Expressive and efficient retrieval of symbolic musical data

Michael Droettboom, Ichiro Fujinaga,
Karl MacMillan
Peabody Conservatory of Music
The Johns Hopkins University
{mdboom,ich,karlmac}@peabody.jhu.edu

Mark Patton, James Warner,
G. Sayeed Choudhury, Tim DiLauro
Digital Knowledge Center
Milton S. Eisenhower Library
The Johns Hopkins University
{sayeed,timmo,mpatton,jwarner}@jhu.edu

ABSTRACT

The ideal content-based musical search engine for large corpora must be both expressive enough to meet the needs of a diverse user base and efficient enough to perform queries in a reasonable amount of time. In this paper, we present such a system, based on an existing advanced natural language search engine. In our design, musically meaningful searching is simply a special case of more general search techniques. This approach has allowed us to create an extremely powerful and fast search engine with minimal effort.

1. INTRODUCTION

The Lester S. Levy Collection of Sheet Music represents one of the largest collections of sheet music available online. The Collection, part of the Special Collections of the Milton S. Eisenhower Library at The Johns Hopkins University, comprises nearly 30,000 pieces of music (Choudhury et al. 2000). It provides a rich, multi-facetted view of life in late 19th and early 20th century America. Scholars from various disciplines have used the Collection for both research and teaching. All works in the public domain are currently available online as JPEG images. The user can browse the collection by category or search based on metadata, such as author, title, publisher, and date. Musical searches, such as finding a particular melodic or rhythmic pattern, are not currently possible since the collection is not in a symbolic music representation.

To solve this problem, an optical music recognition (OMR) system is being developed that will allow for the rapid conversion of the images into a symbolic musical representation (Choudhury et al. 2001). We chose GUIDO as the target representation language due to its simplicity and extensibility (Hoos and Hamel 1997). Having music in a symbolic format opens the collection to sound generation, musicological analysis and, the topic of the present paper, musical searching. The system performing these searches must be expressive enough to perform both simple and sophisticated searches. Also, it must be efficient enough to search through a large corpus in a reasonable amount of time. By extending an existing advanced natural language search engine with simple filters and user-interface elements, we have been able to create a musical search engine that meets these demands.

2. CAPABILITIES

Our musical search engine supports melodic and rhythmic searches. Search queries can also include the notion of simultanaeity. That is, events can be constrained to occur at the same time as other events. The search engine as described here is limited to standard-practice Western music, though modifications could be made to support other musical traditions.

These capabilities are supported by the three main features of the core search engine: secondary indices, partitions, and regular expressions (discussed in Section 5). Secondary indices allow the amount of specificity to vary with each token. Search queries can be performed upon specific discontiguous partitions of the corpus. Regular expressions allow advanced pattern matching.

2.1 Extensibility

Other types of musical searching would require additional layers of musical knowledge to be built on top of the search engine, though they are not prohibited by the general design of the search engine itself. Any analytical data that can be derived from the score data can be generated ahead of time and used as search criteria. This data can be gererated by new custom tools or by existing analysis tools such as the Humdrum toolkit (Huron 1999).

For example, harmonic searches with respect to harmonic function could be added. Western tonal harmonic theory is ambiguous, making it difficult to objectively interpret and label harmonies. However, assuming an acceptable solution to these issues could be found, labeling or harmonic function could be implemented as an input filter.

Also, the core search engine does not include any notion of melodic similarity. This is an open problem strongly tied to subjective matters of human perception (Hewlett and Selfridge-Field 1998). It is possible for a specialized front-end to include notions of melodic similarity by generating special search queries. The search query language of the

core search engine is expressive enough that these advanced features could be added without modifying the core itself.

2.2 Meeting diverse user requirements

We define the users of our musical search engine as anyone who wants to access the collection in a musical way. Of course, the needs of different users is greatly varied. A non-musician may want to hum into a microphone to retrieve a particular melody. A copyright lawyer may want to track the origins of a particular melody, even melodies that are merely similar. A musicologist may want to determine the frequency of particular melodic or rhythmic events. To meet these diverse needs, it is necessary to provide different interfaces for different users. The set of interfaces is arbitrary and can be extended as new types of users are identified. It may include graphical applications, web-based forms and applets, or text-based query languages. Audio interfaces, with pitch- and rhythm-tracking may also be included. The purpose of these interfaces is to translate a set of user-friendly commands or interactions into a query string accepted by the search engine. The details of that query can be hidden from the end-user and therefore can be arbitrarily complex.

At present, we have focused our attention on the core search engine itself. In the second phase of the project, the user interfaces will be developed in collaboration with a usability specialist.

