

Database Approaches to the Modelling and Querying of Musical Scores: a Survey

Adel Aly, Olivier Pivert, Virginie Thion

Univ. Rennes, IRISA, France

Abstract. This paper presents a survey of digital musical score databases, focusing on symbolic representation of the music content (as opposed to audio representation) in the context of Music Information Retrieval (MIR). We first provide a primer on Western classical music notation for unacquainted readers. Then, the core of our study categorizes and discusses various approaches to the data management layer of digital score libraries (DSLs), emphasizing their data models and query specifications. Special attention is given to the differences between ASCII-based, semi-structured, and graph-based data models, alongside high-level abstract models. The paper concludes with a discussion comparing these models based on several criteria, aiming to point out their respective strengths and limitations. This work synthesizes existing knowledge in the field of symbolic MIR and identifies promising areas for future research.

Keywords: Musical Scores · Symbolic Representation · Databases · Querying

1 Introduction

The field of Music Information Retrieval (MIR) emerged in the late 1990s, fueled by advances in data management and the growing volume of music data. MIR, as a multidisciplinary domain, integrates musicology, digital libraries, and computer science, among others, and focuses on music data retrieval and analysis, model development, and graphical user interfaces for music data interaction.

This survey zeroes in on MIR systems dealing with notated music (symbolic representation) in Western Classical notation, differentiating them from audio-focused MIR techniques. Symbolic MIR approaches concentrate on data management within Digital Score Libraries (DSLs), particularly through database management systems for expressive and efficient querying and storage. The readers with an interest in the expansive field of MIR are encouraged to explore comprehensive literature surveys such as those proposed in [3,24,15,2,16,21].

Advances in the MIR field have led to applications ranging from standalone music score editors to web applications that manage large score collections, offering services for querying, transforming, analyzing, and visualizing these collections. Such an application relies on a structured encoding of musical content, termed its *(logical) data model*, which definition constitutes a significant scientific challenge. Among other requirements for DSLs, it is crucial to provide

features that enable (i) a secure and persistent storage of potentially large music collections, and (ii) expressive and efficient tools for querying this data. Classically, such features are ensured by implementing the data storage and querying through a database management system (DBMS).

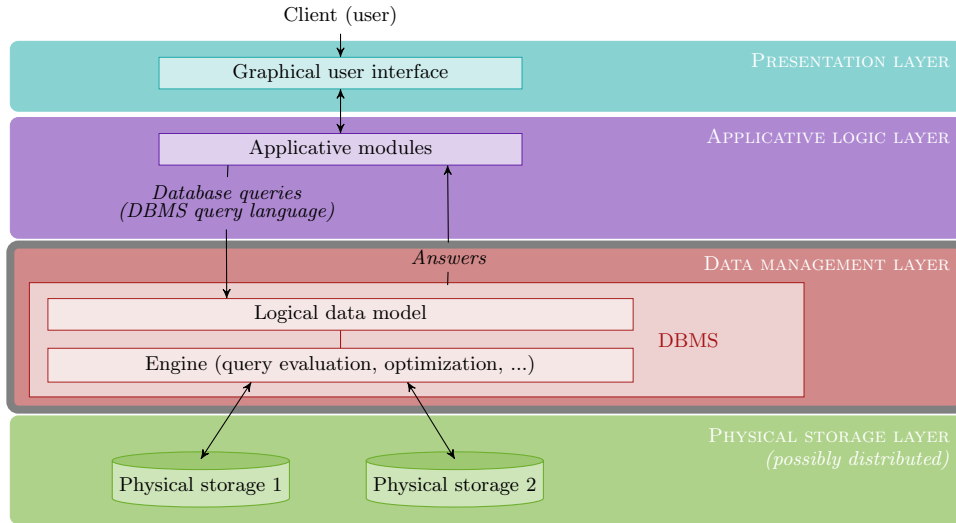


Fig. 1. Typical architecture of a SGBD-based DSL

Figure 1 illustrates the main software layers that compose a typical DSL application. The Presentation layer (in blue) is responsible for the user interface, including visualization tools and interfaces for navigating and querying musical score collections. The Applicative layer (in purple) holds the programs implementing the business logic used to process user inputs. These programs retrieve data by making requests to the Data management layer. The Data management layer (in red) stores, manages, and retrieves musical score data.¹ This layer, which can be as simple as a file system mapping or as complex as a full-fledged DBMS, ensures efficient data access, security, and integrity. It exposes data through the logical data model, which details how is structured available data. The Applicative layer expresses requests according to this data model.

In this paper, we explore the different kinds of data model that were proposed in the literature to represent, store and query digital musical scores, according to the kind of DBMS that can be used to handle collections of such data.

The paper is structured as follows. Section 2 introduces basic concepts of musical scores. Section 3 explores document-oriented approaches to score man-

¹ Musical score data is physically stored in the Physical storage layer (in green in Figure 1).

agement, differentiating between ASCII-based and semi-structured types. Section 4 examines a graph database model and Section 5 discusses more high-level models. Section 6 presents a comparative discussion of these models, and Section 7 concludes the paper.

2 The Content of a Musical Score

This section introduces key concepts of Western music notation, a detailed and expressive language for representing musical compositions. At its core, music notation translates sound properties — pitch and duration — into visual symbols arranged on a staff, a set of five horizontal lines.

