Where Graphs Meet Fuzzy Logic — A DBMS-Centered Engine for Polyphonic Music Matching in Score Databases

Adel Aly
adel.aly@irisa.fr
Univ. Rennes, CNRS
Lannion, France

Olivier Pivert
olivier.pivert@irisa.fr
Univ. Rennes, CNRS
Lannion, France

Virginie Thion
virginie.thion@irisa.fr
Univ. Rennes, CNRS
Lannion, France

Abstract

A Digital Score Library (DSL) is a digital system for storing, managing, and disseminating musical scores. Although such systems have traditionally relied on sheet music for engraving, modern digital representations now enable content-aware services, such as searching for specific musical patterns within the library's scores. Though monophonic pattern retrieval is well-studied, **polyphonic pattern retrieval**, which is the subject of this demonstration, remains a complex, open challenge. This demonstration introduces Skrid, an online DSL using graph-based storage for musical content, and Malo, its flexible querying module that enables the **approximate search** of **polyphonic melodic patterns**, **ranks the results** by relevance, and provides a **detailed explanation** of the answers.

Keywords

Digital musical score library; Graph representation; Approximate melodic pattern matching, Polyphonic pattern

1 Introduction

A new generation of DSLs. Music is an indispensable component of the world's cultural heritage and is studied across various domains, like music, musicology, history, and ethnomusicology. While music sheets have traditionally been used for engraving musical scores, the advent of modern digital formats (such as semi-structured ones [7, 15], or graph-based ones [8, 14]) has transformed music notation into a valuable resource for (semi-)automatic music information processing. Today, DSLs provide access to large collections of digitally encoded musical scores, leveraging the encoding of musical content to enable advanced retrieval features.

Polyphonic melodic pattern searching. The primary users of a DSL encompass music performers, scholars, students, and specialized professionals, including musicologists, engravers, composers, and librarians. These diverse users interact with the library for distinct purposes. Performers typically search for pieces by metadata such as title or composer, while scholars and theorists often seek musical works containing specific melodic patterns — the core challenge addressed in this work. The retrieval of monophonic musical patterns (defined as sequences of musical events) in a single voice has been extensively studied. However, polyphonic pattern retrieval, which involves multiple time-aligned musical sequences, remains a complex and open problem, widely recognized for its difficulty in the literature [6, 10]. This explains why only a few existing DSLs currently offer this functionality (see Section 6).

In the demonstration, we first present the Skrid platform, an online DSL that stores musical scores in a property graph database via the data model of [14]. Such a model provides a topology-oriented and intuitive representation of musical scores and enables expressive graph-pattern queries (see [14] for details), including polyphonic ones. It also provides relationships between the elements of a musical score that facilitate navigation in data, paving the way for the optimization of polyphonic query evaluation. To our knowledge, Skrid is the only DSL in the literature that relies on such a data model.

Then, we present Malo, a querying tool that allows the **approximate searching of polyphonic musical pattern** occurrences in a digital score library. Malo (1) retrieves, from a (graph-based) collection of musical scores, the data that approximately match a given polyphonic pattern, (2) computes a similarity score (also referred to as *satisfaction degree* from now on) between each answer and the given pattern, (3) ranks the answers based on their satisfaction degrees, and (4) provides a detailed explanation for each answer. The software architecture of Malo in Skrid and the query evaluation process are also outlined in the paper. Optimization issues are also discussed. To our knowledge, no other DSL implements a polyphonic pattern matching tool that is both flexible and able to return ranked and explained answers.

The demonstration is divided into two parts: a scripted presentation that showcases the Skrid DSL and the Malo querying module through a scenario highlighting their key features and internal processing, followed by an interactive session in which participants can directly engage with the system to explore its capabilities.

2 Musical notation

First, we introduce key Western music notation concepts using the example score in Figure 1 . In Western music notation, a finite set of frequencies is used to refer to the sounds that may appear in a musical piece. Each frequency is associated with a pitch class (a letter C, D, E, F, G, A, and B) and an index known as the octave (in the set $\{1, \ldots, 7\}$). On a music sheet, *musical events* are graphically represented to define the sounds that compose the music piece. The symbols are written all along a group of horizontal lines called *staves*. A main symbol that appears in a musical score is the *note* (e.g., a symbol among 𝅗𝅥, 𝅗𝅥., ♪, ♫, 𝅝, etc.),