3. PRIOR ART

None of the available options we evaluated met the needs of the diverse user base of the collection, or could handle the large quantity of data in the complete Levy collection. In particular, we evaluated two existing search engines available for public use on-line: Themefinder (Huron et al. 2001) and MELDEX (McNab et al. 1997).

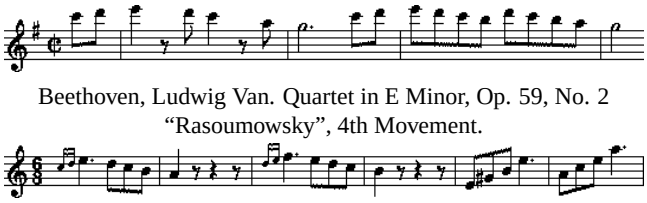
3.1 Themefinder

Themefinder's goal is to retrieve works by their important themes. These themes are manually determined ahead of time and placed in an incipit database.

One can query the database using five different kinds of search queries: pitch, interval, scale degree, gross contour, and refined contour. These five categories served as the inspiration for a subset of our basic query types. The user can query within an arbitrary subset of these categories and then intersect the results. However, Themefinder does not allow the user to combine these query types within a single query in arbitrary ways. For instance, a user may know the beginning of a melodic phrase, while the ending is more uncertain. Therefore, the user may want to specify exact intervals at the beginning and use gross contours or wild-cards at the end. Unfortunately, in Themefinder, the user must have the same level of certainty about all of the notes in the query. Unfortunately, this is not consistent with how one remembers melodies (McNab et al. 2000).

In addition, Themefinder does not have a notion of rhythmic searching. While its invariance to rhythm can be an asset, it can also be cumbersome when it provides too many irrelevant matches. Figure 1 shows the results of a query where one result is more relevant than the other. Such queries may return fewer false matches if they could include rhythmic information.

The searches themselves are executed in Themefinder using a brute-force method. The entire database is linearly searched for the given search query string. While this is acceptable for the 18,000 incipits in Themefinder's largest database, it may not scale well for searching across a full-text database such as the Levy collection.



Beethoven, Ludwig Van. Quartet in E Minor, Op. 59, No. 2 "Rasoumowsky", 4th Movement.

Beethoven, Ludwig Van. Sonata No. 4, in A Minor, Op. 23, Violin and Pianoforte, 1st Movement.

Figure 1: **These two incipits start with the identical set of pitches, $[c\ d\ e\ d]$, but with different rhythmic content. With better rhythmic specificity, irrelevant results could be eliminated.**

3.2 MELDEX

The simple text-based query strings in Themefinder are easy to learn and use by those with moderate musical training. MELDEX, however, has a more natural interface for non-musicians. The user "sings" a melody using a syllable with a strong attack such as "tah." The pitches of the melody are determined using pitch-tracking, and the rhythm is quantized. The results are used as the search query. The query is approximately matched to melodies in the database using a fast-matching algorithm related to dynamic programming. While this approach is highly effective for non-musicians and simple queries, it is limiting to those wanting more fine-grained control.

4. THE CORE SEARCH ENGINE

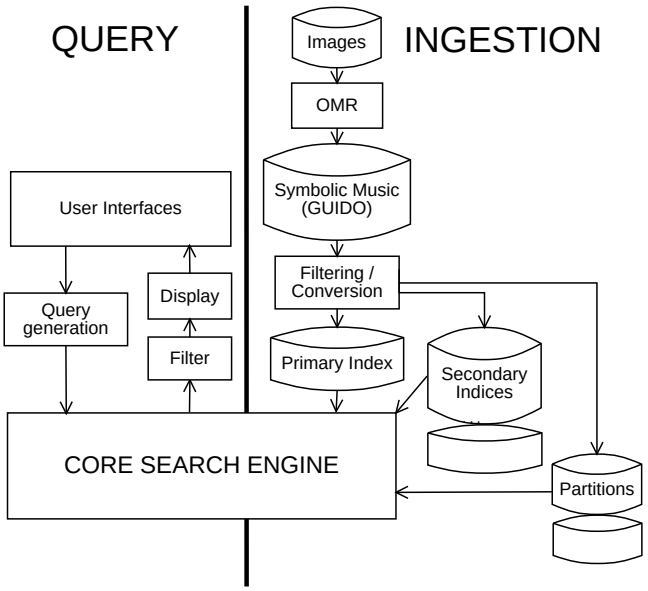


Figure 2: **Workflow diagram of the musical search engine.**

The core search engine in our system was originally developed for text-based retrieval of scores based on their metadata and full-text lyrics. Its overall design was inspired by recent developments in the field of natural-language searching (DiLauro et al. 2001). These features allow the user to perform search queries that derive power and meaning from features of natural languages, such as parts of speech, rhyming scheme, and meter. While not originally intended for musical searching, it was soon discovered that the core was very well suited for searching across symbolic musical data.