Pitches are notated using a combination of letters (A through G), octave indicators (1 through 7), and accidentals (sharps \sharp and flats \flat) to specify the exact frequency. The staff is anchored by a clef, which indicates the pitch range. The vertical position of a note on the staff denotes its pitch, while the note’s shape (including the head, stem, and flags) defines its duration. Rests, distinct symbols on the staff, denote silences of various lengths.

Example 1 (Running example - A simple music score). Let us take Figure 2.(a) as a running example, an extract of the BWV 846 prelude of the well-tempered clavier, adapted for the piano. Let us consider the first staff (with the treble clef). Its first element is a silence, followed by an $E4$ note, then an $A4$, an $E5$, etc. The shape of the note (head — black or white —, stem and flag) determines the duration of the sounds. A ♩ note is four times shorter than a ♪ note, and eight times shorter than a ♫ one. The notes are graphically synchronized over the staves. Additional accidental alterations (*sharp* and *flat*) may be added to the notes themselves, such as in the second measure (\sharp accidentals over the $F4$ notes).

Musical content is organized into measures, time sections delineated by vertical bars on the staff, with their length governed by the time signature. Besides the notes and rests, a score may include additional details such as dynamics, articulation marks, and metadata such as the composer and title.


In addition to the music score example of Figure 2.(a), we consider a simple musical pattern of Figure 2.(b). This pattern will allow us, in the rest of the paper, to illustrate the querying of a music score, according to the data model chosen to encode the musical content of the score.

Example 2 (Running example - Pattern \mathcal{P}). Let us consider a simple musical pattern that consists in a sequence of four consecutive notes $E4$, $A4$, $E5$ and then $A5$, used to retrieve music scores. In this pattern, only the occurrence of the pitches is tackled², independently of the rhythm (note duration and distribution


² In order to keep query examples as understandable and straightforward as possible — considering the heterogeneous query means that we refer to — we do not deal with note duration or alterations, which would lead to intricate examples in some cases. However, we discuss these matters in Section 6.

THE WELL-TEMPERED CLAVIER, BWV 846
(simplified extract)

JS Bach



(a) Simple music score


(b) Pattern \mathcal{P} **Fig. 2.** Running example: simple musical score and musical pattern

in the measures) and independently of the rendering (such as stems orientation and beams). This pattern, called \mathcal{P} , is depicted in Figure 2.(b).

3 Document-Oriented Models

Document-oriented approaches have historically paved the way for the digital representation and processing of sheet music. These formats can be broadly categorized into two distinct types: ASCII-based documents and semi-structured documents. ASCII-based documents utilize plain text to represent musical scores while semi-structured documents employ XML-based formats.

Most of these formats were originally developed for representation, sharing, and interoperability among various music software applications, rather than as data models for DBMS. Today, these formats continue to be used, underpinning several well-established platforms in the domain of music content management.

3.1 ASCII-Based Documents

In the field of MIR, ASCII-based systems are significant for their simplicity and readability, serving early on to facilitate musical data exchange and software support for music representation. Early systems such as DARMS, SCORE, and MuseData, as detailed in [22], were foundational, influencing subsequent ASCII-based notation development. This section focuses on the ABC notation, a widespread example of ASCII-based music notation, illustrating its standard attributes. While also acknowledging other formats, the focus is on delineating their distinctive features compared to ABC.

The ABC notation [1] (first release 1993 – latest release 2011), is a text-based encoding system for music, initially developed for typesetting folk and

```

1 X:1
2 T:THE WELL—TEMPERED CLAVIER, BWV 846 (simplified extract)
3 C:JS Bach
4 L:1/16
5 M:4/4
6 K:C
7 V:1 treble
8 V:2 bass
9 V:1
10 z EAe aAea z EAe aAea | z D^FA dFAd z DFA dFAd ||
11 V:2
12 C8 C8 | C8 C8 ||

```

Fig. 3. Representation of the running example in ABC notation (v 2.1)

traditional tunes. It uses letters A to G to represent pitches, with uppercase for lower and lowercase for higher octaves. For example, uppercase C is C4 (middle C), while lowercase c is C5. Other octave shifts are indicated with commas for lower octaves (C, for C3) and apostrophes for higher octaves (c' for C6). Accidentals are prefixed (^ for sharp, _ for flat) and affect all identical pitch notes within the measure (following the common music notation convention). Note durations are equal to a default value determined by the unit note length field (see details below) unless specified otherwise using numerical suffixes. ABC supports additional musical elements like chords and lyrics. Measures are marked by bar symbols (|), and non-musical data is included in information fields (e.g., K: for key signature, T: for title) within the ABC file's header.

Example 3 (Running example - Data modelling in ABC notation). The example provided in Figure 3 is an ABC encoding of the running example of Figure 2.(a). It illustrates ABC notation's information field usage. It details the encoding for a piece, identifying the title (T:THE WELL—TEMPERED CLAVIER...), composer (C:JS Bach), time signature (M:4/4), key signature (K:C major), and the unit note length (here, L:1/16, a sixteenth note). Two voices, V:1 treble and V:2 bass, represent the piano's keyboard part. For instance, the first measure of V:1 combines a rest (z) and subsequent notes (E (i.e. E4), A (i.e. A4), e (i.e. E5), etc.), all with the default duration of a sixteenth note. In V:2, the notation C8 signifies a C4 note (C8) extended to 8 times (C8) the unit note length, so a half note. Accidentals are applied according to standard practice, implicitly influencing all similar subsequent pitches within the measure.

