Topological Querying of Music Scores

Philippe Rigaux, Virginie Thion

Topological Querying of Music Scores

Philippe Rigaux, Virginie Thion[a,b]

[a] *CEDRIC Laboratory, CNAM, philippe.rigaux@cnam.fr, Paris, France*
[b] *Univ. Rennes, CNRS, IRISA, virginie.thion@irisa.fr, Lannion, France*

Abstract

For centuries, *sheet music scores* have been the traditional way to preserve and disseminate Western music works. Nowadays, their content can be encoded in digital formats, making possible to store music score data in digital score libraries (DSL). To supply intelligent services (extracting and analysing relevant information from data), the new generation of DSL has to rely on digital representations of the score content as structured objects apt at being manipulated by high-level operators. In the present paper, we propose the *Muster* model, a graph-based data model for representing the music content of a digital score, and we discuss the querying of such data through graph pattern queries. We then present a proof-of-concept of this approach, which allows storing graph-based representations of music scores in the Neo4j database, and performing musical pattern searches through graph pattern queries with the Cypher query language. A benchmark study, using (real) datasets stemming from the Neuma Digital Score Library, complements this implementation.

Keywords: Symbolic Music Content, Graph Databases, Graph Pattern Queries, Musical patterns

1. Introduction

Music is an essential part of the world's cultural heritage. Even though audio files constitute the main access channel to music works nowadays, music has been preserved and disseminated as *sheet scores* for centuries. For a part of music production, sheet scores have been – and continue to be – the most complete and accurate way to encode the composer's intents, and to faithfully convey these intents to performers.

A sheet score is a complex semiotic object. In a single and compact layout, it combines a symbolic encoding of the music that must be produced

with a sophisticated visual representation aiming at accurately representing the music content.

Music notation is the most elaborate way of describing music at a symbolic level [1]. It has been traditionally used for engraving musical scores [2], but, since the advent of digital encodings such as the **kern representation [3] or XML-based representations (MusicXML [4], its next generation MNX [5], or MEI [6, 7]), music notation can also be seen as a support for music information processing. Large collections of encoded music scores are now available, either as a result of long-running academic efforts [8, 9], or as a result of the generalized production of music scores with editing software applications that encode their documents in one of the above-mentioned formats (e.g., MuseScore [10]). Such collections are examples of *digital libraries* where the music content is described in a structured and well-organized way, apt at supporting sophisticated computer-based operations.

In general, we expect a digital library to be more than a simple repository of digital documents, encoded in a music-agnostic format that obscures their content. Services that rely on an automated processing of the music content, leveraging the digital representation, are required [11, 12]. These services, said to be "intelligent", include the automatic transformation of music scores (e.g., transcription, extraction, transposition of the music content), analysis services (e.g., the classification of scores, segmentation, quality assessment [13, 14]), or the (user-defined) identification of musically significant fragments [12] whose range spans from finding the occurrences of a single rhythmic or melodic motive [15] to finding the occurrences of synchronized melodies over different voices in polyphonic pieces [16]. To supply such intelligent services, and more specifically for the last use case (user-defined musical patterns extraction), we need a digital representation of the *music notation* embedded in encoded scores, as structured objects apt at being manipulated by high-level operators. Traditional data manipulation languages are not fully adapted to the specificity of music representation. Relational languages for instance are clearly not expressive enough, whereas XML manipulation languages quickly become highly complex and dependent on encoding choices [17]. Using a full-fledged programming approach, such as Music21 [18] yields a wide range of powerful operations, but requires programming skills. Moreover, whereas a powerful toolbox like Music21 is well-suited to inspect the content of an individual score, it is far less convenient for managing large corpuses that require a uniform and well-founded representation, as well as collection-level operations.

In the last few years, *graph databases* [19, 20, 21, 22] have started to attract a lot of attention in the data management community. Their basic

purpose is to manage relational data natively modelled as a graph like, e.g., social networks, biological, topological databases or bibliographic databases. Since the content of an encoded score is composed of highly complex information, with connected items (a note follows another one, it belongs to a voice, it also belongs to a measure, etc.), a topological graph-based modelling of the music content seems to be a relevant approach to leverage existing encodings to a true data model apt at supporting ad hoc querying operations. Although a topological point of view for modelling some information of a music score is not new [23], such approach has never been considered for modelling and querying the comprehensive music content of music scores collections.

Contribution. In the present paper, we propose *Muster*, a graph-oriented data model for representing the music content of a score. *Muster* combines, in a single data model, the points of view of the literature that consist in modelling a music score either according to its rhythmic decomposition, or by considering the music content as a flow of musical events. We expose the formal structure of this data model. We then illustrate the graph-pattern querying mechanism, which is made possible through the utilization of a traditional database-oriented query language, grounded in the underlying *Muster* data model. Such a querying mechanism allows to express musical patterns going from simple musical patterns (finding the occurrences of a single rhythmic or melodic motive [15]), to much more complex ones (finding the occurrences of synchronized melodies over different voices in polyphonic pieces [16], mixing rhythmic and melodic constraints over the pattern).

We subsequently examine the feasibility of the approach through a proof-of-concept implementation, which consists in the two following steps.

1. We populate a graph database with the graph-based representation of a (real) music score collection. Concretely, we upload, in a Neo4j graph database, the graph-based representations of (real) music score datasets stemming from the NEUMA Digital Score Library (DSL). The population of the graph database relies on a tool called MUSYPHER, which allows producing the *Muster* graph-based representation of an encoded score and then uploading it in a Neo4j database.
2. We then show the feasibility and the tractability of the querying process by (i) expressing relevant and possibly complex musical patterns over such graph-based collection (the implementation relies on the Cypher concrete graph-pattern query language), and (ii) performing a benchmark study for measuring the cost of retrieving musical patterns, in order to assess the tractability of the querying process.

This article is an extended version of [24].

Organization of the paper. The paper is organized as follows. Section 2 presents a review of the literature focused on recently proposed music score content models. Section 3 introduces the *Muster* model. Section 4 studies the mechanism for querying *Muster* data through graph patterns queries. Section 5 presents the proof-of-concept implementation of the approach. At last, Section 6 presents the results of the benchmark study that measures the cost of evaluating graph pattern queries over a real collection of music scores. Section 7 concludes and draws some perspectives of this work.

2. Related work: models for symbolic music content

The literature proposes some contributions that rely on the modelling of symbolic music content in the form of a graph. We present these approaches in Subsection 2.1. Although such approaches propose a graph modelling of symbolic music content, their final goals lead them to consider only a partial modelling of the music content, and they do not explore the storage or the querying of data. In Subsection 2.2, we present some other approaches, closer to our purpose, that aim at modelling the comprehensive music content, leading to a data model that makes possible to store, manage and query such data.

2.1. Graph-based intermediate representations for symbolic music content

Some approaches of the literature use a graph-based model as an intermediate representation of symbolic music content data. They can be distinguished between the time-independent representations and the time-dependent ones.

Time-independent abstraction of the music content. Over the time-independent graph-based representations, [23] and [25] use a graph structure to model an intermediate abstract representation of a score melody, called the *melody graph*. In [23], the *melody graph* is a directed cyclic graph that models the adjacency of the pitches that compose a melody. This approach translates the score melody into a (time-independent) abstract drawing that offers an aesthetic point of view for discovering musical patterns (e.g., cycles, or pivot notes). Figure 1 is an illustration of this approach over the first 8 measures of "Lady Jane" (The Rolling Stones, 1966). In [25], the melody graph is also a directed (cyclic) graph that models the interval connectivity between the pitches that compose the melody. This *time-independent* signature of the melody is used to define an indexing algorithm

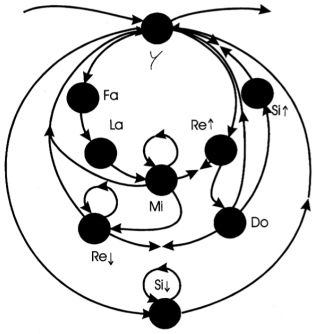


Figure 1: *Melody graph* of the first 8 measures of "Lady Jane" (The Rolling Stones, 1966) [23] (©Karina Zerillo-Cazzaro)

that groups melodies having topological similarities. In the approaches proposed by [23] and [25], the graph models an abstract and time-independent representation of the music content.

Time-dependent representation of music sheet symbols. Another graph-based representation of the symbolic music content is proposed in [26]. Driven by the need to enhance an Optical Music Recognition (OMR) process, the authors of [26] propose an approach that represents as a graph the organisation of the music symbols in a score sheet. In this graph, called *rhythm graph*, each node models the occurrence of a rhythmically relevant symbol (note, rest, or bar line). Each symbol has a duration (a dotted eighth has duration of 3/16, while a bar line has a duration of 0). Nodes are connected either by their order of occurrence inside a voice (a symbol follows another one) or by coincidence edges (symbols of different voices are connected if they share the same onset time). Figure 2 is an example of a rhythm graph associated with a piece of music sheet. In the approach proposed by [26], the representation focuses on the alignment of the music sheet symbols in order to refine music symbols recognition, and models the rhythmic features only.

The above graph-based approaches of the literature do not model the comprehensive music content (they consider only a part of the music content that is needed in their context), and they do not consider the storage and the querying of such information, which is our goal. But some other approaches of the literature share this goal. They are presented in the next subsection.

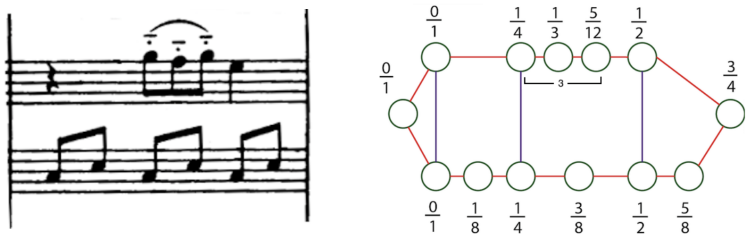


Figure 2: Time-dependent graph-based representation proposed by [26]

2.2. Modelling of the comprehensive music content

Amongst the contributions that model the comprehensive music content, we can distinguish two kinds of approaches: those that structure the score according to a time-wise hierarchical decomposition of the music content (seen as a *vertical* decomposition of the score), and those that consider the music content as synchronized flows of musical events (seen as a *horizontal* decomposition of the score).

Vertical decomposition: tree-like data models. Different types of hierarchical structure underlying a piece of music may be considered [27]. Driven by the need of interoperability between systems and tools, semi-structured models have emerged for representing (Western) music scores. The most widespread ones are MusicXML [28, 29] and MEI [6, 7]. Their common characteristics is to encode the complete content of sheet scores (including graphical specifications that dictate how the notation content has to be visually rendered), and to organize, in the form of a n-ary ordered tree, the musical events (e.g., notes, lyrics) according to the measure they belong to. Figure 3 illustrates the typical organization of data in a semi-structured oriented encoding of a music score.

XML-dedicated query languages (like XPath or XQuery [30]) theoretically have the potential to query such documents. But such score encoding formats intricately blend information pertaining to both the content and its specific rendering. While it is not always obvious to clearly distinguish content from rendering instructions, mixing both concerns leads to an intricate encoding from which selecting the relevant information becomes extremely difficult. In practice, even if simple queries only may really be expressed [30], it turns out that extracting for melodic information from either MusicXML or MEI turns out to be difficult, and more sophisticated extractions (e.g., alignment of melodic information for voices) is almost impossible to express [17].

In order to query music score data, there is a need for a common model

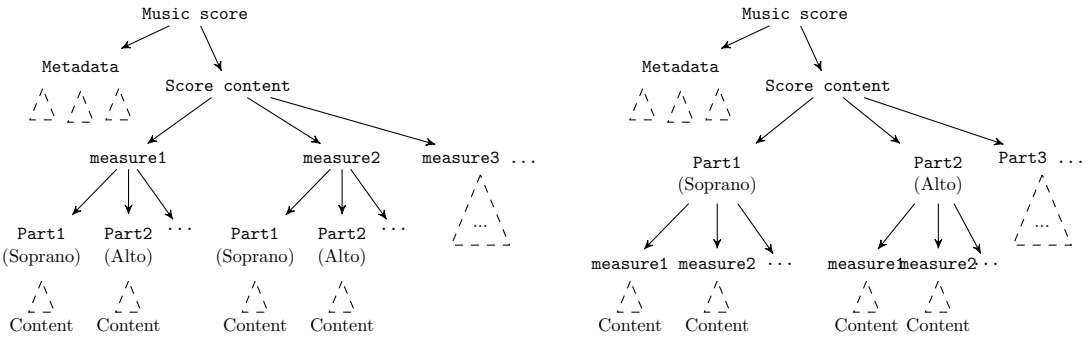
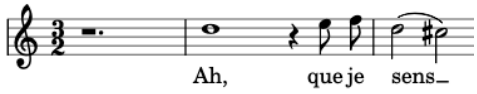


Figure 3: Typical tree-based modelling of a score

that would allow handling scores regardless of the encoding that is used, and apt at supporting the definition of queries dedicated to music content extraction.

