Music Information Retrieval Research and Its Context at the University of Waikato

David Bainbridge
Department of Computer Science, University of Waikato, Hamilton, New Zealand.
E-mail: d.bainbridge@cs.waikato.ac.nz

This article describes the digital music library work at the University of Waikato, New Zealand. At the heart of the project is a music information retrieval workbench for evaluating algorithms and performing experiments used in conjunction with four datasets of symbolic notation ranging from contemporary to classical pieces. The outcome of this experimentation is woven together with strands from our larger digital library project to form the Web-based music digital library MELDEX (short for melody index). An overview of the workbench software architecture is given along with a description of how this fits the larger digital library design, followed by several examples of MELDEX in use.

Introduction

Music information retrieval (MIR) research at the University of Waikato, New Zealand, is an integral part of a digital music library project, which in turn is part of the larger New Zealand Digital Library (http://www.nzdl.org) project (Witten, McNab, Boddie, & Bainbridge, 2000). The music work covers a broad range of topics—from content and query acquisition, through retrieval algorithms, to document presentation and software scalability—topics that mirror the key stages of the main digital library project.

At the core of the music group is a mixture of faculty and graduate students—all active musicians—with expertise that includes signal processing, information retrieval, software architecture, and human computer interaction. The precise boundary between this group and its parent digital library project is intentionally indistinct to encourage overlap and serendipity between researchers. To take one general example: given the context of the music information retrieval work as a digital music library, interface issues are an important consideration, and it has been a natural process for staff skilled in human computer interaction to become involved in the computer music work.

The structure of this article is as follows. First, we detail the software workbench that is at the heart of our research efforts. Designed with flexibility in mind, this software tool allows us to experiment with different algorithms, comparing and contrasting particular aspects: recall and precision, computational cost, memory use, and so on. Next, we describe the datasets we work with, which are based around symbolic notation reflecting our interest in this form of music information retrieval. We conclude by describing our Web-based digital music library service MELDEX (short for melody index), which takes advantage of both our experimentation and digital library components developed elsewhere in the larger project.

A Music Information Retrieval Workbench

Figure 1 gives an overview of the workbench, designed to integrate with the parent group’s broader digital library software architecture. Using an object-oriented paradigm, the software was originally written using a mixture of C++ and Perl; an updated version in Java is in progress. The workbench is divided into two phases: assimilation and runtime. Controlled by a configuration file, the assimilation phase is responsible for collating files together to form indexes and/or databases. The runtime phase is where experimentation is carried out, guided by user input. While assimilation is typically performed once for a given set of experiments, the runtime phase is executed many times to garner results.

Importing takes care of the multitude of different file formats associated with music data, converting input files into a canonical format that is both general and expressive enough for our needs, yet straightforward to parse. After considering a range of possible formats—and factoring in requirements from the larger digital library project—we selected the XML version of Guido (Hoos, Renz, & Gorg, 2001). For each format handled by the workbench, a new object is inherited from the Import base class, and its member functions overridden as appropriate. A mixture of hand-tailored code and, when available, existing open
source utilities are used to perform this. For example, the MIDI import module is based around the MIDI to Guido conversion utility (available at http://www.salieri.org/guido).

Filtering and expanding are optional steps. Conceptually, the former reduces the stream of musical data passing through it while the latter increases it. From an implementation standpoint, however, this distinction is insignificant and both categories inherit from the same base class that is set up to manipulate (add, remove, or modify) a stream of musical data. The two entities are represented separately in the workbench because it is useful to be able to differentiate between them during the configuration phase. An example of filtering is retaining only MIDI tracks that are monophonic, which is useful when the searching algorithm on trial only works on this type of notation. An example of expanding is generating a tune in different keys, which is useful for algorithms that only work in an absolute key.

A more interesting example of filtering is RepairMotif, which detects repeated fragments of tune. Central to the module is the “re-pair” algorithm (Larsson & Moffat, 1999) from which the filter takes its name. This is a rule-based character comparison algorithm that, if coded carefully, detects repeated substrings in linear time and space. To work in a music context, a preprocessing step within RepairMotif maps notes to characters, and a postprocessing step studies the grammar rules produced to pick the most frequent and/or longest substrings and then reverses the mapping for the chosen strings.