When querying ABC files, many approaches use general-purpose text processing tools. Specifically, Unix-based regular expressions within piped commands are commonly employed to perform simple pattern matching and statistical analyses on the data (see the example below).

Example 4 (Running example - Unix command-line querying of an ABC file). Listing 1.1 outlines a Unix command line approach allowing to extract the pat-

tern of Figure 2 from an ABC file, using a series of piped commands that transform and filter the data. In this example, the first `sed` command removes spaces to focus on the sequence of notes regardless of their visual spacing. Another `sed` command eliminates the `|` characters, abstracting the notes from their measure divisions. The first `grep` command filters out lines with information fields, and the final `grep` identifies the specific note sequence (E4, A4, E5, A5) without considering alterations or note durations.

```
sed "s/_//g" runningExampleABC.txt | sed "s/|//g" | grep -v ":" | grep -E "[^\^|-)]|^)E([0-9])*A([0-9])*e([0-9])*a([0-9])*" --color
```

Listing 1.1. Unix command-line querying implementing \mathcal{P}

Defining regex expressions for pattern searching in music scores is notably complex, requiring in-depth knowledge of Unix commands and a precise understanding of their syntax and options. The difficulty is compounded when considering the nuances of ABC notation, such as the handling of chords, which are not accounted for in simpler queries and could lead to inaccuracies if chords are interpreted as sequential notes instead of simultaneous ones. Moreover, the inherent challenge of dealing with implicit accidentals in ABC notation — where alterations may not be explicitly attached to each note but instead governed by key signatures or measure-specific alterations — adds another layer of complexity, making an exhaustive and accurate query significantly more intricate to construct.

Besides the ABC notation, other significant ASCII-based music encoding methods include Humdrum [12] and GUIDO [11]. Humdrum uses its `**kern` notation to organize musical staves in columns, reflecting the timing and simultaneity of notes. GUIDO, like ABC, encodes scores in plain ASCII but uniquely employs probabilistic matching and first-order Markov chains to establish a similarity metric between music scores.

3.2 Semi-Structured Documents

Driven by the need of interoperability between systems and tools dedicated to music notation, semi-structured models have recently gained popularity for encoding (Western) digital sheet music. The most widespread ones are MusicXML [8,9] and MEI [20,14]. They both encode the complete and detailed content of sheet scores (including meta-data, and detailed graphical specifications that dictate how the notation content has to be visually rendered).

MusicXML (first release 2004 – latest release 2021) is an XML dialect format developed for the interchange and distribution of digital sheet music (Western musical notation). MusicXML documents start with some metadata specified in headers, such as title, composer and instruments, followed by the content of the music score itself, encoded measure by measure, where each note element can

encapsulate detailed information including its pitch — defined by step, octave, and accidental —, its duration and an accidental where applicable.

As in other XML-based file formats, MusicXML data is organised in an n-ary ordered tree of attributed elements, where the elements model the components of the score (metadata or musical content). This tree structure is modelled by nesting the elements. For instance, the `note` element is delimited by an opening tag (`<note>`) and a closing one (`</note>`), and may embark attributes attached to the opening tag, and sub-elements between the opening and the closing tag.

Example 5 (Running example - Data modelling in MusicXML). Figure 4 (page 7) illustrates the encoding of a segment of the running example of Figure 2.(a). This segment begins with the declaration of the second measure at line 253. Subsequent lines detail the measure’s components, including a rest (spanning lines 254 to 260) and the sixteenth notes *D*4 (across lines 261 to 271) followed by *F*4♯ (from lines 272 to 284). Notably, this excerpt includes rendering details such as the orientation of note stems (indicated by the `stem` tag).

253	<measure number="2" >	272	<note>
254	<note>	273	<pitch>
255	<rest/>	274	<step>F</step>
256	<duration>1</duration>	275	<alter>1</alter>
257	<voice>1</voice>	276	<octave>4</octave>
258	<type>16th</type>	277	</pitch>
259	<staff>1</staff>	278	<duration>1</duration>
260	</note>	279	<voice>1</voice>
261	<note>	280	<type>16th</type>
262	<pitch>	281	<accidental>sharp</accidental>
263	<step>D</step>	282	<stem>up</stem>
264	<octave>4</octave>	283	<staff>1</staff>
265	</pitch>	284	</note>
266	<duration>1</duration>		
267	<voice>1</voice>		
268	<type>16th</type>		
269	<stem>up</stem>	
270	<staff>1</staff>		
271	</note>	455	</measure>

Fig. 4. Representation of the running example in MusicXML (beginning of the second measure of the first voice)

MusicXML provides two distinct methods for encoding multi-voice scores. The *partwise* representation arranges the score “horizontally”, focusing on individual instrumental or vocal lines one after the other, mirroring how each part appears in isolation within the overall score. In contrast, the *timewise* representation takes a “vertical” perspective, grouping together the declaration of the

measures from different voices or instruments that sound at the same time.