This core itself did not need to be modified to support music searching. Instead, specialized filters and front-ends were added to adapt it to the music domain. In the ingestion stage, the data is filtered to store it in the appropriate indices and partitions (explained below). When searching, special user interfaces handle the details of generating search query strings and filtering and displaying the resulting data. Figure 2 shows how the individual parts of the system fit together to ingest and query the data.

4.1 Inverted indices

Many search engines, including ours, are built on the concept of an inverted list. For a complete discussion of inverted index search engines, see Witten et al. (1999).

Sequential data, such as English prose or melodic data, is stored on disk as a sequence of atoms. In the case of English, the atom is the word and the sequence is simply the ordered words as they appear in sentences and paragraphs. Take for example the following sentence:

```
To be , or not to be , that is
the question .
```

Note that both words and punctuation are treated as indivisible atoms. To search for a particular atom in this string, a computer program would need to examine all thirteen atoms and compare it with a query atom. To increase searching efficiency, an inverted index search engine would store this string internally as:

```
, ⟶ 3, 8
. ⟶ 13
be ⟶ 2, 7
is ⟶ 10
not ⟶ 5
or ⟶ 4
question ⟶ 12
that ⟶ 9
the ⟶ 11
to ⟶ 1, 6
```

Here, each atom in the string is stored with a list of numbers indicating the atom's ordinal location within the string. The set of words in the index is called the vocabulary of the index. To search for a particular atom using the index, the program needs only to find that word in the vocabulary and it can easily obtain a list of indices (or pointers) to where that atom is located within the string. Since the vocabulary can be sorted, the lookup can be made faster using hashing or a binary search.

Inverted indices perform extremely well when the size of the vocabulary is small relative to the body of data, or corpus. In the case of English, of course, the vocabulary is much smaller relative to the size of all the works written in that language. This property also allows us to improve the efficiency of musical searching as we will see below.

5. THE MUSICAL SEARCH ENGINE

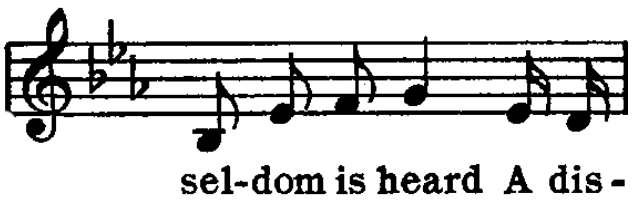


Figure 3: **A measure of music, from the Levy collection, used as an example throughout this section.**

5.1 Secondary indices

The searchable atom, in the case of music, is not the word, but the musical event. Such events include anything that occurs in the score, such as notes, rests, clefs, and barlines. Each of these events, of course, can have many properties

associated with it. For instance, the note $b^\flat$ at the beginning of the fragment in Figure 3 has the following properties:

- **Pitch name:** *b*
- **Accidental:** $\flat$
- **Octave:** first octave below middle-*c*
- **Twelve-tone pitch:** 10^{th} semitone above *c*
- **Base-40 pitch:** -3
- **Duration:** eighth note
- **Interval to next note:** perfect 4^{th}
- **Contour (direction) to next note:** up
- **Scale degree:** *so* (5^{th} scale degree in $E^\flat$ major)
- **Lyric syllable:** "sel–"
- **Metric position:** Beat 1 in a $\frac{6}{8}$ measure

All of these properties are self-evident, with the exception of base-40 pitch, which is a numeric pitch representation where the intervals are invariant under transformation (Hewlett 1992). Note also, we use GUIDO-style octave numbers, where the octave containing middle-*c* is zero, and not ISO-standard octave numbers.

The concept of secondary indices allows the individual properties of each atom to be indexed independently of any other properties. This allows search queries to have arbitrary levels of specificity in each event. The set of properties can be extended to include any kinds of data that can be extracted from the source musical data. For example, if the harmonic function of chords could be determined unambiguously, a secondary index containing chord names in Roman numeral notation could be added. In our design, we use secondary indices to handle the properties of events that change from note to note. Continuous properties of events, that are carried from one event to the next, such as clefs, time signatures, and key signatures, are handled using partitions, explained below (see Section 5.2).