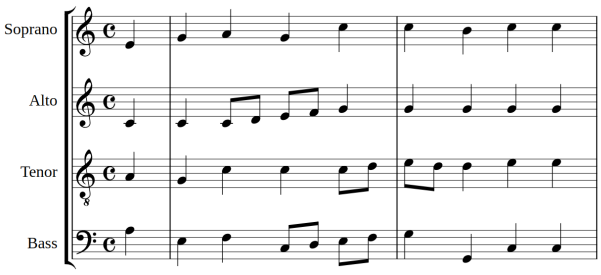
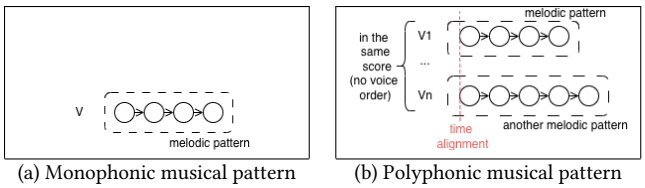


Figure 1: Beginning of Bach's Cantate *BWV111.6*

EDBT '26, Tampere (Finland)

10.48786/edbt.2026.76

which denotes a sound to be played. A note symbol encodes both the information of the frequency and the duration of the denoted sound. The vertical position of the note on the staff determines the frequency (e.g. if we consider the first staff of Figure 1, the first note is a $E4$, followed by a $G4$, an $A4$, etc.), while the shape of a note (including the head, stem, and flags) determines the duration (for instance, a ♪ note is two times shorter than a ♩ note). Time is divided into frames called *measures*, synchronized over the staves, and visually delineated by vertical *bars*. In Figure 1, we can see three measures. The whole musical content of a score is composed of several synchronized sequences of musical events associated with instruments or vocals. These sequences are called *voices*. A *monophonic musical score* is composed of exactly one voice. A *polyphonic musical score* is composed of a set of non-ordered voices w.r.t. the musical content (even if, of course, an order may be associated with voices for the rendering of the score). Figure 1 is composed of four voices.



(a) Monophonic musical pattern (b) Polyphonic musical pattern

Figure 2: Monophonic and polyphonic pattern

3 Polyphonic Musical Pattern

Before presenting the demonstration, let us first clarify the concepts of monophonic and polyphonic musical patterns. A *monophonic musical pattern* (see Figure 2.(a)) is a sequence of musical events found within a single voice of a musical score. This score may itself be either monophonic or polyphonic. If the score is polyphonic, then the monophonic pattern is searched for in each voice. A *polyphonic musical pattern* (see Figure 2.(b)) consists of multiple time-aligned sequences occurring along different voices of the same musical score, where the sequences begin at the same time. Figure 3 is a polyphonic musical pattern, which will be used for the demonstration scenario.

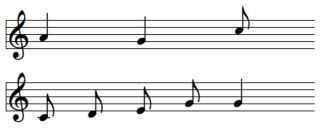


Figure 3: Melodic polyphonic pattern

4 System Overview

Skrid is an open data DSL platform dedicated to the preservation and dissemination of musical scores. Figure 4 presents the system architecture, which consists of two main components: a *back-end* (depicted in blue), responsible for data storage, management, and processing, and a *front-end* (depicted in orange), which hosts the user-interaction modules.

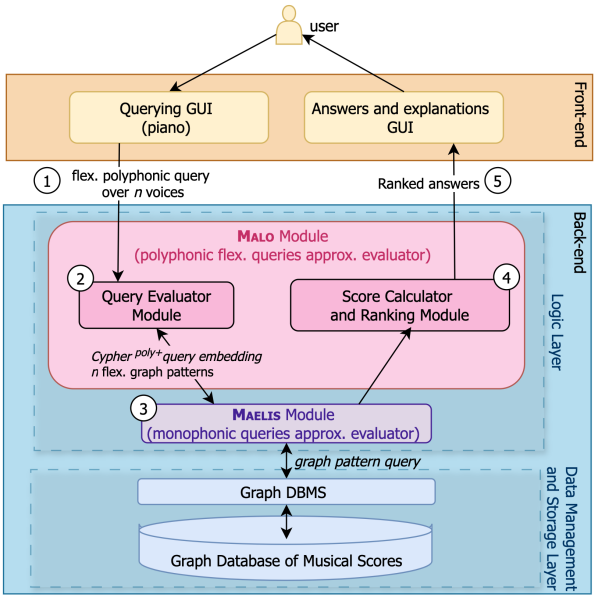


Figure 4: System architecture and query flow

4.1 Back-end

The back-end of the Skrid platform (in blue in Figure 4) is composed of a data management layer and functional modules, including the Maelis querying module the demonstration focuses on.