We now consider the querying of such data. The approach presented in [7] proposed to use the XQuery language to express queries over MusicXML documents. Let us first succinctly introduce XQuery. XQuery is a query language designed to query and process XML documents efficiently, leveraging XPath expressions for navigating XML trees and selecting nodes. The core of XQuery’s functionality lies in its FLWOR (For, Let, Where, Order by, Return) construct (cf. the query example below). The For clause specifies a sequence of items, the Let clause assigns values to variables, the Where clause filters results based on conditions, the Order by clause sorts the results, and the Return clause constructs the output.

Example 6 (Running example - XQuery querying of a MusicXML file). Listing 1.2 shows a query, expressed in the XQuery language, that implements the musical pattern \mathcal{P} . The query browses the score staff by staff (in order to accommodate the timewise organization of data³). It then scrutinizes sequences of four notes for matches to the specified musical pattern. The information returned as answers in this query (return clause) is an XML construct that indicates, for each occurrence of the pattern, the staff where the pattern has been found, the measure where the pattern occurs in, and the index of the note in the staff.

In [7], queries are classified into several distinct categories, tailored to address various aspects of musical score content and metadata within MusicXML databases. These categories range from general database content and metadata-related queries to statistical analyses and direct inquiries into music content.

The Music Encoding Initiative (MEI) semi-structured data model [20] (first release 2010 - latest release 2023), is an alternative XML-based data format for sheet music. While MusicXML was primarily designed to be an interchange format between notation editors, MEI provides greater affordances for encoding semantically and structurally rich metadata and music annotation [14]. MEI also contains identifiers associated with the elements present in the score, enabling seamless linking of retrieved information to its location in the data. It is noteworthy that the systematic encoding of identifiers for elements in the MEI format facilitates the annotation of the score content, the cross-checking of data and the rendering result in the context of data interrogation. While no academic work has proposed to query MEI documents with an XML-based query language, it is in theory possible to query MEI documents with XQuery, in a way similar to what is proposed in [7] for MusicXML documents.

³ In a timewise organization of data, the next declared note in the XML declaration may belong to another instrument.


```

(: For each staff :)
let $staff-list := //staff/text()
let $unique-staff-list := distinct-values($staff-list)
for $staff-number in $unique-staff-list
  (: Extract the notes of the staff :)
  let $notes := //note[child::staff/text()=$staff-number]
  [not(exists(child::chord))][not(exists(following-sibling::chord))] (: not in a chord :)
  (: For each sequence of 4 notes :)
  for $i in (0 to count($notes))
  let $s := subsequence($notes, $i, 4)
  (: Check the occurrence of the musical pattern :)
  where $s[1]/pitch/step/text()='E' and $s[1]/pitch/octave/text()='4'
    and not(exists($s[1]/pitch/alter))
    and $s[2]/pitch/step/text()='A' and $s[2]/pitch/octave/text()='4'
    and not(exists($s[2]/pitch/alter))
    and $s[3]/pitch/step/text()='E' and $s[3]/pitch/octave/text()='5'
    and not(exists($s[3]/pitch/alter))
    and $s[4]/pitch/step/text()='A' and $s[4]/pitch/octave/text()='5'
    and not(exists($s[4]/pitch/alter))
return
<result>
  <staff>{$staff-number}</staff>
  <measure>{$s[1]/ancestor-or-self::measure[1]/@number/string()}</measure>
  <index-first-note>{$i}</index-first-note>
</result>

```

Listing 1.2. A XQuery query implementing \mathcal{P}

4 Graph-Oriented Models

Another type of approach from the literature proposes to model symbolic music content data by means of a graph-based model. In a graph-based data model, data is modelled by nodes (entities) and edges between nodes (relations between entities).

In [19], the authors consider a graph data model for representing the (musical) content of a collection of music score, its storing and its querying. The proposed graph-based data model, based on the property graph data one, is called *Muster*. It combines the hierarchical rhythmic decomposition point of view (that of the semi-structured-based model) and the time series point of view (that makes it possible to navigate in data through the sequence of events of a voice). Figure 5 illustrates the modelling of the running example in the graph-based *Muster* model.

Then, graph-pattern queries may be used to query data. The authors have implemented this framework in the Neo4J database management system where they stored a collections of Bach Chorals, and used the Cypher language in order to query such music data.