5.1.1 Ingestion

During the ingestion phase, the source GUIDO data is first converted to an interim format where all of the event's properties are fully specified. For example, the GUIDO code of Figure 3 is as follows:

```
[ \clef<"treble"> \key<-3> \time<6/8>
    \lyric<"sel-"> b&-1*/8.
    \lyric<"dom"> e&*0
    \lyric<"is"> f
    \lyric<"heard"> g/4
    \lyric<"A"> e&/16
    \lyric<"dis-"> d
]
```

Each event is then extended so it is fully specified. In this format, each note event is made up of a number of property fields of the form:

pitch-name, accidental, octave, twelve-tone-pitch, base-40-pitch, duration, interval, contour, scale-degree, lyric-syllable, metric-position

Figure 4 shows the example in fully-specified symbolic representation.

Each one of these fields is used to index the database in a particular secondary corpus. For example, if the notes in the example were labeled 1 through 6, the data in the secondary indices may look something like:

- **Pitch name**
  - `a` ⟶
  - `b` ⟶ 1
  - `c` ⟶
  - `d` ⟶ 6
  - `e` ⟶ 2, 5
  - `f` ⟶ 3
  - `g` ⟶ 4
- **Accidentals**
  - `n` ($\natural$) ⟶ 3, 4, 6
  - `&` ($\flat$) ⟶ 1, 2, 5
- **Octave**
  - `-1` (first octave below middle-*c*) ⟶ 1
  - `0` (first octave above middle-*c*) ⟶ 2, 3, 4, 5, 6
- **Duration**
  - `1/4` (quarter note) ⟶ 4
  - `1/8` (eighth note) ⟶ 1, 2, 3
  - `1/16` (sixteenth note) ⟶ 5, 6

5.1.2 Searching

The search query itself is simply a series of events. Each event can be indicated as specifically or as generally as the end user (as represented by a user interface) desires. For example, the following query would match any melodic fragment that begins on a $b^\flat$ eighth note, has a sequence of 3 ascending notes, ending on a g:

```
b,&,1/8 / / / g
```

To execute a search query using secondary indices, the search engine looks up each "parameter" in their corresponding secondary indices, and retrieves tokens in the primary index. These tokens are then looked up in the primary index, returning a list of positions. These lists are intersected to find the common elements. This list of locations is then tested for correct ordering.

5.1.3 User interfaces

A text-based user interface may allow a user to be very specific in the query, and then incrementally remove layers of specificity until the desired match is retrieved. An audio-based user interface could be more or less specific depending on the pitch tracker's confidence in each event.

```
[ \clef<"treble"> \key<-3> \time<6/8>
    b, &, -1, 10, -3,  1/8, P4, /, so,  "sel-",     0
    e, &,  0,  3, 14,  1/8, M2, /, do,   "dom",   1/8
    f, n,  0,  5, 20,  1/8, M2, /, re,    "is",   1/4
    g, n,  0,  7, 25,  1/4, M3, \, mi, "heard",   3/8
    e, &,  0,  3, 15, 1/16, m2, \, do,     "A",   5/8
    d, n,  0,  2,  9, 1/16, M2, \, ti,  "dis-", 11/16
]
```

Figure 4: **Fully specified symbolic representation of the example in Figure 3**.

5.1.4 Efficiency

One of the efficiency problems with this approach is that the vocabularies of the individual secondary indices tend to be quite small, and thus the index lists for each atom are very large. For instance, the "pitch name" secondary index has only seven atoms in its vocabulary (*a* - *g*). "Accidentals" is even smaller: {𝄫, ♭, ♮, ♯, ×}. Therefore, a search for a $b^\flat$ must intersect two very large lists: the list of all *b*'s and the list of all flats. However, the search engine can combine these secondary indices in any desired combination off-line. For example, given the "pitch name" and "accidental" indices, the search engine can automatically generate a hybrid index in which the vocabulary is all possible combinations of pitch names and accidentals. The secondary indices can be combined automatically in all possible combinations.

5.2 Partitions

Partitioning allows the corpus to be divided into discontiguous, non-overlapping regions. It can be used to restrict a search query to a particular part of the corpus. Each partition is a collection of regions of a category. Each region is named and has a list of its start and stop positions.

In our music search engine, the metadata is used to divide the corpus into regions. For example, all works by a given composer would make up a discontiguous region in the "composer" partition. Partitions exist for all types of metadata in the collection, including date, publisher, location, etc.

In addition, we have extended partitioning to include musical elements derived directly from the GUIDO data. Regions are generated from key signatures, clefs, time signatures, measures, movements, repeats, etc. This allows for searching for a particular melody in a particular key and clef, for example.

