions. Then, we define a CSP, whose variables represent the different elements of analysis: notes (one for each MIDI note-event), chords (at each position), keys (at each position), and melodies, and whose constraints represent the relationships holding between them. The set of constraints depends on the style and the form of the piece. Then we solve the CSP using standard CSP resolution. We use the BackTalk constraint solver to state and solve the problem. The use of constraints allows us to provide efficient partial

solutions to make musical questions that are very difficult to solve completely.

FASTLab: Large-Scale Analysis

The FASTLab system is an analysis tool for use in music and sound databases, preference-matching applications, and other cases where a multidimensional characterization of a musical selection is needed. It uses a number of different analysis techniques to extract features from musical selections. The present version is oriented towards the analysis of popular music songs stored as MP3 files.

The goal of FASTLab is to derive a number of high-level musical properties from recorded selections so that these properties can be used to match compare and contrast pieces of music, either in a database query, a user-specific music selection system, or related applications.



FASTLab GUI with a time signal and the derived beat and tempo analysis

FASTLab is a flexible and open system that uses a range of sound analysis techniques in the time and frequency domains. It also uses several second-level heuristic programming techniques such as rule-based matching, fuzzy logic, and classification and regression trees (CART).

The framework is extensible both in terms of the low-level analysis and feature extraction techniques it uses, and in terms of the second-level pattern recognition, matching, and classification tools that are available. Lastly, the output format is also changeable, the default action being to dump all analysis data to XML files for loading into a database.

Applications

From the historical perspective presented above, one sees that our efforts have stressed analysis and feature extraction, as well as interface and application construction. The application areas that we have targeted include music education, interactive music selection, archiving of speech, sound-effect databases, and music scholarship.

Plans for Future Work

There are a number of topics that are still difficult design issues in the wide range of potential applications for music/sound database technology. Our specific R&D interest include the following topics:

- Flexible frameworks for music/sound indexing
- Databases for classification and indexing;
- Database-oriented GUIs for music/sound applications;
- Better automatic feature extraction from musical performance and spoken text;
- Efficient re-indexing of media databases;
- Versioning and persistency in “sketch” databases;
- Performance of searching in complex hierarchies of media annotation;
- Applications support for remote media database access;
- User-configurable feature extraction using scripting and query-by-example.

Contact Information

Stephen Travis Pope
Tel: (805) 967-2621
Email: stp@create.ucsb.edu

Updated 2001/3/15