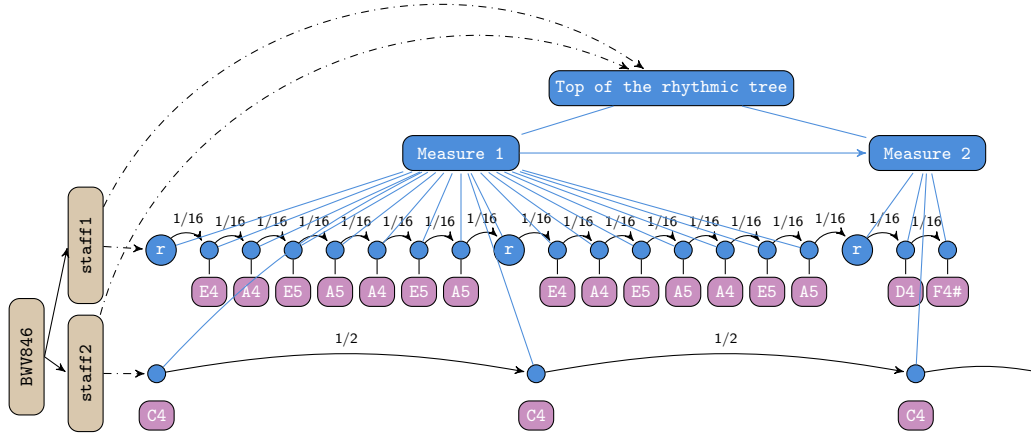


Fig. 5. Graph-based modelling of the running example

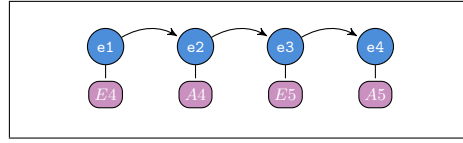
Example 7 (Running example – Graph pattern-based querying of a score). Figure 6.(a) is the graphical representation of the graph pattern implementing the musical pattern \mathcal{P} . It is composed of four musical events. Each musical event is composed of a fact node that embeds the information related to the note associated with the musical event. According to \mathcal{P} , the first fact is an $E4$ (a fact node having the properties class:'e' and octave:4), the second is a $A5$, etc.

Figure 6.(b) is a Cypher query that implements the pattern of Figure 6.(a). The `match` clause defines the shape of the graph-pattern, the `where` clause allows associating additional constraints to the pattern, and the `return` clause defines the information returned by the query as an answer (here, the identifiers of the elements, based on their original MEI encoding, are returned).

Another graph-based data model is used in [13], where the authors propose to convert MusicXML musical scores into RDF [27]. Encoding the music score data in RDF allows querying such data via the SPARQL query language [17,26]. The authors also propose an OWL [25] ontology, called MusicOWL, dedicated to the modelling of knowledge over a musical score.⁴ Then, the querying process is extended by a reasoning based on MusicOWL knowledge. It is worth noticing that the ontology integrates the time series point of view that connects an event to its following one, in order to facilitate the browsing of data through the sequence of events of a voice.

The main differences between the two data models (property graph vs. RDF) in the approaches presented above concern the modelling of metadata and the

⁴ There are other ontologies that have been proposed to model knowledge about musical scores. We consider that ontologies fall beyond the scope of this article. However, MusicOWL is mentioned because [13] focuses on the querying of music score data via the SPARQL query language.

(a) Graph pattern modelling \mathcal{P}

```

match // Searching for the sequences of four events E4 A4 E5 A5
(e1:Event)-[]->(e2:Event)-[]->(e3:Event)-[]->(e4:Event),
(e1)--(f1{class:'e',octave:4}), // e1 is a E4
(e2)--(f2{class:'a',octave:4}), // e2 is a A4
(e3)--(f3{class:'e',octave:5}), // e3 is a E5
(e4)--(f4{class:'a',octave:5}) // e4 is a A5
where // no alteration on F&, f2, f3 and f4
  not exists(f1.accid) and not exists(f1.accid_ges)
  and not exists(f2.accid) and not exists(f2.accid_ges)
  and not exists(f3.accid) and not exists(f3.accid_ges)
  and not exists(f4.accid) and not exists(f4.accid_ges)
return e1.id, e2.id, e3.id, e4.id // Information returned in the answer
  
```

(b) A Cypher implementation of \mathcal{P} **Fig. 6.** Graph pattern \mathcal{P} and its implementation in the Cypher query language

extensiveness of the data model: the RDF modelling includes metadata, and the RDF model is extensive as RDF allows to enrich data by adding links to external sources. Concerning the querying, the graph pattern matching languages Cypher and SPARQL rely on the same principles in the design of a query over musical score data. Using SPARQL may offer reasoning facilities provided by a possibly semantic reasoning based on the content of an ontology. Concerning the querying of the musical content itself, expressing queries and more especially complex ones still remains a non trivial task. Complex SPARQL topological queries over a musical score, e.g. polyphonic queries, are not mentioned in the literature.

5 Time-Series-Oriented Models

The literature also proposed some other abstract, high-level models for modelling the musical content of a score, and extracting information from such modelled data. A first approach is an abstract algebra based on the modelling of the musical content as time series (functions), presented in Subsection 5.1. Another approach relies on the modelling of the notes as weighted points in a two-dimensional Euclidean space, presented in Subsection 5.2.

5.1 Time-Series Algebra

The authors of [4,18,5] introduce an abstract algebra, called *ScoreAlg*, designed for retrieving and processing music content information. *ScoreAlg* is based on

the modelling of the musical content as synchronized time series, setting aside rendering and performance considerations. Conceptually, ScoreAlg is analogous to the relational algebra used for querying relational databases: it is a closed language that serves as a basis for defining the manipulations that can be done over music scores.