To perform searching using the workbench, there is a choice among three types of algorithms: state-based matching (Wu & Manber, 1992), dynamic programming (Mongeau & Sankoff, 1990), and text-based information retrieval (Witten, Moffat, & Bell, 1999b), each with optional arguments that modify their behaviors, for example, matching only at the start rather than at any position within a tune, using a pitch contour rather than exact intervals, and so forth. The output of a search is held as a result set from which statistics are extracted, graphs plotted, and tables generated. The particular operations performed by the workbench in the runtime phase are controlled by the user who can interact directly with the workbench through its command line syntax or develop scripts that batch process the desired experiment.

The group supports open source projects, and there are plans to release the workbench under the GNU public license. It is hoped that this will help foster a stronger sense of community within the music information retrieval field where other groups will add their retrieval algorithms to the workbench, thereby allowing a more comprehensive comparison of the relative strengths and weaknesses of the algorithms being developed.

Datasets

There are four main datasets used for experimentation. We call them Folksongs, Fakebook, Midimini, and Midimax. The first two are formed from symbolic data that have been either entered by hand or generated automatically by processing scanned sheet music using optical music recognition software. The other two, as reflected by their names, are derived from MIDI files and are, therefore, only pseudo-symbolic from the point of view of music notation (because this format captures note-on and note-off events, not true abstract notation) but interesting to work with nonetheless.

The Folksong dataset contains 9,354 monophonic tunes and is a selected combination of the Essen database (Schafrath, 1992) and a snapshot of the Digital Tradition (Greenhaus, 1996) taken in 1996. Some editing of the data was performed when the two sets were put together. In the case of duplication, repeated tunes were initially identified by title, then checked musically for co-incidence, and only one version retained for the merged set. Alternative tunes for the same song, however, were kept; for instance, there are two distinctly different versions of the mining song Clementine and likewise for Auld Lang Syne (the latter came as a surprise for the author, despite being raised in Scotland!). Because the initial workbench did not support triplets, songs with this metric were culled from the dataset.

A fake book is an informal collection of often-requested tunes typically collated by a band performing over several years. The sheet music used typically includes the basic tune (verse and chorus), the lyrics, and some guitar chords. The information is sufficient for a band to give a rendition of a requested song, sometimes “faking it” if they have never heard the piece before.

One such collection was scanned and processed by Cantor, a research-based optical music recognition system initially developed at the University of Canterbury (Bainbridge & Bell, 1996; Bainbridge, 1997) and also used in a research context at Waikato. A customized configuration was used that identified song title information in addition to the notation located on the staff. No attempt was made to recognize guitar chord information, although this would

FIG. 1. A workbench for symbolic music information retrieval.
have made an interesting project in its own right. Textual information such as title, composer, and lyrics, however, was entered by hand. The end result was 1,235 monophonic tunes richly augmented with textual metadata.

Based on a random sample of 20 images, the number of edits necessary to correct the score was nine edits per 100 notes to fix durational mistakes, and one edit per 100 notes for pitch. It is estimated that it would take 12–14 working days to correct the complete collection, although this has not been done. Given that many music-content search algorithms employ approximate matching techniques to allow for discrepancies between how a user remembers a tune fragment and how it is scored in the database, the uncorrected Fakebook dataset makes an interesting test case for how such algorithms perform on an error-prone repository.

Despite being sourced from the same format, the two MIDI datasets have strikingly different characteristics. Midi mini is a collection of 1,216 files from a single Web site where the maintainer had organized the tunes into a hierarchy. Broadly classified as pop, rock, and classical, file names encode the name of the tune and in the case of artists with several songs, a further subdirectory is used to encode this also.

In contrast, the Midi max dataset was gathered from the Web at large. A utility was written that spawned a query to a search engine searching for hyperlinks that ended in .mid. The located files were downloaded using file names that encoded their URLs. The data was then cleaned up, removing zero length files, identifying 404 errors (Web parlance for file not found), and detecting exact (binary) duplicates. The end result was 99,000 MIDI files.

Digital Library Software Architecture

Figure 2 shows the general software architecture devised for the main digital library project (Witten et al., 1999a). Because documents are represented as abstract entities, the design is flexible enough to handle diverse forms of media, as demonstrated by example collections built from text, images, video, and audio documents (Bainbridge et al., 2001). To realize a digital music library, the music information retrieval workbench was embedded into the general digital library design. This was straightforward as the digital library architecture foreshadowed the design of the workbench and, consequently, the workbench was developed with this embedding step in mind.