Another tree-based data model of the literature is the hierarchical structure proposed in [31], that restricts the modelled information to the music content itself and introduces, w.r.t. the models previously discussed, a finer grain in the hierarchical decomposition of the rhythm. This model is further described in Section 3 as we propose to integrate its components in the *Muster* model.

Horizontal decomposition: models based on time-series. Another perspective consists in seeing the music content as time series of musical events (called "voices"). The *ScoreAlg* algebra, defined in [32], is based on such a perspective. In *ScoreAlg*, a voice is a function whose domain is a time measurement (relevant division of time for timestamping the musical events e.g., a time unit that represents the smallest interval between two musical events) and co-domain is the set of musical events. As an example, Figure 4 shows some functions that model the beginning of a voice, composed of a melody and associated lyrics. Voices are synchronized, over the time, to form a complex score. *ScoreAlg* provides a closed form language based on operators for manipulating music scores information (merging, filtering, etc.) Such a language, powerful but based on complex abstractions, may be difficult to use for the average user who is familiar with music notation but not the digitized encoding of music scores, e.g., a music performer (who retrieves music scores in order to play music with his band), a music analyst (who searches for similar patterns in the parts of a music score), or a musicologist (who conducts a philological study on Rameau's compositions



$$v_{melody}(t) = \begin{cases} \bot, \quad t \in [0, 12[\\ D5_{12}^{20}, \ t \in [12, 20[\\ \bot, \quad t \in [20, 22[\\ E5_{22}^{23}, \ t \in [22, 23[\\ F5_{23}^{24}, \ t \in [23, 24[\\ D5_{24}^{28}, \ t \in [24, 28[\\ C\#5_{28}^{32}, t \in [28, 32[\end{cases} \qquad v_{lyrics}(t) = \begin{cases} \bot, \quad t \in [0, 12[\\ Ah_{12}^{20}, \ t \in [12, 20[\\ \bot, \quad t \in [20, 22[\\ que_{22}^{23}, \ t \in [22, 23[\\ je_{23}^{24}, \ t \in [23, 24[\\ sens_{24}^{32}, t \in [24, 32[\end{cases}$$

Figure 4: Music score content modelled as time series of musical events [32]

including all their variants over time). Another contribution is [33], where functions that model the time-wise distance between the adjacent notes of the score, produce a representation of the melodic progression of a music piece. This *precedence tree structure* serves as a basis for calculating a dissimilarity measure between music pieces. But query languages do not apply to such a representation.

Table 1 summarizes the approaches presented above[1]. In a nutshell, state-of-the-art approaches for modelling the comprehensive music content tend to favor either the sequential aspect or the hierarchical one. Even though a flow-oriented model provides an intuitive understanding (succession of sound events), it obscures the rhythmic structure and missing essential features such as weak/strong beats for instance. On the other side, proposing a hierarchical structure for music content significantly complicates the formulation of a flow-oriented browsing. With the graph-based model proposed in the following, we intend to overcome these limitations. Indeed, the focus is on a *topological* vision of music data, capturing – in a single representation – both perspectives and offering – in terms of expressiveness – the benefits of the two approaches. Our primary goal is to enable the expression of queries in a language that allows navigating a score music content either from the hierarchical structure, or from the flow-oriented perspective, or through a combination of both. In terms of querying, the more expressive is the language, the more expensive is the query evaluation. In light of this,

[1]In order to explicit the positioning of the contribution proposed in this paper, the Muster model appears in Table 1

	[23, 25]	[26]	[6, 7, 28, 29, 30]	[17, 32]	*(Muster)*
Data model family					
semi-structured			✓		
graph-based	✓	✓			*(✓)*
time-series				✓	
Modelled features					
Music content					
partial or abstraction	✓	✓			
comprehensive			✓	✓	*(✓)*
Rendering Information					
comprehensive			✓		
partial		✓			
excluded	✓			✓	*(✓)*
Metadata					
included			✓		
excluded	✓	✓		✓	
partial					*(✓)*
Music content querying					
Querying					
not proposed	✓(*)	✓(*)			
through a standard query language			✓ in [30]		*(✓)*
through a non standard query language				✓	
Content browsing					
vertical			✓		*(✓)*
horizontal				✓	*(✓)*

(*) querying data was not relevant according to the objectives the model was proposed for.

Table 1: Summary of the approaches presented in Section 2

our proposal includes a comprehensive benchmark study designed to assess the feasibility and efficiency of the querying process.

3. *Muster*: a graph-based data model for music score content

Our model, called *Muster* (MUsic Score conTEnt as a gRaph), relies on a representation of digital music documents as structured objects, focused on music content, cleared from side information related to representation purposes. *Muster* captures, in a single and uniform representation, both the hierarchical rhythmic decomposition point of view and the time series point of view. This model is based on the *property graph* model, which is presented below.

3.1. Preliminaries: the property graph model

In a graph database management system, the schema is a graph (nodes are entities and edges are relations between entities), and data is handled through graph-oriented operations and type constructors [19, 20, 21]. Our modelling relies on the *property graph* data model [21, 22], where nodes and edges may embed data as *properties* (key-value pairs).

In terms of vocabulary, we assume the existence of the following pairwise disjoint sets: a set $\mathcal{V}$ of *nodes*, a set $\mathcal{E}$ of edges. We also consider a set *Lab* of labels, a set *Prop* of *properties* (a.k.a. *property keys*), and a set *Val* of *values*.

Definition 3.1 (Property graph). *A property graph $\mathcal{G}$ is a tuple $(V, E, \rho, \lambda, \sigma)$ where (1) $V \in \mathcal{V}$ is a finite set of nodes; (2) $E \in \mathcal{E}$ is a finite set of edges; (3) $\rho : E \to (V \times V)$ is a total function assigning to each edge e an ordered pair of nodes (endpoints, tail and head), where $\rho(e) = (n_1, n_2)$ indicates that e in an edge going from n_1 to n_2; (4) $\lambda_V : V \to \mathcal{P}(Lab)$ is a partial function assigning a set of labels to the nodes of V; and $\lambda_E : V \to Lab$ is a total function assigning a label to each edge of E (without ambiguity, λ refers either to λ_V or to λ_E according to the domain of the assigned element); (5) $\sigma : (V \cup E) \times Prop \to Val$ is a partial function assigning property-value pairs to nodes of V and edges of E, where $\sigma(n, p) = v$ (resp. $\sigma(e, p) = v$) indicates that the node n (resp. edge e) has a property p with the value v.*

Classical notions come with the definition of a property graph $\mathcal{G} = (V, E, \rho, \lambda, \sigma)$.

Definition 3.2 (Path connecting nodes). *A path p in $\mathcal{G}$ is a sequence $n_1\, l_1\, n_2\, l_2\, n_3 \ldots n_{k-1}\, l_{k-1}, n_k$ where $\{n_1, ..., n_k\} \subseteq V$ and $\{l_1, ..., l_{k-1}\} \subseteq Lab$ and $k \geq 1$ and each (n_i, l_i, n_{i+1}) s.t. $i \in \{1, ..., k-1\}$ is an edge of $\mathcal{G}$.*[2] *Path p is said to connect n_1 to n_k, and its* size *is $|p| = k - 1$. Any non empty path that is a subsequence of p, is a* subpath *of p. Path p is a* cycle *iff $k \geq 2$ and $n_1 = n_k$. Path p is is said to be* cyclic *iff one of its subpaths is a cycle, otherwise p is said to be* acyclic*.*

The preliminary notions are above presented, we can now address the proposed data model.

[2] (n_i, l_i, n_{i+1}) is an edge of $\mathcal{G}$ means that there is $e \in E$ such that $\rho(e) = (n_i, n_{i+1})$ and $\lambda(e) = l_i$.

3.2. The Muster *model*

Intuitively, the music content is partly modelled by a mapping of a structured temporal domain to a set of *musical facts*. The temporal domain takes the form of a hierarchical structure, called *rhythmic tree*, which recursively divides a finite time range into disjoint intervals. The combination of such an interval and a musical fact yields a *musical event*. Musical *voices* are defined as sequences of musical events. In the following subsection, we formally define this representation. Let us begin with the notion of musical fact.

3.2.1. The domain of musical facts

The following concepts will be illustrated with an excerpt of the German anthem, *Das Lied der Deutschen*, composed by *Joseph Haydn* in 1797 [34], shown on Figure 5.

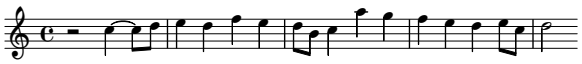


Figure 5: Firts notes of *Das Lied der Deutschen*, of *Joseph Haydn* (1797).

Let us first consider the notion of (atomic) musical fact. Several fact domains can be envisaged to describe a musical object. Due to space limitation, we will focus here on the main one, *sounds*. A sound can be characterized by many properties, including intensity, timbre and frequency. In the language of music notation, a finite set of frequencies, or *pitches*, is used to refer to the sounds usable in a musical piece. We follow the designation of the International Standards Organization (ISO) for enumerating the pitches [35]. Each pitch is referred to by a pitch class P (a letter A, B, C, D, E, F, or G), an index O in $[1, 7]$, and an optional accidental a in $\{\sharp, \flat\}$ (*sharp* and *flat* accidentals). We will thus model a sound as a triple $\{class : P, octave : O, accidental : a\}$, and represent it as a symbol of the form $P[a]O$. The addition of other relevant properties (e.g., intensity) is trivial.

Graphically (i.e., in music scores), frequency levels are materialized by groups of horizontal lines (called staves) and pitches are represented by black or white heads vertically positioned on staves. The first pitch in the score of Figure 5 is a $C5$, followed by a $D5$, an $E5$, etc. Music is also made of silences (or *rests*), and we thus add the *rest symbol* r to the domain. The German anthem starts with a rest, graphically represented by a small black rectangle.

Finally, in conventional music notation, sounds can be "tied" (graphically represented as curves over the heads, such as in the first measure of Figure 5). We add the *continuation symbol* _ to our domain to represent ties. We obtain the domain of musical facts.

Definition 3.3 (Domain of (atomic) musical facts). *The domain $\mathcal{F}$ of musical facts consists of: (1) the set of sound facts $\{P[a]O | P \in \{A, B, C, D, E, F, G\}$ and $a \in \{\sharp, \flat\}$ and $O \in [1, 7]\}$, (2) the rest fact, denoted by* `r`*, (3) the continuation fact, denoted by _, which allows modelling the ties or sound continuations (a single sound whose representation requires several symbols).*

As an example, A♯4 is a musical fact, modelling the pitch of a A at the octave 4, altered by an accidental ♯.

Modelling facts. In the *Muster* model, facts are modelled by nodes with properties. Figure 6.(a) is a fact node that models an instance of an A♯4 note. The node has three properties, namely `class` (this property has value `A` for the node), `octave` (having value `4`) and `accid` (having value `sharp`, which models a ♯ alteration). In the following, such a node will be depicted by a compact representation embedding the core property values in the node itself (Figure 6.(b)).

(a) node with its properties (b) compact representation

Figure 6: Node modelling a musical fact

3.2.2. Temporal organization of facts

Music is hierarchically organized. Different types of hierarchical structure underlying a piece of music may be considered [27]. In the following, we focus on the modelling of the *metrical structure*, based on the hierarchy of beats defined in the score, as proposed in [31].

A music piece is a temporal organization of sounds inside a bounded time range. Musical facts fall on a set of positions that result from a recursive decomposition of temporal intervals, yielding a rhythmic organization which is inherently hierarchical. In Western music notation, the recursive division of a music piece in measures, beats, sub-beats, etc., generates a hierarchy of pulses called *metrical structure*. It can be represented as a *rhythmic tree*

based on the following rules: i) the time range of the piece in divided in equal-sized measures, ii) each measure is recursively divided according to metric rules, and iii) each leaf of the tree corresponds to a musical fact that occurs during the sub-interval defined by the position of the leaf.

Figure 7 illustrates such a structure: a rhythmic tree that temporally organizes the musical facts. The part with blue nodes represents the rhythmic tree. The root (s) corresponds to the whole music piece. The level under the root is made of *measure nodes*. Each measure is itself decomposed in beats, quavers, etc., according to the required temporal decomposition. The first measure for instance is decomposed in four beats, the latest being itself decomposed in two quavers. The duration and temporal position of each node can easily be determined from the rhythmic tree. Indeed, if N is a node covering an interval $I = [\alpha, \beta[$, with subtrees $(N_1, \cdots, N_n)$, the interval $itv_I(N_i)$ covered by each N_i is $[\alpha + (i-1) \times s, \alpha + i \times s[$ where $s = \frac{\beta - \alpha}{n}$ is the size of each sub-interval (see [31] for details).