ScoreAlg treats the music content as functions which domain is the time and co-domain is the set of musical events. These functions allow the modelling of music scores with time-series data, segmented into discrete intervals, with musical events linked to these intervals. The events are defined as domain values, which can encompass a range of musical elements such as pitches, lyrics, or any other relevant information. In Figure 7, the running example of Figure2.(a) is represented in the formalism of the ScoreAlg algebra. The musical content is composed of two voices, whose musical content (cleared from information related to representation purposes or metadata) is modelled by a function.

$$v_1(t) = \begin{cases} \emptyset, & t \in [0, 1[\\ E4, & t \in [1, 2[\\ A4, & t \in [2, 3[\\ E5, & t \in [3, 4[\\ A5, & t \in [4, 5[\\ A4, & t \in [5, 6[\\ E5, & t \in [6, 7[\\ A5, & t \in [7, 8[\\ \emptyset, & t \in [8, 9[\\ \dots & \\ D5, & t \in [31, 32[\end{cases} \quad v_2(t) = \begin{cases} C4, & t \in [0, 8[\\ C4, & t \in [8, 16[\\ C4, & t \in [16, 24[\\ C4, & t \in [24, 32[\end{cases}$$

Fig. 7. Abstract representation in the algebra

The algebra comprises five structural operators, which use the aforementioned functions. A *synchronization* operator takes two separate voices and concatenates them, resulting in a combined score where the voices remain distinct. A *projection* operator allows to extract one or several voices from a score. A *selection* operator is used to select parts of the score satisfying certain conditions that can be based on domain values or specific time intervals. A *merge* operator merges two voices into a single voice. A *map* operator extends the algebra by allowing the application of any ad hoc function that takes one or more voices as input and returns a transformed voice. Two categories of functions are defined natively, allowing temporal and domain transformations respectively.

The algebra itself is a theoretical scientific contribution. In [6], a concrete implementation using XQuery is proposed. This implementation includes an

extraction layer that can process input from music encoding formats such as MusicXML or MEI.

5.2 Representation in an Euclidean space

In [23], an approach to music score similarity detection is explored, where musical notes are represented as weighted points in a two-dimensional Euclidean space. The coordinates of these points represent their onset time and their pitch, while the weight of each point corresponds to the note's duration. This weighting system assigns greater importance to longer notes, reflecting the comparatively large proportion of time they occupy compared to shorter notes. Here, the overarching goal is to establish a dissimilarity metric between music scores, and so a query takes the form of a music score (or an incipit) and a query result is an ordered list of music scores, ordered by how similar they are to the query.

Pitch in this model is encoded using Walter Hewlett's Base-40 system [10], which represents various notations of the same pitch differently (this refers to accidentals, allowing some notes to have multiple ways of being written depending on the scale used). The Base-40 system was developed to facilitate interval computation, recognizing that the same musical interval can have varying nomenclatures in music theory, despite representing the same number of semi-tones, depending on the specific pitches involved. Durations are quantified as divisions of a quarter note; for instance, in a system where 96 divisions represent a quarter note, a sixteenth note would be represented by 24 divisions, and a half note by 192.

Example 8 (Running example – Euclidean space representation). In Figure 8, the Euclidean space representation of the running example (see Figure 2.(a)) is depicted, showcasing both voices. Notably, the two voices are represented together. The small and big dot sizes represent the weight of each note, and correspond respectively to the duration of an eighth note and a half note. For instance, the bottom left dot represents the first half note of the second voice, with its larger size accurately reflecting its longer duration. In this representation, a quarter note duration has been standardized as one division, given that no musical events in this example are shorter than a quarter note.

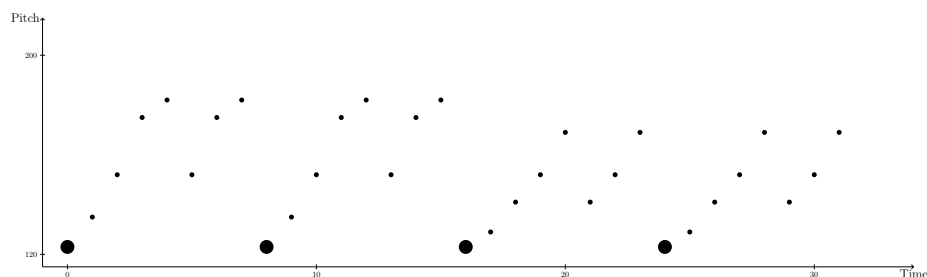


Fig. 8. Representation of the running example in Rainer's work [23]

To assess similarities between music scores or between a music score and a given query (also encoded as a score), the approach employs a normalized version of the Earth Mover’s Distance (EMD). The EMD metaphorically views a set of weighted points as a quantity of earth or mass (where the points themselves describe its silhouette), measuring the minimum work required to transform one distribution of mass into another. This work is quantified as the product of the weight units and the distance over which they are moved. The assignment of points as source or target only alters the sign of the resulting distance.

This method demonstrates potential for original use cases, such as identifying composers of anonymous pieces and detecting similarities in music scores even when variations like additional grace notes or rhythm changes are present.

6 Discussion

In this section, we compare the approaches based on a set of heterogeneous characteristics of interest that we devised from our study of the field. These criteria are not intended to rank the approaches but rather to distinguish them by highlighting their unique features and capabilities in various contexts. We then present a comparative table (Table 1) that encapsulates the positioning of the previously discussed approaches w.r.t. these criteria.

Rendering information and metadata. Data models like ABC, MusicXML, and MEI incorporate rendering details due to their initial nature as exchange format for software score rendering, which adds complexity to content data access [6]. Similarly, metadata such as titles, genres, and composer names, while peripheral to the querying of musical content in its stricter form, are often pertinent for DSL users alongside the content itself. Information such as title or composer name can also be used as score identifiers in query results. Table 1 acknowledges these aspects by outlining how each data model incorporates rendering information and metadata.