Data Model. The back-end integrates a storage layer that implements the property graph data model for musical score representation, proposed in [14], into a Neo4j [11] database. In this model, musical scores are stored as graph structures, where each musical voice is represented as a sequence of notes connected by edges, with additional edges organizing notes into measures (see Figure 5, which represents a section of the musical score of Figure 1). This graph-based representation offers an intuitive, topology-driven view of the scores. For brevity, we omit additional details about the model, as the discussion concentrates on the querying functionality (see [14] for an in-depth presentation of the data model and its features).

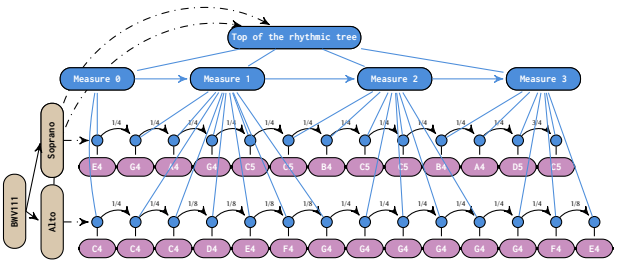


Figure 5: Graph-based representation of a part of Figure 1

Flexible Polyphonic Querying. Highlighted in pink in Figure4, Malo is the querying module for polyphonic melodic pattern evaluation. Malo relies on an extension of the Cypher graph-pattern query language, denoted by Cypher^{poly+} hereafter, that

allows to express flexible polyphonic queries. The extension supports queries containing the following key elements. **The polyphonic musical pattern** represents a subgraph to be searched within the data. This subgraph typically follows the structure depicted in Figure 2(b) and is expressed in a Cypher-like syntax. **Approximate matching** offering highly customizable flexibility in musical aspects such as pitch, duration, and note sequencing. Cypher^{poly+} leverages fuzzy set theory to model gradual criteria for these aspects, associating each with a membership function that computes a satisfaction degree. This quantifies the resemblance between the user's pattern and its approximate occurrences in the database. The individual satisfaction degrees are then aggregated into a global score.

Malo is implemented as an open-source Python-based add-on layer. This implementation consists of two modules: a *Query Evaluator* module and a *Score Calculator and Ranking* one, as depicted in pink in Figure 4. The *Query Evaluator* module processes user inputs from the front-end layer in the form of a composite pattern made of n melodic patterns (see Figure 2) and flexibility parameters that set the maximal tolerance thresholds allowed for the pitch, the duration, and the sequencing (stage 1 in Figure 4). For each threshold, a triangular fuzzy membership function is derived to compute a scoring function (as proposed in [2], straightforwardly extended to the polyphonic context). The Query Evaluator then generates a Cypher^{poly+} query (stage 2 in Figure 4), embedding the evaluation of the pattern over the database and retrieving the data needed to compute the satisfaction degrees. Malo leverages Maelis, a specialized module for approximate monophonic melodic pattern matching (Stage 3 in Figure 4). As demonstrated in [4] and detailed in [2, 3], Maelis identifies all occurrences of each (monophonic) melodic pattern within the polyphonic query and computes their corresponding satisfaction degrees.

Optimization issues. At this point, let us consider optimization issues. A naive evaluation of a polyphonic query would be to independently evaluate each melodic pattern and then merge the matches to reconstruct possible answers for the polyphonic query. However, Malo introduces a graph-aware **optimization** of the polyphonic query that (1) starts by matching the first melodic pattern of the query, and (2) for each match, as the melodic patterns of the query are time-aligned (must begin at the same time), Malo leverages the graph's relational structure to directly enter into the other voices (in detail, the path navigates through measure nodes to locate candidate nodes that serve as potential starting points for subsequent patterns), searching for the other melodic patterns of the query. This allows pruning large portions of the search space by exploiting the graph's connectivity. This optimization is made possible by the graph-based storage structure of musical scores, which enables efficient traversal between voices.

Then, the *Score Calculator and Ranking* module retrieves the answers and the information needed to compute the satisfaction degrees (stage 4 in Figure 4) and calculates the satisfaction degree associated with each answer, based on the aggregation of the partial satisfaction degrees. The overall degree allows rank-ordering the answers before presenting them to the user. The partial satisfaction degrees provide a detailed explanation for each answer (stage 5 in Figure 4), as illustrated in Section 4.2.

Malo handles monophonic queries as a special case of polyphonic queries, where only a single melodic pattern is involved. Thus, Malo evaluates both monophonic and polyphonic patterns. Additionally, exact melodic pattern queries are a special case of approximate ones, where the scoring function enforces strict matching only. Consequently, Malo supports all combinations of exact and approximate queries, whether monophonic or polyphonic.

