Article

A Framework for Content-based Search in Large Music Collections

Tiange Zhu [1], Raphaël Fournier-S'niehotta [1], Philippe Rigaux [1] and Nicolas Travers [2,1]

[1] CEDRIC Laboratory, CNAM Paris, France;
tiange.zhu@lecnam.net, fournier@cnam.fr, philippe.rigaux@cnam.fr
[2] Léonard de Vinci Pôle Universitaire, Research Center, Paris La Défense, France ; nicolas.travers@devinci.fr
* Correspondence: tiange.zhu@lecnam.net (T.Z.); nicolas.travers@devinci.fr (N.T.)
† This paper is an extended version of our paper published in ISMIR19.

Abstract: We address the problem of scalable content-based search in large collections of music documents. Music content is highly complex and versatile, and presents multiple facets that can be considered independently or in combination. Moreover, music documents can be digitally encoded in many ways. We propose a general framework for building a scalable search engine, based on i) a music description language which represents music content independently from a specific encoding, ii) an extendible list of feature-extraction functions and, iii) indexing, searching and ranking procedures designed to be integrated into the standard architecture of a text-oriented search engine. As a proof of concept, we also detail an actual implementation of the framework for searching in large collections of XML-encoded music scores, based on the popular ElasticSearch system. It is released as open-source in GitHub, and available as a ready-to-use Docker image for communities that manage large collections of digitized music documents.

Keywords: Music collections; Digital music encoding; Music Information Retrieval; Scalable and content-based Search

Citation: Zhu, T.; Fournier-S'niehotta, R.; Rigaux, P.; Travers, N. A Framework for Content-based search in Large Music Collections. *Big Data Cogn. Comput.* **2022**, *1*, 0. https://doi.org/

Received: 7 October 2021
Accepted:
Published:

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

1. Introduction

Search engines have become essential components of the digital space. They help to explore large and complex collections by retrieving ranked lists of relevant documents related to a query pattern. They rely on scalable indexing structures and algorithms that allow instant response to queries for web-scale collections [1]. Notable successes have been obtained for text-based documents, and extended to multimedia collections [2–4].

Compared to other media (text, image or even video), the research on content-based music information retrieval presents some specific challenges. Musical content is intricate, and hard to describe in natural and intuitive terms. Temporal aspects (tempo, metric, synchronisation) are a major source of complexity that complicate attempts to provide a synthetic representation. Moreover, musical contents are extremely versatile: from improvisation to highly constrained forms, from a single performer to a whole orchestra, from classical to popular music, there exists a wide range of facets that yield a boundless number of genres, styles and forms. Last but not least, periods and locations (of composition or interpretation) are other important aspects that increase the variability of the material.

Finally, when it comes to digital representations, one is confronted to highly diverse encoding paradigms. The *audio* format is the most common. It usually contains recordings of studio or live performances, and constitutes the basis of digital music markets, particularly with the advent of streaming distribution. On the other hand, *symbolic representations* aim at a structured description of musical pieces. The *MIDI* format encodes information related to the production of sound by a MIDI device [5]. *Music notation* is the most elaborated way of describing music at this symbolic level [6]. It has been traditionally used for engraving musical scores [7], but, since the advent of digital encodings such as the

**kern format [8] or XML-based variants (MusicXML [9], its next generation MNX [10], or MEI [11,12]), music notation can also be seen as a support for music information processing. **kern is for instance explicitly designed as a digital encoding of scores that feeds the music analysis modules of the Humdrum toolkit [13]. Large collections of digitally codified music scores are now available, either as results of long-running academic efforts [14,15], or as a side-effect of the generalized production of music scores with editing softwares that encode their documents in one of the above-mentioned formats (*e.g.*, MuseScore [16]). Such collections are examples of datasets where the music content is described in a structured and well-organized way, apt at supporting sophisticated computer-based operations.

To the best of our knowledge, however, most existing search tools for large music collections highly rely on metadata. This is the case for search engines incorporated in music streaming services like *Deezer* or *Spotify* [17], and for renowned digital music databases like *Discogs* [18] and *AllMusic* [19]. *Musixmatch* [20] allows lyrics search with access to libraries of major music streaming platforms. *Shazam* [21] allows searching audio recordings by indexing the fingerprints of files, and its result are therefore highly dependent on the specificities of audio music encoding. *SoundHound* [22] offers a *Query by Humming* [23] functionality that relies on the measurement of melodic similarity, thus it cannot search other aspects of music. The few approaches that address search operations applied to symbolic representation propose an exhaustive scan of the digital encoding, such as for instance the Humdrum tools based on Unix file inspections [13] or the search methods incorporated in the Music21 toolkit [24]. They do not scale to very large music datasets.