We adopt the following convention for temporal values: the duration of a measure is 1, it extends over interval $[0, 1[$, and the music piece range is $I = [0, n[$, where n is the number of measures. Both the duration and interval of a node result from the recursive division represented by the tree. If the meter is 4/4, the duration of a half note for instance is $\frac{1}{2}$, the duration of a beat is $\frac{1}{4}$, etc.

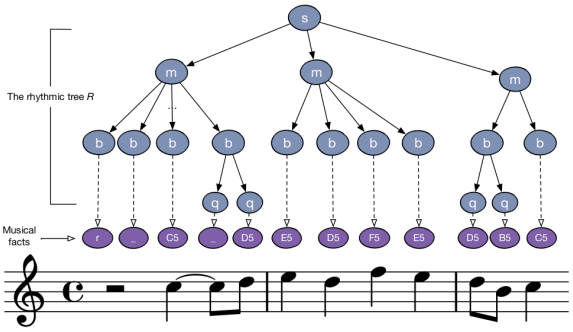


Figure 7: Modelling the German anthem, with its rhythmic tree and associated facts

Definition 3.4 (Voice). *A* voice *is a pair $(R, \mathcal{M})$, where R is a rhythmic tree and, $\mathcal{M}$ is mapping from the leaves of R to the set of musical facts $\mathcal{F}$.*

On Figure 7, the dotted edges represent the mapping $\mathcal{M}$. Such a mapping leads to the notion of *musical event*.

3.2.3. Musical events

Each leaf l of the rhythmic tree is linked to a musical fact $\mathcal{M}(l)$. The first beat of the first measure is a *rest* fact, the second is a *continuation* fact (extending the previous rest), the third is a *sound* fact (a $C5$). The fourth beat is divided in two facts: a continuation of the previous fact ($C5$), and a $D5$ fact. Since each leaf l of a rhythmic tree covers an interval $itv(l)$, one obtains *musical events* as pairs $(itv(l), \mathcal{M}(l))$, associating duration and fact. Such a sequential representation is commonly found in music notation.

Definition 3.5 (Musical event). *Let I be a time range, and $V = (R, \mathcal{M})$ be a voice. If l is a leaf of R, then the pair $(itv_I(l), \mathcal{M}(l))$ is a* musical event.

Turning back to Figure 7, the first event is (I_1, r), with $I_1 = [0, \frac{1}{4}[$ (the first beat of the first measure is a rest). The second event is $(I_2, \_)$, with $I_2 = [\frac{1}{4}, \frac{2}{4}[$ (we recall that it represents a continuation of the rest for one beat). The third event is $([\frac{2}{4}, \frac{3}{4}[, C5)$, the fourth is $([\frac{3}{4}, \frac{7}{8}[, \_)$, etc.

Definition 3.6 (Onset and offset). *Let $(itv_I(l), \mathcal{M}(l))$ be a musical event, where $itv_I(l) = [\alpha, \beta[$. Then α (resp. β) is said to be the onset (resp. offset) of the event. By extension, α (resp. β) is said to be the onset (resp. offset) of the fact $\mathcal{M}(l)$.*

Any instrument part that appears in a music score can be modelled by a set of voices (typically several voices for each polyphonic instrument), and each voice can be structured according to a rhythmic tree.

3.2.4. Modelling the music content *of a voice as a graph*

Intuitively, the *Muster* graph-based modelling of a score consists in representing, in a single data model, the two following points of views: (i) the rhythmic decomposition (see Figure 7) and (ii) the flow of events for organizing the music content as synchronized time series. Such a representation observes the following principles.

Principle 1. The rhythmic tree and the musical events (see Figure 7) are modelled as a graph (see Figure 8). In such a representation, the root of the rhythmic tree becomes a node called *top node* in the graph model. The leaves become nodes called *event nodes* in the graph model. The facts become nodes called *fact nodes* in the graph model (see Section 3.2.1 for the modelling of a musical fact as a node embedding properties). The other

potential intermediate levels of the rhythmic tree depend on the considered hierarchical structure.

More formally, given a voice $V = (R, \mathcal{M})$ as defined in Definition 3.4, the graph-based representation of V is a graph $\mathcal{G} = (V\,, E\,,\, \rho\,, \lambda,\, \sigma)$ such that there is an injective function h that maps the elements (nodes, edges and facts) of V into the elements of $\mathcal{G}$. As the matching expressed by h is trivial, we do not go into dispensable formalization details.

In the example of Figure 8, for the sake of simplicity, the hierarchical structure is restricted to three levels: a *top* level, a *measure* level, and the *event* level. Considering any other tree-based decomposition like e.g., a finer-grained decomposition of the rhythm [31], or a tree grammar [27], are straightforward extensions that just consist in modelling another tree -possibly containing more intermediate levels- leading to the event nodes.

Principle 2. If n_1 and n_2 are two nodes of the initial rhythmic tree such that n_2 is the immediate following sibling of n_1, then $h(n_1)$ and $h(n_2)$ are linked by a directed edge going from $h(n_1)$ to $h(n_2)$.

Principle 3. If l_1 and l_2 are two leaves of the initial rhythmic tree such that l_2 is the following sibling of l_1, then $h(l_1)$ and $h(l_2)$, which are event nodes, are linked by a directed edge r, and r embeds a property duration whose value is the size of $itv_I(l_1)$[3].

By construction, there is only one event node that has no ancestor in the flow of events. This event node is said to be the *first event node* of the voice.

Principle 1 models the point of view that consists in organizing a voice according to its rhythmic decomposition. Principles 2 and 3 model the point of view that consists in seeing a voice as a flow of musical events. The edges of Principles 2 and 3 implement a topological representation of the flow of events, and maintain the order carried by the (intrinsically ordered) rhythmic tree. The duration property attached with the edges connecting two adjacent event nodes makes the notion of *time-distance between two events* compliant with the notion of *distance between two (event) nodes in a weighted graph*, paving the way to a topological-oriented browsing of the flow of events.

[3]For the completeness of the representation, each flow of event ends by a supplementary special "ending" event that models the end of the series, and this special event is the ongoing node of a last edge that carries the duration of the last musical event.

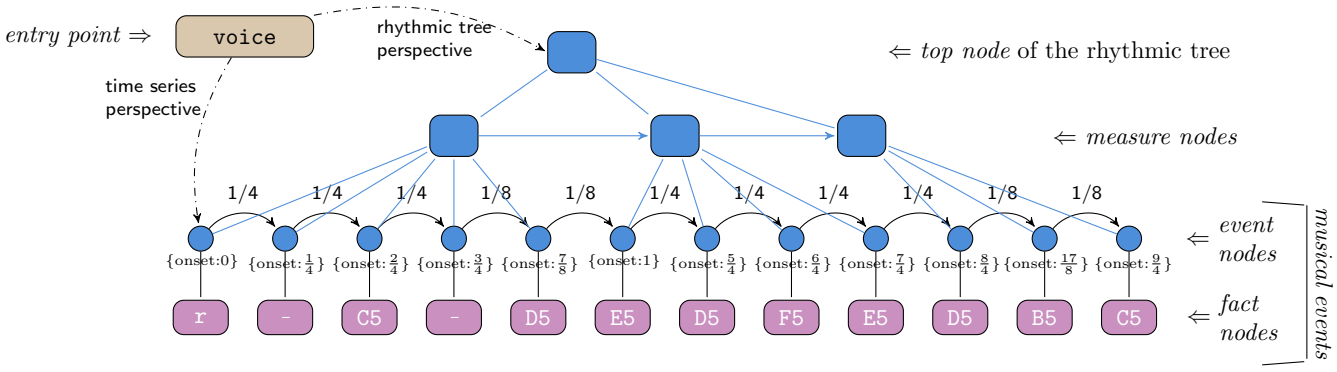


Figure 8: Graph modelling of a voice

Principle 4. And the last principle: the content of a voice may be reached, from an *entry point* of the modelling, either from the root of the rhythmic tree (said to be the *rhythmic tree perspective*) or from its first event (said to be the *time series perspective*). Any relevant attribute of a voice (e.g., the instrument or the name of the voice) may be attached to the *entry point* of the voice via node properties.

In the following, we restrict the rhythmic tree to the modelling of three levels: the *top* (root of the rhythmic tree), the *measure* and the *event* levels. One obtains, from the voice of Figure 7, the graph of Figure 8 (for a compact representation, an edge duration property of the form $\{duration : \frac{1}{X}\}$ is denoted by $\frac{1}{X}$ only).

Remark 3.1 (Embarking additional information for usability). *In practice, in order to improve the usability of data, nodes and edges may be labelled (set Lab associated with the function λ in Definition 3.1). For instance, the nodes modelling measures may be labelled by a label* `Measure`*, the entry point node of the voice may be labelled by a label* `Voice`*, the event nodes may be labelled by the label* `Event`*, and the fact nodes may be labelled by the label* `Fact`*. Additional labels may also specify the nature of a fact, e.g.,* `Note` *or* `Rest`*. Edges may be labelled by the kind of relationship they model (e.g., the edges that model the time series of events may be labelled by the label* `Next`*). An* event node *e embarks its onset, which is a duplicate information as the onset of e the length of the weighted path going from the first event node to e on he flow of events, where the weighted is the duration associated with the edges connecting the event nodes (as shown in Figure 8), its offset (also a duplication information), or any relevant information like*

its duration (duplicate information again), for a "direct" access such information in queries.

3.2.5. Modelling polyphonic music

A *polyphonic* piece of music is composed of a set $\{V_1, V_2, \cdots, V_n\}$ of voices (e.g., Figure 9 on page 18). The modelling a polyphonic piece relies on the modelling of its voices, attached to a common node that model the whole piece (an example of such an object is given in the next section). The voices of a piece of music always have, intrinsically, the same organisation of rhythmic tree, from the root to the measures (synchronisation of the voices in the music notation itself). Then, the next levels of the rhythmic trees differs according to the content of the voices. This leaves two choices for modelling polyphonic music.
(1- Duplicating the rhythmic trees.) We can choose to model a polyphonic piece as a set of voices, with the constraint that all the upper level (measures) of their respective rhythmic trees are similar, but the common part of the rhythmic trees remain duplicated.
(2- Factorizing the common parts of the rhythmic trees.) We can also choose to "factorize" these upper levels, representing the common sequence of measures, and then managing individual subtrees for each measure and each voice.

We adopt the second option, given its gain in space and its ability to easily infer synchronisation properties. As an illustration, the graph of Figure 10 (page 19) models two voices that share the rhythmic structure.

Polyphonic pieces also model the organization of the voices, which takes the form of a hierarchy that structures the voices according to the music parts composing the score, leading to the entry point nodes of the voices. On Figure 10, such a hierarchy is composed of the brown nodes (at the left of the graph). The hierarchy may be as complex as needed: for instance, a score may be composed of several parts (e.g., instruments of an orchestra), and some parts may themselves be internally polyphonic (this is typically the case for keyboards).

Size of a graph-based representation. The size of the *Muster* graph representation of a score depends on the number of measures, the number of voices, and of course the number of musical events that compose the score. This size varies from a collection to another. For instance, the average size of the graph representation of a Bach choral stemming from the collection of the Neuma platform [36], which models a rather long polyphonic music piece, is around 700 nodes (see Section 6 for details). However, the average

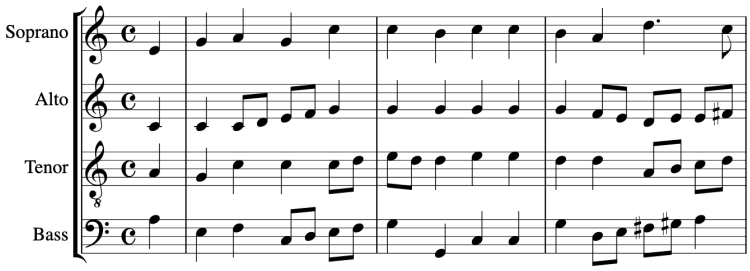


Figure 9: Excerpt of the J.S. Bach Cantata BWV111, 6th movement

size of the graph representation of a Joseph Mahé's score[4] stemming from a collection of the SKRID platform [38], which models a rather short monophonic music pieces, is around 100 nodes.

The *Muster* graph-based data model for music scores provides an expressive and intuitive modelling, which paves the way towards the storage and the querying of such data though graph-pattern queries. This subject is discussed in the next section.