4.2 Front-end

Users do not have any knowledge about the data model, formal query languages, or the process that retrieves the data. They interact with the system through user-friendly modules of the front-end layer (the orange block in Figure 4), which provide an online (web) access to the DSL. The querying GUI relies on a virtual piano keyboard to define the musical pattern, and web components for setting the flexibility parameters. Another GUI module is responsible for presenting musical scores retrieved by Malo, each annotated with its global satisfaction score. For every matching pattern occurrence, users can access detailed breakdowns of the satisfaction degrees, including fine-grained evaluations for individual notes. The GUI modules are implemented in JavaScript, and musical scores visualization is powered by the open-source library Verovio [16].

5 Demonstration Scenario

1. Scripted demonstration. We first showcase the capabilities and internal processing of Malo through a use case: UseCase$_{Marie}$ considers a musicologist, Marie, who searches for occurrences of the polyphonic melodic pattern of Figure 3, to analyze contrapuntal relationships in the (real) database of Bach's chorals[1], that contains 400 musical scores, modeled as a graph database of 254,104 nodes and 386,843 edges (during the demonstration, the audience will visualize the content of such a graph database through the Neo4j database visualization application [11]).

Using the virtual piano GUI, Marie enters the pattern of Figure 3. The exact search retrieves no answer. Then, Marie decides to look for approximate occurrences. She chooses to be more tolerant on the notes' pitches (she allows a tolerance of 2 tones on the pitches of the notes) and on the pattern rhythm (she enables a tolerance factor of 2 on the rhythm, meaning that she authorizes the duration of a note in a musical score to be at most the double and at least the half of that specified in the pattern). For each note, greater deviations in duration or pitch from the corresponding pattern note result in a lower satisfaction degree along those dimensions. Marie enters these preferences in the GUI, in a dedicated frame under the piano. Then, 131 approximate answers are retrieved, with 21 having a satisfaction degree greater than 70% (meaning, from the fuzzy set theory perspective, having a global membership degree greater than 0.7), ranked in decreasing order of their satisfaction degrees.[2] For each answer, a detailed explanation is provided. The Skrid's screenshot of Figure 6 is an example of such an answer, which presents an explained occurrence of Marie's polyphonic pattern (Fig 3) in the musical score BWV111.6 of Fig 1.

[1] Bach's chorals collection was provided by the Neuma DSL [12], and injected in Skrid.

[2] By default, the degree of an occurrence is the average of the degrees of each note, and the degree of a note is the average of the degrees for each dimension. The global degree of a musical score is the maximum degree amongst the occurrences of the pattern in the score. The aggregation operators can be parameterized in the framework.

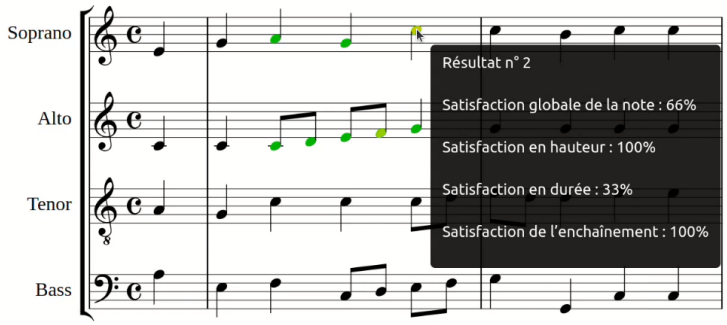


Figure 6: Explanation of an approx. matching (UseCase$_{Marie}$)

During the demonstration, the audience will observe how the user interacts with Skrid, with particular emphasis on flexible querying functionalities and the underlying graph-database queries. We will focus on the query processing pipeline, with a highlight on Malo's generation of the Cypher^{poly+} query and the implementation of the forementioned optimization techniques, Maelis' handling of this query, and the end-to-end data processing through the pipeline.

Skrid's framework also allows transposition-tolerant retrieval, which is a sub-problem known to be difficult [10], but made easy in Skrid as it relies on the interval between notes instead of the notes themselves (see [2] for the monophonic patterns processing, adapted here to the polyphonic patterns processing). This feature will be showcased in the demonstration.

2. Online interaction. In the second part of the demonstration, the audience will be invited to interact with the system.

6 Related Systems

Monophonic queries. Other systems in the literature address melodic pattern searching in musical score databases. However, most of them consider an exact matching process (see [1] for an overview). In the few systems that permit flexible matching, flexibility is not parameterizable, no matching degree is attached to the answers, and no explanation functionality is provided. In [13], the authors propose a monophonic query algorithm that matches note sequences across multiple voices in a polyphonic score. This approach differs from ours, as the queries it deals with remain monophonic in nature, even if polyphonic musical scores are queried.