The workbench assimilation phase corresponds to the lower half of Figure 2 used to build a collection that is searchable and browsable. The term “import” in Figure 2 carries a broader meaning in the general design due to the diverse assortment of data types handled. Using a system of “plugins”—modules of code tailored to parse particular formats—source documents in potentially different formats are imported into the collection and transformed into a homogeneous XML format from which the necessary indexes and databases can be formed. The transformation process also includes the ability to augment the data. Examples of this include assigning bibliographical information to audio recordings and detecting hyperlinks that point outside of the collection (useful when your digital library collection is shipped out as a CD-ROM).

Because importing requirements can differ greatly from one collection to another, even if they use the same data format, a collection has a configuration file that lists the plugins needed and supplies the specific transformations required as options to these plugins. All these abilities tie in well with the music information retrieval workbench: its assimilation phase already incorporates the notion of a configuration file, and the filtering and expanding components drop into place as plugin options.

Less apparent between the diagrams, but equally well aligned in terms of integration, is the workbench’s runtime system, which maps to the collection server in the digital library design. Although not shown in Figure 2, inside the collection server component are Search and Result objects, among other things. These two objects perform the same function as their workbench counterparts, and through inheritance a new collection music server can be readily constituted that, by accessing the workbench, overrides only the parts specific to supporting music content. The other objects to the collection server remain unchanged, functioning as before providing generic services such as user authentication.

The role of the receptionist component in the digital library system is to provide an interface between the digital library’s content and a user wishing to access it. By decoupling user interface issues from content management, a flexible and versatile environment is formed. For example, one receptionist might be set up to be multilingual and interact through HTML pages, whereas another—still accessing the same basic content—might provide a graphic querying environment through a Java client. In addition to this, communication between the receptionist and collection server is through a protocol, enabling distributed configurations of the digital library. In response to a user’s query, a receptionist might contact two collection servers running on different computers and present the result, seamlessly merged, to the user. This ability is used in the digital music library to support multiple indexes to large collections that are divided into smaller sections and distributed to multiple computers.

For now, our interest is in how the receptionist component fits in with the workbench, and again the answer is smoothly due to forward planning. Whereas in the workbench instructions come directly from the user, in the digital library they arrive via the receptionist. For the workbench to be used to support musical queries, therefore, all the receptionist need do is translate these requests into meaningful instruction for the workbench (aided by the specially inherited music collection server) and to format the result into a form the digital library is expecting. The object-oriented design of the digital library architecture is such that once more, implementation is a matter of inheriting a new ob-
ject—this time a specialized music receptionist—that over-
rides specific member functions.

Used in this way, it is true that the full power of the
workbench is not utilized by the receptionist, but it is
equally true that the receptionist supplies extra services,
such as browsing titles A–Z, through other non-workbench
related components.

MELDEX: Example Services and Abilities

We now present a selected tour of services and abilities
of the resulting digital music library system, MELDEX,
with emphasis on the music-specific features. We start with
searching by music content using a sung raw audio query,
then move on to combined text and music queries, this time
illustrating a symbolic form of music query entry through a
graphically presented piano keyboard. Also important in a
digital music library is delivery of a music document, and
we present a Java applet designed to enrich the playback
experience. Other features of the digital library not specific
to music content, such as hierarchical phrases and key-
words, are described elsewhere (Bainbridge et al., 2001;
Witten, Bainbridge, & Boddie, 2001; Paynter & Witten,
2001).

Here is one brief text-based example. In the Midimax
collection, an interactive phrase hierarchical of the collec-
tion’s URLs makes for a compelling environment to browse
around. The user starts by entering a single word, and then
all the root phrases containing that word appear in an
interactive panel. Through mouse clicks, a user can expand
and contract phrases, and play the tunes containing the
selected phrase. A particularly nice feature of the hierarchy
is that it naturally groups together songs by band name if
this has been used in its URL.

Raw Audio Queries

Figure 3 shows a user entering a sung fragment of audio
through the Java applet interface to MELDEX; the user is
singing a phrase from Abide With Me. Although not shown,
through the digital library’s preference page the user has
elected to use the dynamic programming algorithm for
matching, compare interval pitches, but ignore durations.
From the query page itself, the user can choose between
matching anywhere in the tune or only at the start (con-
trolled through a pull-down menu in the upper half of the
interface). If so desired, the user can supply a textual query
as well.

The buttons in the bottom portion of the interface control
the acquisition of the music query. Not only can the user
record a query by singing directly (as is shown), the user can
also load in a query saved earlier and see the result of
transcribing the audio as music notation before deciding to
submit it as a search.