4. Graph patterns for music scores

We now consider the problem of querying the graph-based modelling of an encoded score that respects the *Muster* model proposed in the previous section. We use as an example the polyphonic music score of the Figure 9, called BWV111 in the following. This score contains four voices: *Soprano*, *Alto*, *Tenor* and *Bass*. The four first measures appear on Figure 9, numbered from *0* to *3*. Figure 10 presents the *Muster* graph-based representation of BWV111, restricted to the beginning of the *Soprano* and *Alto* voices. This graph is denoted by $\mathcal{G}_{\text{BWV111}}$ in the following.

Before considering the exploration of such data thought graph pattern queries, we recall the basic notion of property graph pattern. Formally, a *graph pattern* is classically defined as a graph where variables and conditions can occur [39, 21].

[4]Joseph Mahé (1760-1831) [37] was a clergyman, considered as the first collector of Traditional Breton music of Brittany, in France. Its collections are widely studied by the musicologists who study the Traditional Breton music.

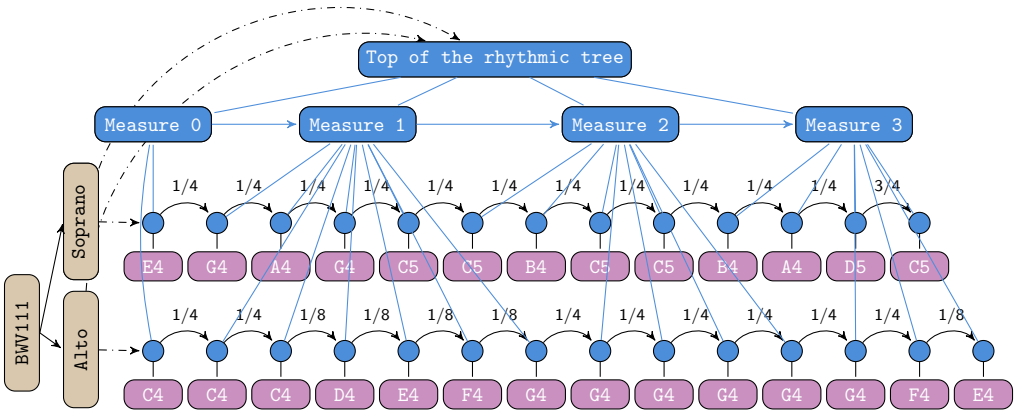


Figure 10: ($\mathcal{G}_{\text{BWV111}}$) Graph-based modelling of BWV111

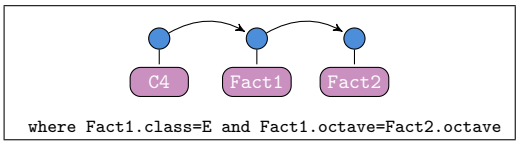


Figure 11: Two events following a note $C4$

Definition 4.1 (Graph pattern syntax). *Let Var_{nodes} and Var_{lab} be distinct sets of node variables and label variables respectively. A* graph pattern query *is a tuple of the form $(V, E, \rho, \lambda, \sigma, Cond_n, Cond_e)$ where variables can occur on nodes and on edge labels,*[5] *and $Cond_n$ (resp. $Cond_e$) denotes Boolean conditions over property values of the elements of V (resp. E) (e.g., if f_1 is a node variable, then a condition could be $f_1.class = E \wedge f_1.octave = 4$, which indicates in our context that the node mapping in f_1 must be a $E4$ fact note).*

Intuitively, a graph pattern defines a shape that has to be found in data.

In its simplest form, a graph pattern denotes a *path*. Figure 11, Figure 12 and Figure 13 are some examples of such patterns. In these patterns, nodes and edges are free variables, and conditions may be attached to the elements of the pattern. The pattern of Figure 11 aims at retrieving the occurrences

[5] meaning that $(V, E, \rho, \lambda, \sigma)$ is a property graph such that $V \in \mathcal{V} \cup Var_{nodes}$ and $\lambda : (V \cup E) \to Lab \cup Var_{lab}$.

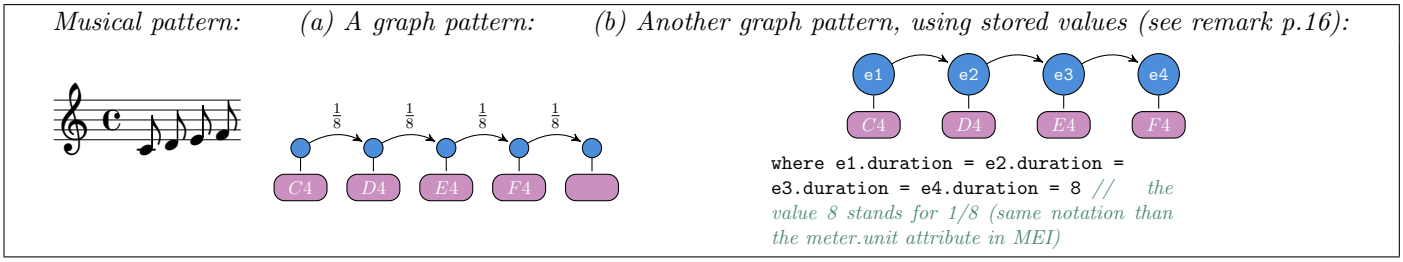


Figure 12: Sequence of notes C4-D4-E4-F4

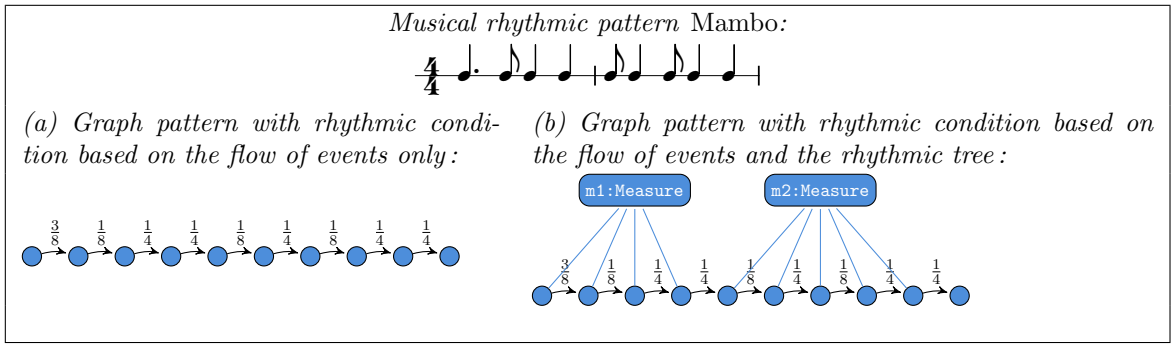


Figure 13: *Mambo* rhythmic pattern

of two notes following a $C4$ one. In this pattern, Fact1 and Fact2 are node variables. A condition is attached with Fact1 that must be a E note. Another condition is attached to the pattern concerning the adequacy of the octaves of Fact1 and Fact2 (expressed by the condition Fact1.octave=Fact2.octave). There is no other condition over the notes (for instance, there is no condition over the duration of each of the three notes). The pattern of Figure 12 aims at retrieving the sequences of eighth notes $C4$-$D4$-$E4$-$F4$ that appear in data. The pattern contains conditions over the pitch class and the octave, and also over the events duration. The patterns of Figure 11 and Figure 12 rely on the perspective seeing a voice as a flow of events, independently from its rhythmic tree (no matter the measures the notes belong to). Adopting instead the rhythmic tree perspective brings the ability to consider the rhythmic status of an event, i.e., its position in the measure it belongs to. This is particularly important when weak/strong rhythmic positions have to be considered. The two points of view can be accommodated in graph pattern query since the data model models the two points of view. Figure 13 shows two graph patterns aiming at retrieving a sequence of musical events corresponding to a mambo rhythm [40]. Pattern (a) is a sequence of musical

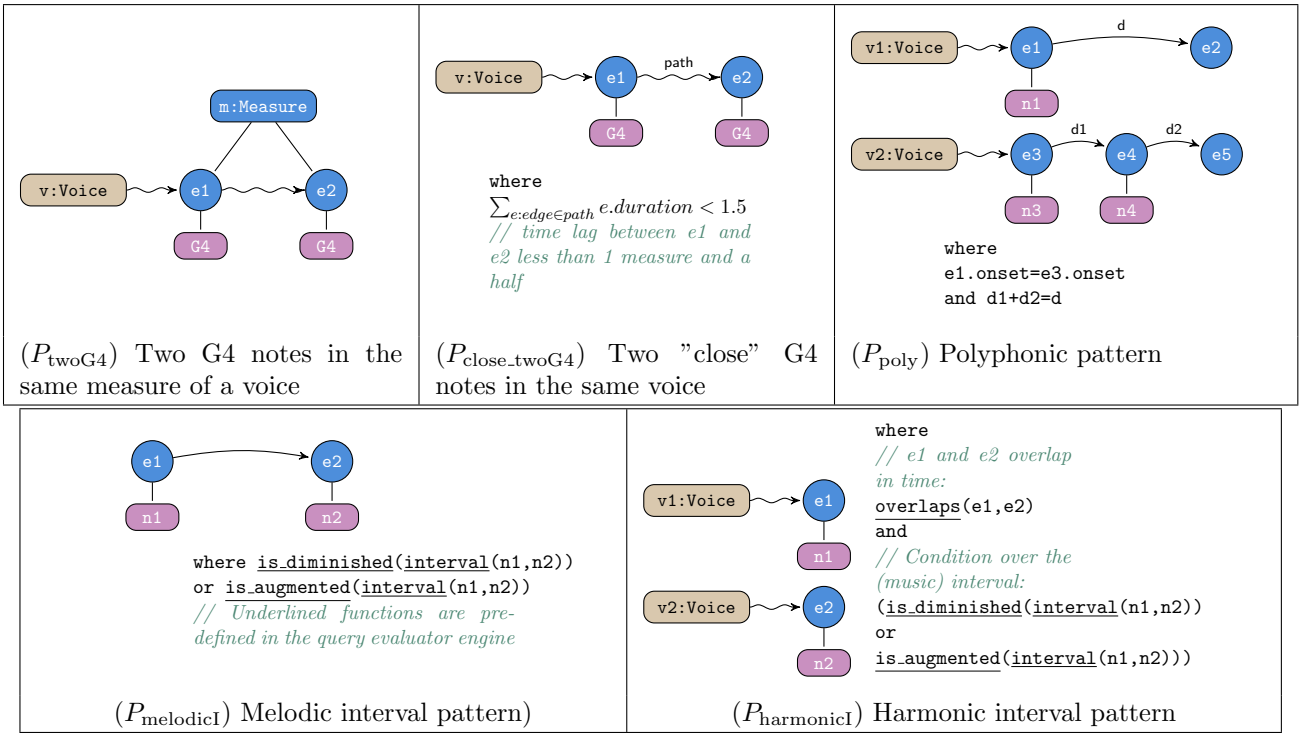


Figure 14: Examples of "complex" graph patterns

events (node and edge variables) for which the rhythmic properties of the sequence is imposed (values of the *duration* property). This version only considers the flow of events, and ignores weak/strong beast in particular, and in general the rhythmic organisation of the score. Pattern (b), instead, constrains the position of the events thanks to its specification based on the rhythmic tree.

The language turns out to be highly expressive by allowing the specification of flexible constraints. For instance, the polyphonic pattern P_{twoG4} (Figure 14) aims at retrieving two $G4$ notes that belong to the same measure in the same voice (but the notes are not necessary adjacent). The wavy relationship between the events denotes an arbitrary path. An interesting declination, shown in $P_{\text{close\_twoG4}}$, consists in considering the time-wise distance between the events, expressing a notion of user-defined *distance between events* compliant with the weighted graph topological distance (the weight is the value of the `distance` property of the edges). In the pattern $P_{\text{close\_twoG4}}$, `e1` and `e2` must be "close", where the notion of closeness defined

here means less distant than a time lag of one measure and a half.

Some other complex graph patterns may address several voices. The polyphonic pattern is P_{poly} (Figure 14) aims at retrieving the occurrence of two notes (n3 and n4 on the figure) played during the time of another one (n1 on the figure) in another voice.

Another quite useful feature, when it comes to manipulate music content, is the definition of specialized functions. As an illustration, let us imagine a Dinterval() function that implements the notion of (diatonic) *interval* between two pitches, while other functions allow checking the properties of an interval, e.g., the Boolean functions is_diminished or is_augmented. Let us also consider a function *overlaps* which, given two events, states if the events overlap in time, according to their *time intervals* (see Section 3). Patterns P_{melodicI} and $P_{\text{harmonicI}}$ aim at retrieving melodic and harmonic intervals respectively, for which the interval is either diminished or augmented.

Let us now turn to the formal definition of a graph pattern semantics (in other words, the result of the evaluation of a graph pattern query over some data).