Polyphonic queries. Several approaches have been proposed in the literature for polyphonic query processing. While some methods offer expressive query capabilities (e.g., [5]), they lack flexibility in handling approximate matches. Geometric approaches based on two-dimensional point sets (see [9] for a comprehensive overview) provide transposition-invariant matching, but existing implementations of these approaches typically lack user-configurable parameters and do not provide detailed explanations for their results w.r.t. user-defined configuration. Moreover, their underlying logic depends on an intermediate representation that prevents full exploitation of the DBMS engine, which is precisely what our work allows (see the optimization issues in Section 4.1).

Resources

Skrid is available online at https://shaman.enssat.fr/skrid. Its source codes are available in a GitLab platform, divided between the back-end https://gitlab.inria.fr/skrid/backend (Malo), the front-end https://gitlab.inria.fr/skrid/frontend, and the data https://gitlab.inria.fr/skrid/data.

Acknowledgments

We gratefully acknowledge Lannion-Trégor Communauté for its funding. We also wish to acknowledge the Dastum association, in particular, for its valuable collaboration, as well as the Neuma digital library of music scores. We extend our sincere gratitude to Robert Bouthillier, David Guillemois, Anne-Marie Nicol and Gwenaël Piel, together with the musicians and musicologists whose insights enriched our work.

References

[1] Adel Aly, Olivier Pivert, and Virginie Thion. 2024. Database Approaches to the Modelling and Querying of Musical Scores: a Survey. In *Proc. of the Intl. Conf. on Theory and Practice of Digital Libraries (TPDL).*
[2] Adel Aly, Olivier Pivert, and Virginie Thion. 2025. A Flexible Framework for Transposition-Aware Querying of a Musical Score Database (Best Paper Award). In *Proc. of the Intl. Conf. on Research Challenges in Information Science (RCIS).*
[3] Adel Aly, Olivier Pivert, and Virginie Thion. 2025. Fuzzy Retrieval of Musical Scores Based on Melodic Patterns. In *Proc. of the IEEE Intl. Conf. on Fuzzy Systems (FUZZ-IEEE).*
[4] Adel Aly, Olivier Pivert, and Virginie Thion. 2025. Maelis4Skrid: an Approximate Query Engine for an Online Graph-Based Musical Score Library. In *Companion Proc. of the ACM Web Conference 2025 (WWW).*
[5] Mathieu Bergeron and Darrell Conklin. 2008. Structured Polyphonic Patterns, Juan Pablo Bello, Elaine Chew, and Douglas Turnbull (Eds.). 69–74.
[6] Michael A. Casey, Remco Veltkamp, Masataka Goto, Marc Leman, Christophe Rhodes, and Malcolm Slaney. 2008. Content-based music information retrieval: current directions and future challenges. *Proc. of the IEEE* 96, 4 (2008), 668–696.
[7] Michael Good. 2001. *The Virtual Score: Representation, Retrieval, Restoration.* MIT Press, Chapter MusicXML for Notation and Analysis, 113–124.
[8] Jim Jones, Diego de Siqueira Braga, Kleber Tertuliano, and Tomi Kauppinen. 2017. MusicOWL: the music score ontology. In *Proc. of the Intl Conf. on Web Intell. (WI).*
[9] Antti Laaksonen and Kjell Lemström. 2024. Advanced Polyphonic Music Pattern Matching Algorithms with Timing Invariances. In *Proc. of the Intl. Conf. Mathematics and Computation in Music (MCM).* 242–254.
[10] Anna Lubiw and Luke Tanur. 2004. Pattern Matching in Polyphonic Music as a Weighted Geometric Translation Problem. In *International Society for Music Information Retrieval Conference.*
[11] Neo Technology. 2026 (consult. date). Neo4j web site. www.neo4j.org.
[12] Neuma 2025. The NEUMA platform. http://neuma.huma-num.fr. (consult. date).
[13] Bryan Pardo and Manan Sanghi. 2005. Polyphonic Musical Sequence Alignment for Database Search. 215–222.
[14] Philippe Rigaux and Virginie Thion. 2024. Topological querying of music scores. *Data Knowl. Eng.* 153 (2024).
[15] Perry Rolland. 2002. The Music Encoding Initiative (MEI). In *Proc. of the Intl. Conf. on Musical Applications Using XML.* 55–59.
[16] Verovio 2026. Verovio web site. "https://www.verovio.org". (consult. date).