Locally encoded information. Some data models, particularly ASCII-based ones, adopt a sheet music-mirroring approach to encoding, where alterations (accidentals) are noted only if they are explicitly marked on the score. This method maintains the implicit nature of alterations as influenced by the key signature or previous accidentals within the same measure, aligning with how music is traditionally written and read. This approach supports specific use cases, like analyses focused on rendering or tonality, where the context of the time signature and visual presentation are crucial.

Conversely, formats like MusicXML explicitly include comprehensive pitch information within each note’s encoding, including chromatic alterations. This is more suitable for melodic pattern matching and comparisons between scores where pitches values (chromatic scale) themselves are concerned (e.g. finding the occurrences of a melody).

Richness and extensiveness of data models. Some formalisms are inherently extensive, offering the capability to incorporate supplementary information directly into the musical score. This includes the ability to add metadata, external links or annotations. This is the case for MEI, which provides extensive support for encoding semantically and structurally rich metadata alongside musical notation. Additionally, it enables the integration of links, external relationships, and annotations associated with the musical content.

Querying languages and polyphonic queries. When the data model relies on a standard encoding format, it enables the utilization of off-the-shelf generic querying languages and tools for extracting information. For instance, the semi-structured modelling of data has led the authors of [7] to use the XQuery language for musical information extraction, and the graph-based modelling has led the authors of [19] to use the Cypher language. On the one hand, using a generic query language enables the use of the available off-the-shelf tools associated with the language (e.g., optimized query engines in DBMSs, dedicated API software layers, etc.). On the other hand, expressing sophisticated extraction queries over highly specific data may be difficult. In the context of symbolic musical data extraction, polyphonic queries (i.e. queries that concern information aligned in multiple voices) are such a kind of sophisticated extraction queries. For instance, as discussed in Subsection 3.2, MusicXML has two ways of representing polyphony, which impacts the way an XQuery query has to be written, especially if such a query is polyphonic. It is in theory possible to write a polyphonic query using XQuery, but this would be rather cumbersome [6]. Likewise, such a query expressed with a Unix-like regular expression, in the context of an ABC library, would be very hard to write. The approaches presented in Sections 4 and 5 are more appropriate for queries spanning multiple staves: the algebra enables the merging of multiple voices, which in turn allows querying the merged result; the Euclidean space representation approach works regardless of how many staves there are in the scores; the graph approach allows pattern searches on multiple staves using graph patterns.

7 Conclusion and perspectives

We have provided an overview of data models that support symbolic, content-based MIR systems. We set aside the goal of exhaustiveness in favor of delineating families of approaches, choosing to delve deeper into selected works within each family. Our contribution lies in the synthesis of existing approaches, organizing them into families and comparing their characteristics, particularly in the discussion section. We have observed that while ABC notation continues to be utilized often as a textual format for musical data manipulation due to its simplicity and readability, semi-structured formats have seen a rapid increase in popularity. These formats have become somewhat of a standard for data interchange among music notation software. This trend underscores the evolving landscape of music information retrieval, where the demand for more sophisticated data handling capabilities continues to grow.

	ASCII based			Document based		Graph		Time-series		
	ABC	Humdrum	GUIDO	MusicXML	MEI	RDF	Muster	ScoreAlg	Euclidian space	
Rendering information										
Excluded						✓	✓	✓	✓	
Included	✓	✓	✓	✓	✓					
Metadata										
Excluded							✓	✓	✓	
Included	✓	✓	✓	✓	✓	✓				
Richness and extensiveness										
					✓	✓				
Handling of accidentals										
Implicit propagation	✓ ¹		✓ ²							
Explicit propagation		✓		✓	✓	✓	✓	✓	✓	
Querying languages										
Standard query language				XQuery	N/M	SPARQL	Cypher			
Non-standard query language									Algebraic operators	
Other means of querying	Unix-like commands	**kern commands	probabilistic similarities				OWL ontol. reas.			similarity measures
Polyphonic querying										
Challenging	✓		✓	✓	✓	✓				
Suitable and illustrated		✓					✓	✓	✓	

¹ can be specified using the `%%propagate-accidentals` option

² cautionary accidentals can be specified

N/M: Not mentioned in the literature

Table 1. Comparative table

Looking forward, MIR systems still grapple with the challenge of developing similarity tools that are adaptable and flexible enough to meet a wide array of use cases, such as data verification, musicological analysis, and pattern-based score retrieval. Additionally, while there exists various works aimed at expressing musical queries — including methods such as Query by Humming, musical contour, rhythm search, and note sequences — there remains a need for truly intuitive methods for formulating complex and comprehensive queries that fully encompass all aspects of sheet music, including polyphony. In this regard, the graph model approach we have presented appears particularly promising.