Graph pattern semantics. Formally, the interpretation $[\![P]\!]_{\mathcal{G}}$ of a graph pattern query P over a graph database $\mathcal{G}$ consists in finding all the subgraphs of $\mathcal{G}$ that are homomorphic to P [39].

Definition 4.2 (Graph pattern interpretation over a graph). *The answer $[\![P]\!]_{\mathcal{G}}$ of a pattern query P over a graph $\mathcal{G}$ is the set of subgraphs $\{g \in \mathcal{P}(\mathcal{G}) \mid g$ "matches" $P\}$. A subgraph g "matches" P iff there is a homomorphism h from the nodes and labels variables of P to $\mathcal{G}$ and each node (resp. label) $h(v)$ satisfies its associated conditions $Cond_n$ (resp. $Cond_e$).*

As an example, we consider the evaluation of the graph pattern P_{twoG4} of Figure 14 over the graph $\mathcal{G}_{\text{BWV111}}$. The pattern matches into several subgraphs of $\mathcal{G}_{\text{BWV111}}$, including a subgraph that contains the first and third notes of the measure 1 for the voice *Soprano*, and another subgraph that contains the first and second notes of the measure 2 for the voice *Alto* (see Figure 15). Then, these two subgraphs of $\mathcal{G}_{\text{BWV111}}$ belong to $[\![P_{\text{twoG4}}]\!]_{\mathcal{G}_{\text{BWV111}}}$. (Note that other subgraphs that match P_{twoG4} can be found in $\mathcal{G}_{\text{BWV111}}$.)

By extension, retrieving the occurrences of a musical pattern in a collection of graphs consists in searching for the pattern in each graph of the collection.

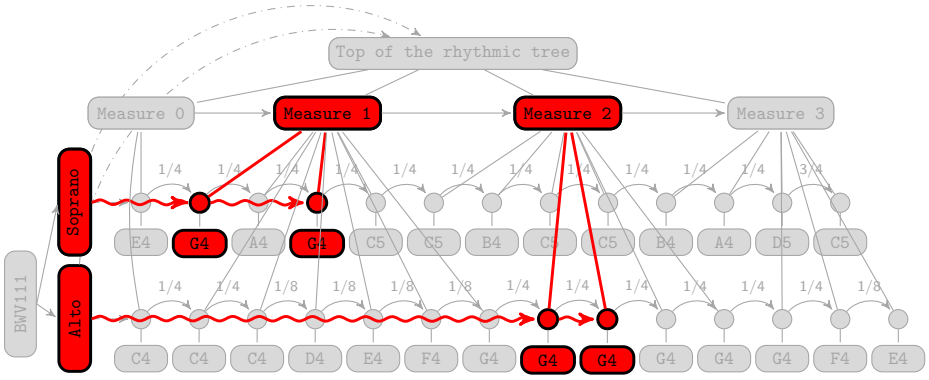


Figure 15: ($\mathcal{G}_{\text{BWV111}}$) Two subgraphs that belong to $[\![P_{\text{twoG4}}]\!]_{\mathcal{G}_{\text{BWV111}}}$

Definition 4.3 (Graph pattern interpretation over a collection). *Let $\mathcal{C}$ = be a collection of graphs. The answer $[\![P]\!]_{\mathcal{C}}$ of a pattern query P over the collection $\mathcal{C}$ is $\cup_{\mathcal{G}\in\mathcal{C}}[\![P]\!]_{\mathcal{G}}$.*

5. Implementation

We implemented a proof of concept of the theoretical framework proposed above. The implementation relies on a software application called MUSYPHER[6], which allows to process a MEI (XML-based) file in order to transfer its music content into a Neo4j graph database that respects the *Muster* model (see Section 3). The stored graph-based versions of the music scores can then be processed by graph-based query tools, including the Neo4j Desktop which proposes an graphical user interface for querying data *via* the Cypher [41] graph pattern query language. The architecture that supports the benchmark is composed of the same building blocks than the architecture of the SKRID platform proposed in Figure 23 (page 36).

In order to illustrate the implementation, we consider again the music score of Figure 9 (page 18), whose digitized content was initially available in the MEI format in the J.S. Bach collection of the NEUMA platform [36]. This music score was processed by the MUSYPHER tool for creating its graph-based representation, and this graph was loaded into a Neo4j database (see

[6] MUSYPHER is a Java application, developed for our project. It is based on a DOM parsing of the initial XML encoded score. Its current version is available at `https://www-shaman.irisa.fr/musypher`

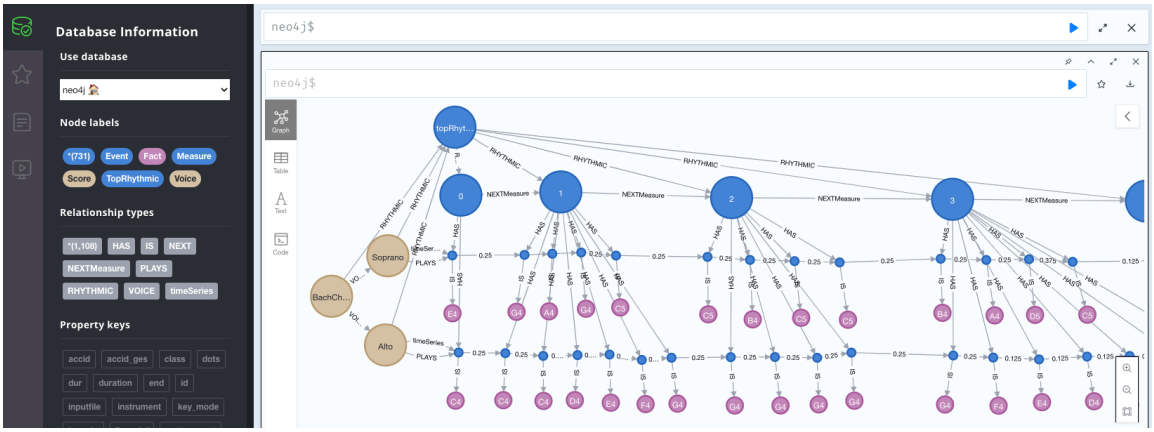


Figure 16: $\mathcal{G}_{\text{BWV111}}$: *Neo4j Desktop* screenshot

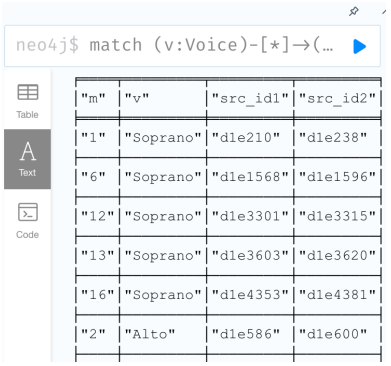
the screenshot of the Neo4j desktop on Figure 16).

Let us return to the graph pattern P_{twoG4} of Figure 14.(a) (page 21). In the Cypher language, a query based on P_{twoG4} could be expressed by Query Q_{twoG4} given in Figure 17. The shape of the graph pattern is declared in the MATCH clause (lines 1 to 6). In the Cypher formalism, the graphic symbol ( ) denotes a node, which may contain information of the form variable:Type, and the symbol −[expr]−> defines the form of a connection (expr is either an edge label, or a regular expression denoting a path). Expressions of the form {attr1:value1, attr2:value2, ...} are additional conditions over properties. The RETURN clause (lines 7 to 9) contains the elements of interest that should be rendered as a result. Here, Query Q_{twoG4} retrieves the measure number, the voice name, and the initial MEI identifier of the elements, considered as a relevant information in our context (for the identification of the elements in the MEI sources file).

A part of the result of the evaluation of Q_{twoG4} over $\mathcal{G}_{\text{BWV111}}$ (the evaluation is made by the query engine of Neo4j), is given in Figure 17 (screenshot of the *Neo4j Desktop*, at the right of the query). Amongst the retrieved subgraphs that match the pattern, we can see, in the first and the sixth lines, the subgraphs previously identified during the evaluation of P_{twoG4} over $\mathcal{G}_{\text{BWV111}}$ (see Figure 15 and the associated discussion in page 22).

Another example is the query Q_{poly} of Figure 18 (page 25), which expresses in the Cypher language the graph pattern P_{poly} of Figure 14 (see page 21). The first line of the result shows that, at timestamp 0.5 (2/4) of the score, the voice *Soprano* plays a note $A4$ (second note of the measure 1 for this voice) while the voice *Alto* plays a note $C4$ followed by a note $D4$

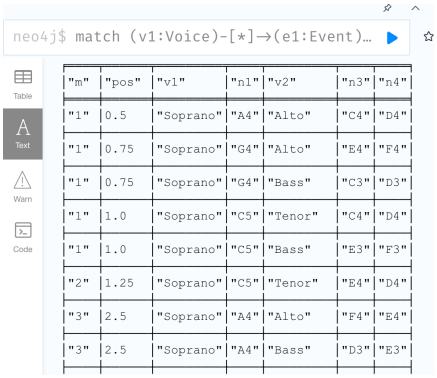
```
1  MATCH
2   (v:Voice)−[∗]−>(e1:Event)−[∗]−>(e2:Event), // e1
    belongs to a voice. e2 belongs to the same voice and
    appears ‘‘somewhere” after e1.
3   (m:Measure)−−(e1), // e1 belongs to a measure (m)
4   (m)−−(e2), // e2 belongs to the same measure (m)
5   (e1)−−({class:'g',octave:4}), // e1 is a G4
6   (e2)−−({class:'g',octave:4}) // e2 is a G4
7  RETURN
8   m.number AS m, v.name AS v,
9   e1.id AS src_id1, e2.id AS src_id2
```



"m"	"v"	"src_id1"	"src_id2"
"1"	"Soprano"	"d1e210"	"d1e238"
"6"	"Soprano"	"d1e1568"	"d1e1596"
"12"	"Soprano"	"d1e3301"	"d1e3315"
"13"	"Soprano"	"d1e3603"	"d1e3620"
"16"	"Soprano"	"d1e4353"	"d1e4381"
"2"	"Alto"	"d1e586"	"d1e600"

Figure 17: Query Q_{twoG4} and its result over $\mathcal{G}_{\text{BWV111}}$

```
1  MATCH
2    (v1:Voice)−[∗]−>(e1:Event)−[r:NEXT]−>(e2:Event),
3    (v2:Voice)−[∗]−>(e3:Event)−[r1:NEXT]−>(e4:Event),
4    (e4)−[r2:NEXT]−>(e5:Event), (m:Measure)−−(e1),
5    (e1)−−(n1:Fact{type:'note'}),
6    (e3)−−(n3:Fact{type:'note'}),
7    (e4)−−(n4:Fact{type:'note'})
8   WHERE
9    e1.onset=e3.onset AND // onset of time intervals
10   r1.duration+r2.duration = r.duration // durations
11  RETURN
12   m.number AS m, e1.onset AS pos,
13   v1.name AS v1, n1.name AS n1, v2.name AS v2,
14   n3.name AS n3, n4.name AS n4
```



"m"	"pos"	"v1"	"n1"	"v2"	"n3"	"n4"
"1"	0.5	"Soprano"	"A4"	"Alto"	"C4"	"D4"
"1"	0.75	"Soprano"	"G4"	"Alto"	"E4"	"F4"
"1"	0.75	"Soprano"	"G4"	"Bass"	"C3"	"D3"
"1"	1.0	"Soprano"	"C5"	"Tenor"	"C4"	"D4"
"1"	1.0	"Soprano"	"C5"	"Bass"	"E3"	"F3"
"2"	1.25	"Soprano"	"C5"	"Tenor"	"E4"	"D4"
"3"	2.5	"Soprano"	"A4"	"Alto"	"F4"	"E4"
"3"	2.5	"Soprano"	"A4"	"Bass"	"D3"	"E3"

Figure 18: Query Q_{poly} and its result

(second and third notes of measure 1 for this voice).

By extension, retrieving the occurrences of a musical pattern in a collection of *Muster* graph consists in evaluating the pattern query over a database that is populated with the collection.

6. Benchmark: the cost of querying

We now discuss the cost of topologically querying data in a graph database populated by *Muster* representations of music scores. We perform a benchmark, considering the evaluation cost of twenty-two topological musical pattern queries, expressed in different query sizes, executed over eight

databases storing an increasing volume of data[7]. We then discuss the results of this benchmark.

6.1. The benchmark queries

For the sake of clarity, we distinguish the musical patterns according to two main criteria: (1) the number of voices implied in the musical pattern: *monophonic* for one voice, and *polyphonic* for several voices ; and (2) the conditions expressed in the pattern. The latter either addresses (i) the rhythm only (leading to consider the connection between musical events in the graph-based representation), or the pitches only (leading the consider the facts attached to the musical events in the graph-based representation), or both, and (ii) the involvement or not of the hierarchical structure in the pattern.