Figure 4 shows the resulting matches that are found by
activating the “Begin Search” button. At the top of the page
are navigational aids that access particular services within
the library: home page, preferences, titles sorted alphabeti-
cally A–Z. Next is a short excerpt of music notation displaying the computer’s transcription of the raw audio query. There is also a hyperlinked speaker icon that plays the transcription back as audio should a user wish to hear it. Then comes a ranked list of the matching documents.

In this particular collection (based on the Faivebook dataset), for each matching tune MELDEX lists four icons for retrieving the document in various forms: the first is the scanned original; the second (for the sake of interest) is the computer-generated music notation based on the error-prone optical music recognition process; the third delivers an audio rendition of the piece; and the fourth is the supplementary text that was entered by hand. Following the icons is the title of the tune, a thumbnail visualization that displays where in the tune and how well the query matched, the database name (superfluous in this context, but is more meaningful when there are subcollections such as pop, rock, and classical), and its overall matching score. Further down the page, but not shown in Figure 4, is the familiar next link to view subsequent pages (and where appropriate previous pages).

**Enriching Playback**

To enrich the audio delivery experience of a tune retrieved by a melodic query, a playback applet has been developed. If, instead of clicking on one of the four icons to the left of a matching title, the user clicks on the thumbnail visualization, the playback applet is activated. Figure 5 shows one such example. At the top, a larger version of the visualization is shown, and a set of control buttons similar to a tape or CD player is provided below that lets the user play, skip to the next matching segment of tune, pause and stop the tune. As the tune is played, a time-based progress bar located between the enlarged visualization and control panel is used to proportionally show where in the tune the playback currently is. Additional controls vary the tempo and volume.

**Text and Symbolic Entry Music Queries**

Figure 6 shows a combined text and music query being entered, where an alternative form of music query entry is demonstrated. The user has entered the term
“Beethoven” in the text box in the upper half of the screen, and is in the process of tapping out a music query on the graphical rendition of a piano keyboard. Since the purpose of this form of music input is to eliminate the possibility of transcription errors in the entered query, the note durations are controlled precisely through the central part of the interface. Employing an HCI principle known as equal opportunity, the user is free to click on...
the music notation showing the duration they want, move
the slider to the required duration, or enter the duration
directly in the text box provided, whichever makes the
most sense to them at any given instance. All three
methods are linked, and changing any one automatically
updates the other two.

The remaining controls provide the same abilities as
those in Figure 3: pull-down menus to control how much of
a tune is searched for a match and the like; and button
controls to record, save, and load queries.

Conclusions

In this article, we have described the context to the music
information retrieval work at the University of Waikato.
Motivated by a digital music library project, a workbench
has been developed that not only provides an environment
for experimentation, but is engineered to integrate smoothly
with the group’s wider digital library software architecture,
forming a key component in the services offered.

Using this design, the concept of plugins provides a
flexible approach for importing music files in a range of
formats, and the ability for plugins to take options allows for
the possibility of filtering or expanding the input stream of
music data. Both are desirable abilities given the different
characteristics and needs of the datasets.

One filter, for example, RepairFilter, identifies repeated
fragments of tune using a linear space and time algorithm.
While undoubtedly a crude method with little to no music
theory involved, in practice we have found it effective on
MIDI file–based collections. In a large collection with mil-
lions upon millions of notes, it is a useful way of reducing
the index to prominent parts of a tune. Furthermore, by
starting a query with one of the identified motifs, tunes that
are most likely variations of the same tune are also found.
This is how MELDEX provides its “find more tunes like
this” service.

The distributed ability of the general digital library ar-
chitecture is also exploited by MELDEX for large-scale
collections, such as ones based around the Midimax dataset.
In addition to basing matching on motifs, parallel indexes
can be distributed around several computers, and even a
single index can be divided into constituent parts for distri-
bution. The receptionist component of the design is respon-
sible for issuing the query to the various collection servers
and for gathering in and merging the results.
Human computer interaction is also an important aspect of the work, which benefits greatly from the expertise of the wider digital library project. Above we presented two applets developed in Java that provide better user interface support than is available using HTML alone. Using the user’s own computer resources, the first applet provides a rich environment for constructing a music query with the ability to playback, save, record, and show the user the query transcribed before the more costly operation of searching is activated on the digital music library server. The second applet enriches the playback experience of the matching tunes. It displays a visualization of where in the tune matches occurred and how close each of these matches was to the original query. It also plays the tune and provides the ability to skip ahead to the next section of tune that matched the query.

References