References

1. Abc notation. <https://abcnotation.com/>
2. Casey, M.A., Veltkamp, R., Goto, M., Leman, M., Rhodes, C., Slaney, M.: Content-based music information retrieval: Current directions and future challenges. Proceedings of the IEEE **96**(4), 668–696 (2008). <https://doi.org/10.1109/JPROC.2008.916370>

3. Downie, J.S.: The scientific evaluation of music information retrieval systems: Foundations and future. *Comput. Music. J.* **28**(2), 12–23 (2004). <https://doi.org/10.1162/014892604323112211>
4. Faget, Z., Rigaux, P., Gross-Amblard, D., Thion-Goasdoué, V.: Modeling synchronized time series. In: *Proc. of the International Database Engineering and Applications Symposium (IDEAS'10)*. pp. 82–89. ACM (2010). <https://doi.org/10.1145/1866480.1866493>
5. Fournier-S'niehotta, R., Rigaux, P., Travers, N.: Querying music notation. In: (TIME'16) *International Symposium on Temporal Representation and Reasoning*. pp. 51–59 (10 2016). <https://doi.org/10.1109/TIME.2016.13>
6. Fournier-S'niehotta, R., Rigaux, P., Travers, N.: Querying XML score databases: XQuery is not enough! In: *Proceeding of International Conference on Music Information Retrieval (ISMIR)*. pp. 723–729 (2016), https://wp.nyu.edu/ismir2016/wp-content/uploads/sites/2294/2016/07/136_Paper.pdf
7. Ganselman, J., Scheunders, P., D'haes, W.: Using XQuery on MusicXML Databases for Musicological Analysis. In: *Proceeding of International Conference on Music Information Retrieval (ISMIR)*. pp. 433–438 (2008), http://ismir2008.ismir.net/papers/ISMIR2008_217.pdf
8. Good, M.: The Virtual Score: Representation, Retrieval, Restoration, chap. "MusicXML for Notation and Analysis", pp. 113–124. W. B. Hewlett and E. Selfridge-Field, MIT Press (2001)
9. group, W.C.: MusicXML Version 4.0, Final Report. <https://w3c.github.io/musicxml/> (2024)
10. Hewlett, W.B.: A Base-40 Number-line Representation of Musical Pitch Notation. *Musikometrika* **4**, 1–14 (1992)
11. Hoos, H.H., Renz, K., Görg, M.: GUIDO/MIR - an experimental musical information retrieval system based on GUIDO music notation. In: *Proceeding of International Conference on Music Information Retrieval (ISMIR)*. pp. 41–50 (2001), <http://ismir2001.ismir.net/pdf/hoos.pdf>
12. Huron, D.: *Beyond MIDI: The Handbook of Musical Codes*, vol. 1, chap. Humdrum and Kern: Selective Feature Encoding. Cambridge: The MIT Press (1997)
13. Jones, J., de Siqueira Braga, D., Tertuliano, K., Kauppinen, T.: Musicowl: the music score ontology. In: *Proc. of the International Conference on Web Intelligence (WI'17)*. p. 1222–1229. Association for Computing Machinery (2017). <https://doi.org/10.1145/3106426.3110325>
14. Music Encoding Initiative (MEI) web site. <http://www.music-encoding.org> (2024), (consult. date)
15. Orio, N.: Music retrieval: A tutorial and review. *Found. Trends Inf. Retr.* **1**(1), 1–90 (2006). <https://doi.org/10.1561/15000000002>
16. Orio, N.: Music indexing and retrieval for multimedia digital libraries. In: *Information Access through Search Engines and Digital Libraries*, pp. 147–170. The Information Retrieval Series, Springer (2008). https://doi.org/10.1007/978-3-540-75134-2_8
17. Prud'hommeaux, E., Seaborne, A.: SPARQL query language for RDF, W3C Recommendation (2006)
18. Rigaux, P., Faget, Z.: A database approach to symbolic music content management. In: *Proc. of the International Symposium on Computer Music Multidisciplinary (CMMR'10)*. *Lecture Notes in Computer Science*, vol. 6684, pp. 303–320. Springer (2010). https://doi.org/10.1007/978-3-642-23126-1_19

19. Rigaux, P., Thion, V.: Topological Querying of Music Scores. *Data and Knowledge Engineering* **153** (2024). <https://doi.org/10.1016/j.datak.2024.102340>, <https://inria.hal.science/hal-04614440>
20. Roland, P.: The Music Encoding Initiative (MEI). In: *Proc. of the International Conference on Musical Applications Using XML*. vol. 1060, pp. 55–59 (2002)
21. Schedl, M., Gómez, E., Urbano, J.: Music information retrieval: Recent developments and applications. *Found. Trends Inf. Retr.* **8**(2-3), 127–261 (2014). <https://doi.org/10.1561/15000000042>
22. Selfridge-Field, E. (ed.): *Beyond MIDI: the handbook of musical codes*. MIT Press, Cambridge, MA, USA (1997)
23. Typke, R., Giannopoulos, P., Veltkamp, R.C., Wiering, F., van Oostrum, R.: Using transportation distances for measuring melodic similarity. In: *Proc. of the International Conference on Music Information Retrieval, (ISMIR'03)* (2003), <http://ismir2003.ismir.net/papers/Typke.pdf>
24. Typke, R., Wiering, F., Veltkamp, R.C.: A survey of music information retrieval systems. In: *Proc. of the International Conference on Music Information Retrieval (ISMIR'05)*. pp. 153–160 (2005), <http://ismir2005.ismir.net/proceedings/1020.pdf>
25. W3C: SOWL 2 Web Ontology Language Document Overview (Second Edition) (2012), <https://www.w3.org/TR/owl2-overview/>
26. W3C: SPARQL Query Language (2013), <http://www.w3.org/TR/sparql11-query/>
27. W3C: RDF overview and documentations (2014), <http://www.w3.org/RDF/>