Queries *Qseries** browse the time series of the music scores of the database, looking for a sequence of notes. The queries[8] of the form of *QseriesPathPitches* take into account the pitches only, meaning that such queries consider the flow of events and the facts attached to the events (in the spirit of Figure 11). The queries of the form of *QseriesPathRhythm* take into account the rhythm only, meaning that such queries consider the flow of events and the duration in the edges connecting events (in the spirit of Figure 13.(a)). The queries of the form of *QseriesPathRhythmAndPitches* take both the pitches and the rhythm into account (in the spirit of Figure 12.(a)). Such queries have (almost) the form of a path (as discussed in page 19). Let us note that if the pattern takes the pitches into account, then the implemented graph pattern is not strictly a path as it not only concerns a path of event nodes but also concerns the fact nodes attached to the event nodes (see the patterns in Figure 11 and Figure 12). The queries of the form of *QhierPath* browse the scores' hierarchical structures only, looking for a path in such structures. In the following, we denote the *QhierPath* queries and the *Qseries** queries by "*path-like*" queries.

The queries of the form of *QmixPitches*, *QmixRhythm* and *QmixRhythmAndPitches* are declined versions that, in addition to the flow of events, take into account the membership of the events in some measures of the hierarchical structure (which is an additional rhythmic condition). Similarly

[7]The benchmark queries, music scores and their *Muster* representations can be dowloaded at `https://www-shaman.irisa.fr/benchmarkmusicscoresgraph/`

[8]We recall that, for each form of query, queries of different sizes are considered in the benchmark (the sizes of the queries are discussed later).

Query	Musical pattern					Query characteristics					
	Type		Cond.			Form		Data explor.			
	Monophonic	Polyphonic	Pitches	Durations	Hier. structure	Path-like	Acyclic graph	hierar. structure	series of events	Anywhere in score	Incipit
QseriesPathPitches	✓		✓			✓			✓	✓	
QseriesPathRhythm	✓			✓		✓			✓	✓	
QseriesPathRhythmAndPitches	✓		✓	✓		✓			✓	✓	
QhierPath	✓					✓		✓		✓	
QmixPitches	✓		✓		✓		✓	✓	✓	✓	
QmixRhythm	✓			✓	✓		✓	✓	✓	✓	
QmixRhythmAndPitches	✓		✓	✓	✓		✓	✓	✓	✓	
QmixIncipitPitches	✓		✓		✓		✓	✓	✓		✓
QmixIncipitRhythm	✓			✓	✓		✓	✓	✓		✓
QmixIncipitRhythmAndPitches	✓		✓	✓	✓		✓	✓	✓		✓
QPolySeriesPathPitches		✓	✓				✓		✓	✓	
QPolySeriesPathRhythm		✓		✓			✓		✓	✓	
QPolySeriesPathRhythmAndPitches		✓	✓	✓			✓		✓	✓	
QPolyMixPitches		✓	✓		✓		✓	✓	✓	✓	
QPolyMixRhythm		✓		✓	✓		✓	✓	✓	✓	
QPolyMixRhythmAndPitches		✓	✓	✓	✓		✓	✓	✓	✓	
QPolyIncipitSeriesPathPitches		✓	✓				✓		✓	✓	✓
QPolyIncipitSeriesPathRhythm		✓		✓			✓		✓	✓	✓
QPolyIncipitSeriesPathRhythmAndPitches		✓	✓	✓			✓		✓	✓	✓
QPolyIncipitMixPitches		✓	✓		✓		✓	✓	✓	✓	✓
QPolyIncipitMixRhythm		✓		✓	✓		✓	✓	✓	✓	✓
QPolyIncipitMixRhythmAndPitches		✓	✓	✓	✓		✓	✓	✓	✓	✓

Table 2: Benchmark queries

to the *QseriesPath** versions, in the flow of musical events, the queries of the form *QmixPitches* consider the pitches (flow of events and facts attached to the events) only, the queries of the form *QmixRhythm* consider the rhythm only (flow of events and the duration in the edges connecting events) only, and the queries of the form *QmixRhythmAndPitches* consider both of them. Queries *Qmix** rely on an acyclic pattern that concerns both the flow of events and the hierarchical structure. Thus, for each score, they mix the browsing of the flow of events and the hierarchical structure.

Queries *QmixIncipitPitches*, *QmixIncipitRhythm* and *QmixIncipitRhythmAndPitches* look for the same pattern than *QmixPitches*, *QmixRhythm*

and *QmixRhythmAndPitches* respectively, but the musical pattern must be the incipit (i.e., at the beginning) of the score.

Finally, queries of the form of *Qpoly** are polyphonic queries versions of the above queries, that search for a pattern embedding two synchronized voices, in the spirit of the P_{poly} pattern in Figure 14. Two kinds of queries are considered: queries *QpolySeries**, which are polyphonic versions of the *Qseries** monophonic queries, and the queries *QPolyMix**, which are polyphonic versions of the *Qmix**monophonic queries.

Each *QPolyIncipitX* query implements the same polyphonic pattern than *QpolyX*, but restricted to the incipit of the score.

Topologically, the queries may be characterized by the form of the pattern they involve and the part of the graph-based representations they intent to explore, as summarized in table 2. These characteristics will allow us to explain results of the benchmark.

Regarding the benchmarking, each kind of query is provided into nine different sizes going from 2 to 10 musical events in the musical patterns, involving between 4 to 36 nodes according to the form of the query[9]. Table 3 gives the number of nodes and edges associated with the number of musical events for each query version.

6.2. The benchmark databases

The queries were evaluated[10] over 8 different databases populated with an increasing amount of *Muster* representations of music scores stemming from the – real – Bach chorals collection available at the Neuma platform [36] (the whole collection contains 403 scores[11]). Table 4 summarises the content of the benchmark databases.

At this stage, it is worth recalling that the size of a database that contains graph representations of scores varies from a collection to another: a database of 200,000 nodes may contain approximately from 300 complex

[9] In order to preserve a relevant semantic of the queries, the number of nodes in the intermediate versions may vary from a kind of query to another.

[10] The benchmark was performed with Neo4j 4.4.3, installed on a Ubuntu 20.04.2 LTS (GNU/Linux 5.15.0-56-generic x86_64) system with Intel(R) Xeon(R) Silver 4110 CPU @ 2.10GHz CPU with 376 GB of RAM. The Neo4j caches are cleaned between each query execution.

[11] The scores in the Neuma platform are made available in the MEI format. These scores were retrieved from the platform and then converted in their *Muster* representation, via the Musypher tool, in order to populate the Neo4j graph databases of the benchmark.

Query: \ Size versions:	2 events	3 events	4 events	5 events	6 events	7 events	8 events	9 events	10 events
QseriesPathPitches	(4,3)	(6,5)	(8,7)	(10,9)	(12,11)	(14,13)	(16,15)	(18,17)	(20,19)
QseriesPathRhythm	(2,1)	(3,1)	(4,3)	(5,4)	(6,5)	(7,6)	(8,7)	(9,7)	(10,9)
QseriesPathRhythmAndPitch	(4,3)	(6,5)	(8,7)	(10,9)	(12,11)	(14,13)	(16,15)	(18,17)	(20,19)
QhierPath	(4,3)	(5,4)	(6,5)	(7,6)	(8,7)	(9,8)	(10,9)	(11,10)	(12,11)
QmixPitches	(8,9)	(10,11)	(12,14)	(14,17)	(17,20)	(19,23)	(21,26)	(21,29)	(26,33)
QmixRhythmAndPitches	(8,9)	(10,11)	(12,14)	(14,17)	(17,20)	(19,23)	(21,26)	(21,29)	(26,33)
QmixRhythm	(6,6)	(7,8)	(8,10)	(9,12)	(10,15)	(12,17)	(13,19)	(14,21)	(16,24)
QPolySeriesPathPitches	(6,4)	(8,6)	(10,8)	(12,10)	(14,12)	(16,14)	(18,16)	(19,18)	(22,20)
QPolySeriesPathRhythm	(2,1)	(3,2)	(4,3)	(5,4)	(6,5)	(7,6)	(8,7)	(9,8)	(10,9)
QPolySeriesPathRhythmAndPitches	(6,4)	(8,6)	(10,8)	(12,10)	(14,12)	(16,14)	(18,16)	(19,18)	(22,20)
QPolyMixPitches	(7,6)	(9,11)	(11,12)	(14,17)	(16,19)	(19,13)	(20,15)	(22,29)	(24,31)
QPolyMixRhythm	(5,4)	(6,6)	(7,8)	(9,12)	(10,14)	(11,15)	(12,17)	(13,19)	(14,21)
QPolyMixRhythmAndPitches	(7,6)	(9,11)	(11,12)	(14,17)	(16,19)	(19,13)	(20,15)	(22,29)	(24,31)

(For the incipit versions, add 1 node dans 1 edge.)

Table 3: Number of nodes and edges embedded in the benchmark queries

Database	Scores	Nodes	Event nodes	Relationships
DB1	50	30,558	14,790	46,351
DB2	100	68,218	33,070	10,3421
DB3	150	107,645	52,282	163,038
DB4	200	136,118	66,053	20,6217
DB5	250	168,958	81,949	256,002
DB6	300	198,593	96,272	300,948
DB7	350	229,278	111,121	347,474
DB8	400	260,861	126,415	395,370

Table 4: Databases of the benchmark

music scores to 1800 more simple ones (see the discussion concerning the size of a graph-based representation on page 17).

6.3. Results and discussion

Figure 19 shows the execution time of the monophonic queries embedding a pattern that contains 10 nodes, according to the size of the database. In this figure, the path-like queries are modelled by dotted lines, and the other queries, which implement an acyclic pattern, are modelled by dashed lines (for the mix incipit queries) and solid lines (for the other mix queries). Clearly, these results show that some queries have a drastically lower execution time than the other ones. The most effective queries are the query

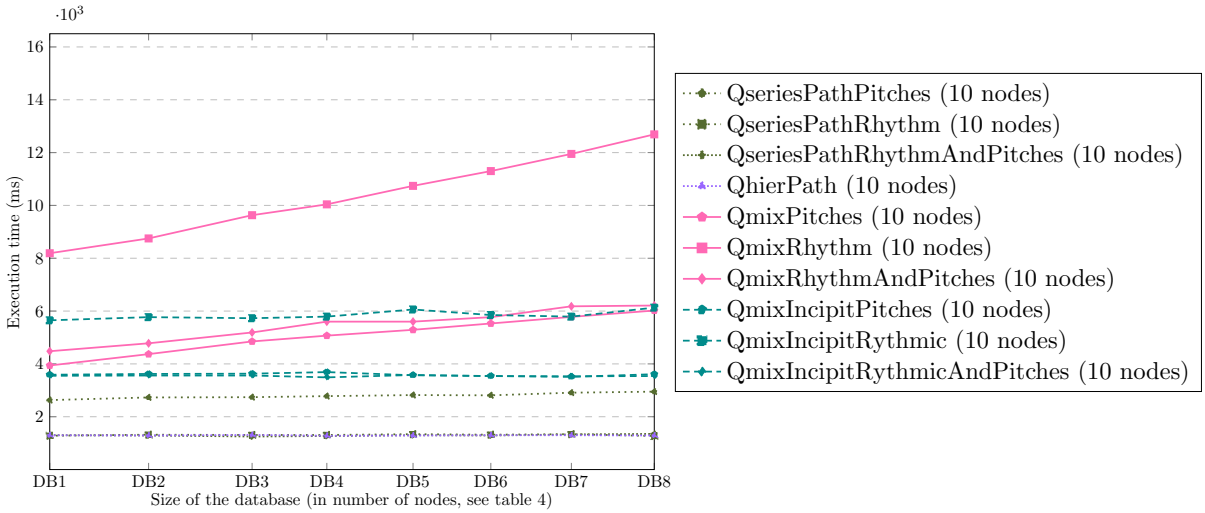


Figure 19: Query execution time (in ms) of the monophonic queries, in their size version of 10 nodes, w.r.t. the size of the database

QhierPath (path query over the hierarchical rhythmic structure) and queries *QseriesPath** (path and path-like queries over the flow of events)[12] w.r.t. the queries that implement a pattern having the form of a graph. This observation is consistent with the general theoretical studies of the literature, which proved that the cost of the topological query evaluation over a graph depends on the form of the query pattern [42, 39, 43, 21], where path queries are proved to be well-behaved fragments of graph pattern queries. In our case, the considered "path-like" queries are not strictly path queries (see the discussion in page 26), but the tractability remains in our context, explained by the fact that the evaluation of such queries requires "only an additional step" to be added for each mapping of an event node, in order to reach an attached fact node.