Scores are also partitioned at the most atomic level by "moments". A moment is defined as any point in time when any event begins or ends in any part. Each moment within a score is given a unique id, and all events active at a given point are included in a moment region. In this way, one can search for simultaneous polyphonic events very efficiently.

5.2.1 Ingestion

When a new piece is added to the corpus, the data is partitioned automatically. First, the metadata regions, such as title, composer, and date, are set to include the entire piece. As the piece is scanned, continuous musical elements, such as clef, key signature, and time signature, are regioned on the fly. Therefore, when the ingestion filter sees a "treble clef" token, all further events are added to the "treble clef" region until another clef token is encountered. Lastly, events are added to the moment regions on an event-by-event basis.

For the example in Figure 3, again assuming the notes are numbered 1 through 6, the partitions may look something like:

- **Title partition**
  "Home on the range" ⟶ [1, 6]
- **Clef partition**
  Treble clef ⟶ [1, 6]
- **Time signature partition**
  $\frac{6}{8}$ ⟶ [1, 6]

5.2.2 Searching

Extending the example in Section 5.1.2, the user may wish to limit the search to the key signature of $E^\flat$*-major*:

```
( b,&,1/8 / / / g ) @ key:"E major"
```

Here the non-partitioned search query is performed as described above, and then the results are intersected with the results of the partition lookup. Since in our case, the entire range of notes [1, 6] is in the key signature of $E^\flat$*-major*, the query will retrieve the example in Figure 3.

5.3 Regular expressions

The core search engine supports a full complement of POSIX-compliant regular expressions. Regular expressions, a large topic beyond the scope of this paper, are primarily used for pattern-matching within a search string (Friedl 1997).

Many users find regular expressions difficult and cumbersome ways to express searches. However, it is our intent that most of these details will be hidden from the user by appropriate interfaces. For example, regular expressions would be very useful for an interface that allowed searching by melodic similarity.

6. CONCLUSION

Based on an existing advanced natural-language search techniques, an expressive and efficient musical search engine was

developed. Its special capabilities include: secondary indices for gradiated specificity, partitions for selective scope, and regular expressions for expressive pattern matching. This allows users with differing search needs to access the database in powerful and efficient ways.

7. ACKNOWLEDGMENTS

The second phase of the Levy Project is funded through the NSF's DLI-2 initiative (Award #9817430), an IMLS National Leadership Grant, and support from the Levy Family.

8. REFERENCES

Choudhury, S., T. DiLauro, M. Droettboom, I. Fujinaga, B. Harrington, and K. MacMillan. 2000. Optical music recognition within a large-scale digitization project. *ISMIR 2000 Conference*.

Choudhury, G. S., T. DiLauro, M. Droettboom, I. Fujinaga, and K. MacMillan. 2001. Strike up the score: Deriving searchable and playable digital formats from sheet music. *D-Lib Magazine*: 7(2).

DiLauro, T., G. S. Choudhury, M. Patton, J. W. Warner, and E. W. Brown. 2001. Automated name authority control and enhanced searching in the Levy collection. *D-Lib Magazine*: 7(4).

Friedl, J. E. F. 1997. *Mastering regular expressions.* Sebastopol, CA: O'Reilly.

Hewlett, W. B. 1992. A base-40 number-line representation of musical pitch notation. *Musikometrika* 4: 1–14.

Hewlett, W. B., and E. Selfridge-Field. 1998. *Melodic similarity: Concepts, procedures and applications.* Cambridge, MA: MIT Press.

Hoos, H. H., and K. Hamel. 1997. GUIDO music notation: Specification Part I, Basic GUIDO. Technical Report TI 20/97, Technische Universität Darmstadt.

Huron, D., W. Hewlett, E. Selfridge-Field, et al. 2001. How Themefinder works. `http://www.themefinder.org`

Huron, D. 1999. *Music research using Humdrum: A user's guide.* Menlo Park, CA: Center for Computer Assisted Research in the Humanities.

McNab, R. J., L. A. Smith, D. Bainbridge, and I. H. Witten. 1997. The New Zealand Digital Library MELody inDEX. *D-Lib Magazine*: 3(5).

McNab, R. J., L. A. Smith, I. H. Witten, and C. L. Henderson. 2000. Tune retrieval in the multimedia library. *Multimedia Tools and Applications*: 10(2/3) 113–32.

Witten, I., A. Moffat, and T. Bell. 1999. Managing gigabytes. 2nd Ed. San Francisco, CA: Morgan Kaufmann.