We expose in the present paper the design of a general framework for *scalable content-based search* in large digital collections of music documents. Here, *scalable* means a sub-linear search complexity, delivering very fast response time even in the presence of very large collections; search operations are *content-based* because they rely on a structured representation of music inspired by music notation principles, and can thus refer to specific aspects of a music document (*e.g.*, a melodic pattern in the violin part of a symphony); and finally our design addresses digital music documents, independently from a specific music representation, thanks to an intermediate step that extracts the structured content upon which all index/search/rank operations are based.

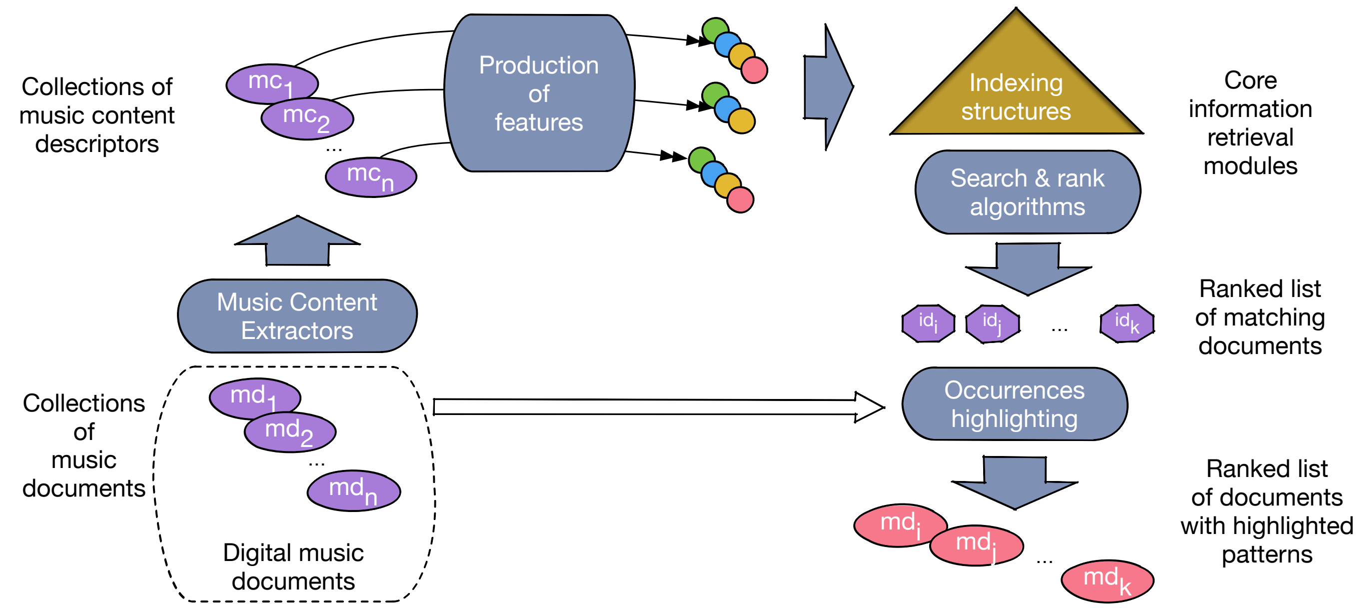


Figure 1. Overview of the architecture of our music search framework. Digital music documents (bottom left) undergo a series of transformations and are finally ranked according to a search pattern.

The proposed design is summarized in Figure 1. Initially, we deal with a large collection of digital music documents in audio, symbolic or other formats. A first step processes these documents by *extractors*, in order to obtain a structured representation, called *music content descriptor*, complying with a Music Content Model (MCM). We enter then in a more classical information retrieval workflow. First, features are produced from each descriptor. This step is akin to the pre-processing operations in standard text-based Information Retrieval (*e.g.,* tokenization, lemmatization, etc.) adapted to the characteristics of music representation. Those features must be encoded in a way that is compatible with functionalities of the core information retrieval modules: indexing, searching and ranking. Given a query pattern, they cooperate to deliver a ranked list of matching documents. The last step of this IR workflow identifies all the fragments of the retrieved document that match the query pattern, called *pattern occurrences*. This step is necessary for highlighting the matching patterns in the user interface.