We now turn to patterns having the form of an acyclic graph (dashed and solid lines in the figures), denoted by *Qmix**. They exhibit a higher execution cost than the path-like ones. This is not so surprising as queries *Qmix** rely on acyclic graph patterns, which is a class of queries that is strictly more expressive than the class of path-like queries [42, 39, 43, 21]. Queries *QmixIncipitPitches*, *QmixIncipitRhythm* and *QmixIncipitRhythm* search a

[12] As the cost of the *QseriesPath** query is drastically low, we do not present the evaluation cost for their incipit version, which is even lower.

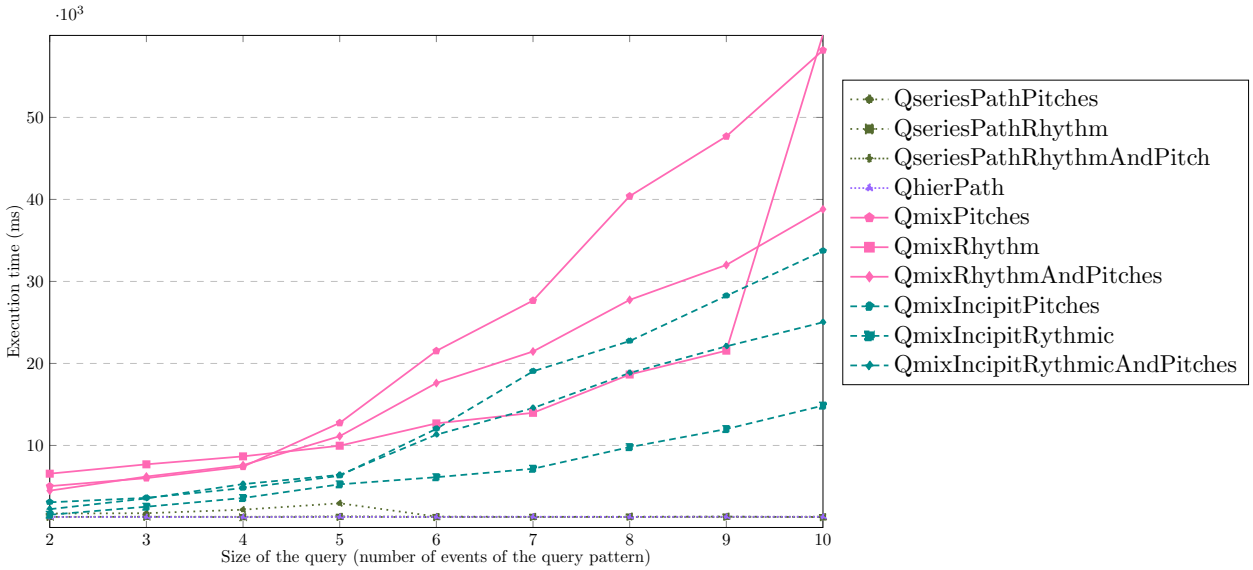


Figure 20: Execution time, in DB8, for the monophonic queries w.r.t. the number of events in the query pattern

pattern at the beginning of each score, so their search space is restricted (only the beginning of each score is browsed for evaluating the query). This explains why that size of the database hardly affects their evaluation. However, when the pattern has to be found anywhere in the score, that is to say for the queries *QmixRhythm*, *QmixPitches* and *QmixRhythmAndPitches*, the search space (obviously) increases faster with the size of the database, and so does the evaluation cost.

Finally, we can also observe that the queries *Qmix(Incipit)Rhythm*, w.r.t. the other queries *Qmix**, have a higher execution cost. This can be explained by the fact that, for a same total number number of nodes involved in the query pattern, the queries *Qmix(Incipit)Rhythm* contain -by construction- a higher proportion of *event* nodes to be retrieved amongst the nodes composing the pattern (as they do not explore *fact* nodes). This observation leads to examine the impact of the number of musical events (rather than the total number of nodes) embedded in the query over the evaluation time.

Figure 20 shows the execution time of the monophonic queries, for database DB8 (260,861 nodes), w.r.t. the size of the query in terms of number of events that compose the query pattern (the correspondence between the size of each query and its number of events in given in Table 3). According to this figure, the evaluation cost of a monophonic query also depends on the

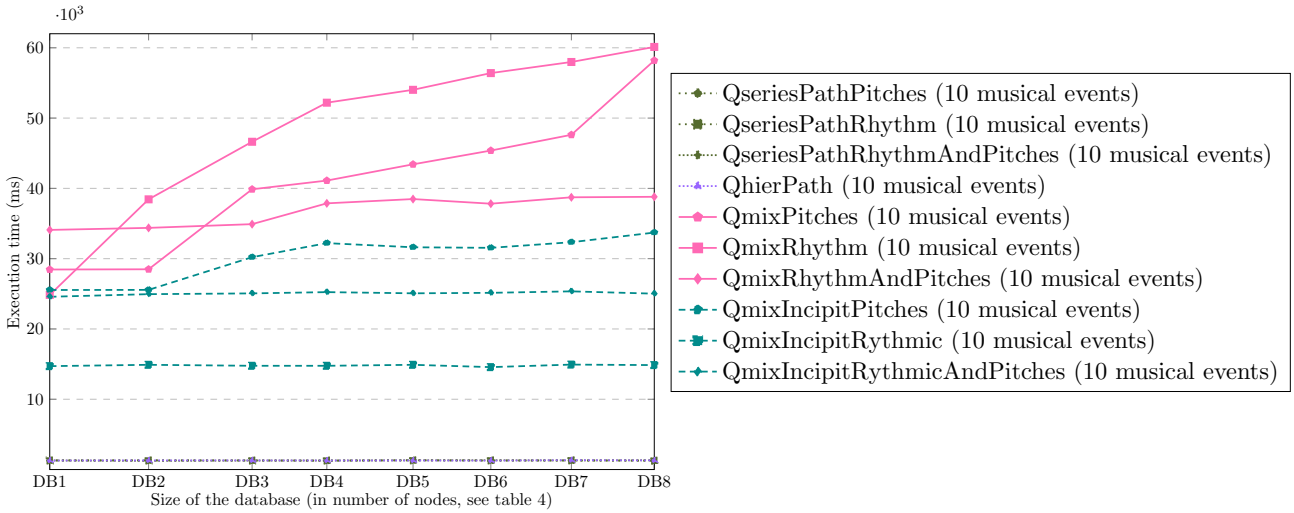


Figure 21: Query execution time (en ms) of the monophonic queries, in their size version of 10 musical events, w.r.t. the size of the database

number of musical events that appear in the query pattern. Such a vision also provides a functional-oriented perspective of the size of a query, as the number of musical events reflects the number of notes considered by the user in his musical pattern.

Figure 21 shows the behaviour of monophonic queries for the queries embedding 10 musical events. The observations drawn from these results reach the previous ones: path-like queries have a low execution cost; incipit queries have satisfying and steady execution time (proportional to the database size), and finally mix queries have an execution time that increases faster with the size of the database, reaching approximately 1 minute long in our worst case.

Figure 22 shows the execution time of polyphonic queries (see Table 2) of 10 events. Such queries look for a pattern of 5 musical events in a voice of a score, synchronised with another pattern of 5 musical events in another voice of the same score (in other words, compared to the monophonic version 10 events, the events to be retrieved are "distributed" over two voices (5 events in each voice) in the polyphonic version).

These results show first that the behaviour of polyphonic queries mostly follows the trend of the monophonic ones, with an execution cost that is close to the monophonic queries of the same size. This result is rational as, from a theoretical point of view, the evaluation of a polyphonic query of 10 events can be reduced, at worst, to independently evaluating the music pattern of

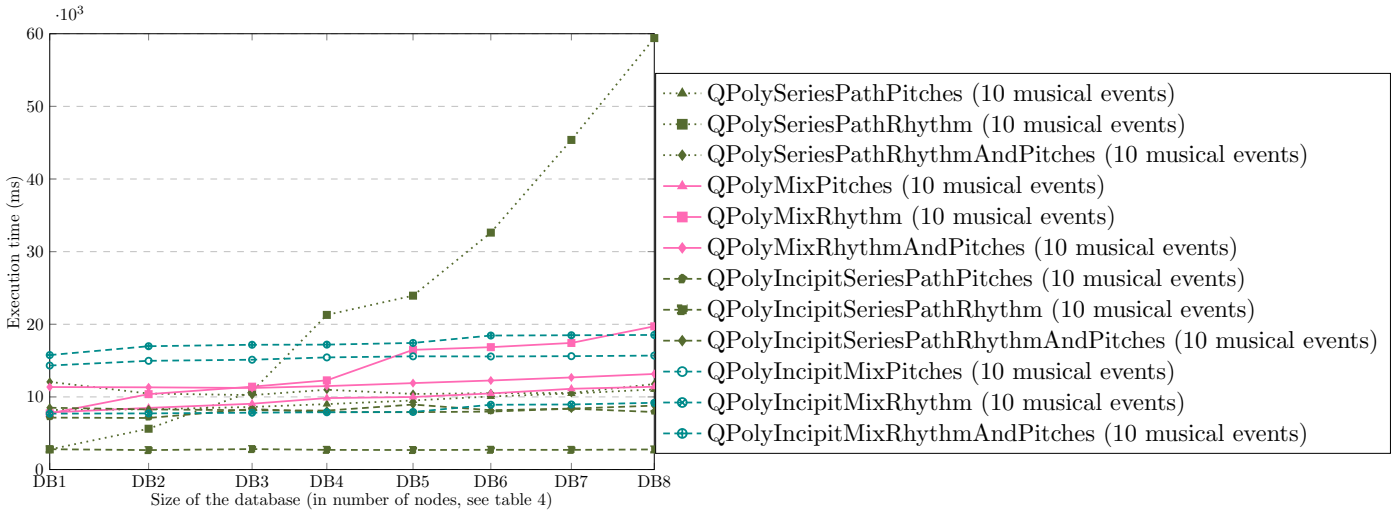


Figure 22: Query execution time (in ms) of the polyphonic queries, in their size version of 10 events, w.r.t. the size of the database

5 events for each voice, and then intersect the two results sets. But, depending on the query, the additional cost may be significantly higher for some polyphonic versions. This is the case here for a *QPolySeriesPathRhythmic* query. The extra cost is explained by the fact that a polyphonic pattern has necessarily the form of an acyclic graph (even if the polyphonic pattern searches for two series) as it considers both the organisation of the voices that compose the score (see the left part of Figure 10), and the content of the voices. It then "theoretically" looses the good properties of path-like queries. In practice here, the *QPolySeriesPathRhythmic* does not prune easily the search space (statically, there are more data that satisfy a rhythm pattern than data that satisfy a melodic one, so a larger data space has to be browsed in order to find the rhythm pattern, with consequently a higher cost of merging the results of the two voice patterns)[13].

In conclusion of this section, we can summarize the results of the benchmark study. The monophonic queries have a good behaviour in terms of execution time, with a particularly low execution cost for path-like queries, and a steady and affordable execution cost of the incipit ones. The other monophonic queries that combine the exploration of the flow of events and

[13]The mix version is more efficient as it prunes more efficiently the search space by means of the constraint over the rhythmic tree expressed in the query.

the hierarchical structure express an acyclic graph pattern, whose evaluation time grows faster in the data size. Polyphonic queries follow the trends, but some of them may have a significantly higher cost, when the pattern has the form of an acyclic graph pattern and does not allow to prune easily the search space during evaluation. Higher execution costs are around 1 minute for some queries. These are acceptable execution costs, but they remain definitely too slow for a real-time querying context. Optimization mechanisms that could improve the efficiency of the execution process could be envisioned in order to enhance the usability of the "slower" queries. These are future works (see the Section 7 for more details).

7. Conclusion and perspectives

We proposed a graph-based data model called *Muster*. It relies on a representation of digital music documents as structured objects, focused on music content, cleared from side information related to representation purposes. *Muster* allows to express, in a single representation, both the hierarchical temporal organisation of the music events flow, and the time series one, where such flows are simply modelled as time series. We believe that this representation is expressive, captures the essential aspects conveyed by music notation, and stays close to the intuitive understanding of sheet music scores. It provides a comprehensive and unambiguous topological representation of the music content, aiming at supporting the user in the conception of graph pattern queries.

We then investigated the exploration of such data through graph pattern queries, and gave several examples of patterns identification in a collection of music scores. This approach benefits from some unique features in the field of declarative languages. A first one is its flexibility: patterns can specify full or partial conditions on nodes, and they cover either tiny or large parts of the score graphs. A second is its expressivity. It appears easy to define relations that link remote parts of a score, and these relations can be strongly or loosely constrained, depending on the user's needs. We believe that this constitutes a promising and original way to enable a powerful interaction with digital music notation.

We presented a proof-of-concept of the approach, implementing the *Muster* model in the form of property graphs loaded in the Neo4j database (using a tool called Musypher), and then expressing searches and analyses with the Cypher language. We also studied, through a benchmark implemented over a real dataset of music scores (stemming from the Neuma DSL), the behaviour of a set of graph pattern queries.

We plan to extend this work in several directions. A first one concerns the **extension of the model**. We considered only the domain of sound facts, but other domains can be added to enrich the description of music information, following the same principle: such information is a mapping from a rhythmic organization of time to facts. The domain of *syllables* for instance allows to model lyrics in vocal music; one could add domains of semantic facts (timbre, texture, intensity) to model in general *annotations* that qualify temporal fragments of a music piece, at any relevant timestamp of the metric organization.

A second perspective concerns the **query optimization**. The benchmark study (Section 6) showed that execution costs are tractable (worst cases around 1 minute for our use cases), but the optimization of the execution process should be envisioned in order to improve the usability of the approach in a real-time querying context. A possibility could be to add complementary information, attached with the nodes of the data, that would prune as early as possible the search space, like the future notes that are reachable from an event (see for instance [44], that could be adapted to the research of paths in the flow of events). Another possibility would be to explicit the synchronisation of the events, in the graph-based modelling itself. Such a shortcut could be used during the evaluation process, preventing the system to retrieve candidate synchronized nodes and compare their timestamp information.

A third part of future work relates to the **implementation** of the proposed approach, as *a querying module embedded in a real DSL*, offering a user-friendly interface to the users. This implementation is in progress in the SKRID platform [38] (see Figure 23). It takes the form of an intermediate application layer, standing between a digital piano keyboard as graphical user interface (it allows the user to define a monophonic musical pattern, by possibly relaxing the constraints over the pitches or the rhythm), and the graph database. The layer (i) transcripts the (monophonic) musical pattern in a graph pattern query, (ii) sends this query to a Neo4j database (populated with the score collections), (iii) retrieves the Neo4j answers, (iv) which are then rendered to the user by displaying the set of music scores featuring the pattern, where the pattern occurrences are highlighted. For advanced users, a specific frame will allow the users to leverage the whole expressive power of the model and graph pattern queries, by defining their own *ad-hoc* Cypher queries. Implementing this frame will allow us to assess the understandability of the data model and the **usability of the graph-based query language**.

A limitation of the approach stands in the fact that, in the DSL that

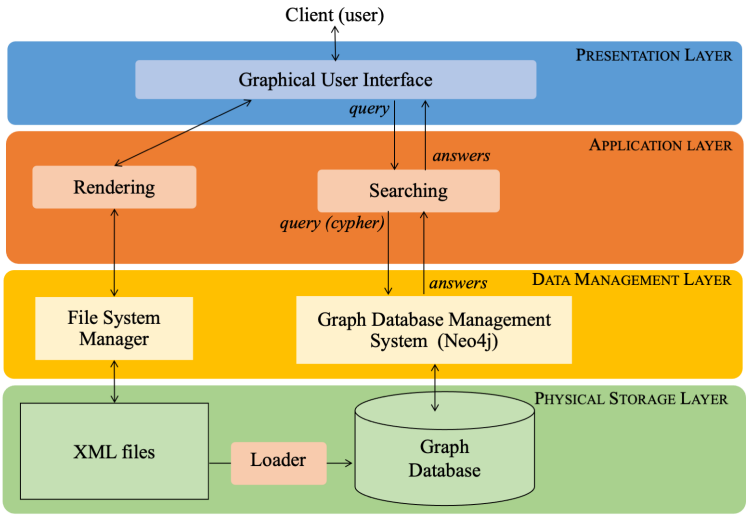


Figure 23: Architecture of the proof-of-concept (SKRID DSL platform)

implements the framework, the music content is stored, duplicated, 1) in the graph database, and 2) in the form of the initial XML-based files. XML-based files are still needed because they contain all the metadata and rendering information associated with a music score. Technically, the graph database could store the comprehensive content of XML files (as the property graph formalism is strictly more expressive than the semi-structured one) but adding this information to the graph would obscure its content by intricately blending information pertaining to both the content and its specific rendering. One then has to choose between improving the efficiency/precision of the model, and improving its understandability. This is a classical trade-off that has to be found between interdependent quality dimensions of a data quality model [45]. A possible solution would be to implement an external schema mechanism [46] that would hide the unneeded information to the user in the querying process.

Acknowledgement

This work is funded by the University of Rennes[14] and by the European project Polifonia[15].

References

[1] W. Apel, The Notation Of Polyphonic Music 900-1600, The Medieval Academy Of America, 1961.

[2] E. Gould, Behind Bars, Faber Music, 2011.

[3] D. Huron, Music information processing using the humdrum toolkit: Concepts, examples, and lessons, Computer Music Journal 26 (2) (2002) 11–26.

[4] M. Good, The Virtual Score: Representation, Retrieval, Restoration, MIT Press, 2001, Ch. MusicXML for Notation and Analysis, pp. 113–124.

[5] Music Notation Community Group, `https://www.w3.org/community/music-notation/`, (consult. date) (2023).

[6] P. Rolland, The Music Encoding Initiative (MEI), in: Proc. of the Intl. Conf. on Musical Applications Using XML, 2002, pp. 55–59.

[7] Music Encoding Initiative (MEI) web site, `http://www.music-encoding.org`, (consult. date) (2023).

[8] C. S. Sapp, Online Database of Scores in the Humdrum File Format, in: Proc. of the Intl. Conf. on Music Information Retrieval (ISMIR), 2005, pp. 664–665.

[9] P. Rigaux, L. Abrouk, H. Audéon, N. Cullot, C. Davy-Rigaux, Z. Faget, E. Gavignet, D. Gross-Amblard, A. Tacaille, V. Thion-Goasdoué, The design and implementation of neuma, a collaborative digital scores library - requirements, architecture, and models, Int. J. on Digital Libraries 12 (2-3) (2012) 73–88.

[14] SKRID project of the *Défis scientifiques* ("Scientific challenges") program of the University of Rennes (France).

[15] `https://polifonia-project.eu/`

[10] MuseScore web site, https://musescore.org/, (consult. date) (2023).

[11] N. Orio, Music retrieval: A tutorial and review, Found. Trends Inf. Retr. 1 (1) (2006) 1–90. doi:10.1561/1500000002.
URL https://doi.org/10.1561/1500000002

[12] M. Casey, R. Veltkamp, M. Goto, M. Leman, C. Rhodes, M. Slaney, Content-based music information retrieval: current directions and future challenges, Proceedings of the IEEE 96 (4) (2008) 668–696.

[13] F. Foscarin, P. Rigaux, V. Thion, Data Quality Assessment in Digital Score Libraries. The GioQoso Project, Intl. Journal on Digital Libraries 22 (2) (2021) 159–173.

[14] D. Fiala, P. Rigaux, A. Tacaille, V. Thion, m. o. The Gioqoso project, Data Quality Rules for Digital Score Libraries, Research report, IRISA, Université de Rennes (2018).
URL https://hal.inria.fr/hal-01734821

[15] J. Ganseman, P. Scheunders, W. D'haes, Using XQuery on MusicXML Databases for Musicological Analysis, in: Proc. of the Intl. Conf. on Music Information Retrieval (ISMIR), 2008.

[16] R. Fournier-S'niehotta, P. Rigaux, N. Travers, Querying music notation, in: 23rd International Symposium on Temporal Representation and Reasoning, TIME 2016, Kongens Lyngby, Denmark, October 17-19, 2016, IEEE Computer Society, 2016, pp. 51–59.

[17] R. Fournier-S'niehotta, P. Rigaux, N. Travers, Querying XML score databases: XQuery is not enough!, in: Proc. of the Intl. Conf. on Music Information Retrieval (ISMIR), 2016, pp. 723–729.

[18] M. S. Cuthbert, C. Ariza, Music21: A Toolkit for Computer-Aided Musicology and Symbolic Music Data, in: Proc. of the Intl. Conf. on Music Information Retrieval (ISMIR), 2010, pp. 637–642.

[19] R. Angles, C. Gutierrez, Survey of graph database models, ACM CSUR 40 (1) (2008) 1–39.

[20] R. Angles, A comparison of current graph database models, in: Proc. of the ICDE Workshops, 2012, pp. 171–177.

[21] R. Angles, M. Arenas, P. Barceló, A. Hogan, J. L. Reutter, D. Vrgoc, Foundations of modern query languages for graph databases, ACM Comput. Surv. (CSUR) 50 (5) (2017) 68:1–68:40.

[22] A. Bonifati, G. H. L. Fletcher, H. Voigt, N. Yakovets, Querying Graphs, Synthesis Lectures on Data Management, Morgan & Claypool Publishers, 2018.

[23] L. Peusner, A graph topological representation of melody scores, Leonardo Music J. 12 (2002) 33–40.

[24] P. Rigaux, V. Thion, Towards a graph-oriented perspective for querying music scores, in: Proc. of Inforsid, 2022, pp. 63–78.

[25] A. Pinto, R. Leuken, M. F. Demirci, F. Wiering, R. Veltkamp, Indexing music collections through graph spectra, in: Proc. of the Intl. Conf. on Music Information Retrieval (ISMIR), 2007, pp. 153–156.

[26] R. Jin, C. Raphael, Graph-based rhythm interpretation, in: Proc. of the Intl. Conf. on Music Information Retrieval (ISMIR), 2015, pp. 343–349.

[27] F. Lerdahl, R. Jackendoff, An overview of hierarchical structure in music, Music Perception: An Interdisciplinary Journal 1 (2) (1983) 229–252.

[28] M. Good, The Virtual Score: Representation, Retrieval, Restoration, W. B. Hewlett and E. Selfridge-Field, MIT Press, 2001, Ch. "MusicXML for Notation and Analysis", pp. 113–124.

[29] W. C. group, MusicXML Version 4.0, Final Report, `https://w3c.github.io/musicxml/` (2021).

[30] J. Ganseman, P. Scheunders, W. D'haes, Using XQuery on MusicXML databases for musicological analysis, in: Proc. of the Intl. Conf. on Music Information Retrieval (ISMIR), 2008, pp. 433–438.

[31] T. Zhu, R. Fournier-S'niehotta, P. Rigaux, N. Travers, A Framework for Content-based Search in Large Music Collections, Big Data and Cognitive Computing 23 (6) (2022).

[32] R. Fournier-S'niehotta, P. Rigaux, N. Travers, Modeling Music as Synchronized Time Series: Application to Music Score Collections, Information Systems 73 (2018) 35–49.

[33] B. Mokbel, A. Hasenfuss, B. Hammer, Graph-based representation of symbolic musical data, in: Proc. of the Workshop on Graph-Based Representations in Pattern Recognition, Vol. 5534 of Lecture Notes in Computer Science, 2009, pp. 42–51.

[34] J. Haydn, Das lied der deutschen, lyrics by August Heinrich Hoffmann von Fallersleben (1797).

[35] International Organization for Standardization, ISO 16:1975 Acoustics – Standard tuning frequency (1975).

[36] The NEUMA platform, `http://neuma.huma-num.fr`, accessed Feb. 2023 (2022).

[37] J. Mahé, Essai sur les Antiquités du département du Morbihan, Vannes, imprimerie Galles aîné, 1825, available in the Gallica digital library of the Bibliothèque Nationale de France (BnF) at `https://gallica.bnf.fr/ark:/12148/bpt6k61026071`.

[38] The SKRID platform, `https://shaman.enssat.fr/skrid/`, (consult. date) (2023).

[39] P. Barceló, Querying graph databases, in: Proc. of the ACM Symp. on Principles of Database Syst. (PODS), 2013, pp. 175–188.

[40] A. Blatter, Revisiting Music Theory: A Guide to the Practice., London and New York: Routledge, 2007.

[41] Neo Technology, The Neo4j documentation, https://neo4j.com/docs/, (Consult. date) (2023).

[42] P. T. Wood, Query languages for graph databases, SIGMOD Rec. 41 (1) (2012) 50–60.

[43] P. Barceló, L. Libkin, J. L. Reutter, Querying regular graph patterns, J. ACM 61 (1) (2014) 8:1–8:54.

[44] M. Farhan, Q. Wang, Y. Lin, B. D. McKay, A highly scalable labelling approach for exact distance queries in complex networks, in: Proc. of the Intl. Conf. on Extending Database Technology (EDBT), 2019, pp. 13–24.

[45] W. H. Delone, E. R. McLean, The DeLone and McLean Model of Information Systems Success: A Ten-Year Update, Journal of Management Information Systems 19 (4) (2003) 9–30.

[46] R. Ramakrishnan, J. Gehrke, Database Management Systems, 3rd Edition, McGraw-Hill, Inc., USA, 2002.