We further position our work with respect to the state of the art in Section 2, and expose then our main contributions:

- A *Music Content Model*, or MCM (Section 3). It borrows from the principles of music notation, reduced to the aspects that are independent from presentation purposes (i.e., ignoring staves, clefs, or other elements that relate to the layout of music scores). Although strongly influenced by the Western music tradition, we believe that this model is general enough to represent a large part of the currently digitized music. We call *Music Content Descriptor* (MCD) a description of a music document according to this model. The model supports some major functionalities of a search engine, namely *transformations* corresponding to the classical linguistic operations in text-based search engines, and *ranking*.
- A set of *features* that can be obtained from a MCD thanks to the above-mentioned transformations. The features set presented in the current work (Section 4) is by no way intended to constitute a final list, and the framework design is open to the addition of other features like harmony, texture or timbre.
- The design of the core modules of a search engine, based on these features and dedicated to music retrieval (Section 5). They consist in *indexing*, *searching*, *ranking*, and on-line identification of fragments that match the query pattern.
- An actual implementation (Section 6), dedicated to XML-encoded musical scores' collections, that shows how to integrate these modules in a standard information retrieval system, with two main benefits: reduction of implementation efforts, and horizontal scalability.

Finally, Section 7 concludes the paper and lists some future extensions.

2. Related Work

Our approach relies on an abstract music content model. It consists of a tree-based decomposition of a music score that reflects its temporal organization. This draws heavily from [25–27], which introduced into the Music Information Retrieval literature some ideas and tools from the fields of databases systems and computer linguistics (*e.g.,* hierarchical decomposition of musical content and context-free grammars). Recently, [28] used a similar graph-based representation to study music similarity.

The process that extracts instances of our model from digital music documents depends on their specific representation. *Automatic Music Transcription* (AMT) applies to audio files (*e.g.,* either *pulse-code modulation* representation, or (un)quantized-MIDI) and produces symbolic data (generally quantized MIDI). Most of AMT methods nowadays use machine learning approaches [29,30], and deliver satisfying results in limited cases (mostly, monodic inputs). Research currently focuses on the difficult problem of polyphonic transcription. *Optical Music Recognition* (OMR) is an active field of research that studies tools and methods to extract music notation from an image (*e.g.,* a scan of a musical score) [31–33]). The quality of the results is highly dependent on that of the input image, but significant success have been obtained recently [34], event for degraded inputs (*e.g.,* manuscripts). Finally, the simplest content extraction situation comes when the digital document is itself in a structured format (whether **kern, MusicXML, or MEI), in which case a standard parsing followed by convenient filtering and structuring steps is sufficient.

Textual encoding of symbolic music representation is an attractive idea in order to use text algorithms. The *HumDrum* toolkit [35] relies on a specialized text format and adapts Unix file inspection tools for music analysis. Exact and approximate string matching algorithms for melody matching have also been used in *ThemeFinder* [36,37] or *Musipedia* [38]. Text-based operations raise the problem of independence with respect to the physical content encoding: it is a widely admitted principle, in the community that the result on text of a query should not be tied to a specific representation, but rather defined with respect to a "logical" data model. We proposed such a model in [39–41], and the design presented in the present paper relies on such a high-level representation.

Ranking musical pieces according to their *relevance* with respect to a query pattern is an essential part of an information retrieval system. In the MIR community, extensive studies have been devoted to music similarity over the last decades, with the goal of obtaining robust computational methods for evaluating the likeness of two musical sequences [42]. A major problem is that similarity judgments are highly dependent on both the particular aspects being compared and on the user taste, culture, and experience [43,44]. A recent survey [45] summarizes the recent trends observed in the SMS track of the MIREX competition. Our work proposes well-established similarity measures, based on edit distances, to support the ranking process. They could easily be replaced in the framework design by other ranking functions, as long as they can be evaluated on our music content descriptors. We believe, in addition, that using a hierarchical representation gives rise to a wider range of possibilities for evaluating similarities, such as for instance adding strong/weak beats as input parameters.

Developing search engines dedicated to musical content is a rather emerging topic, because it is only during the last decade that large collections of digital music have been produced and made widely available. [46] is a survey on pioneering works on music information retrieval systems, followed a few years later by a contribution detailing the "specifications and challenges" for Music Search Engines [47]. The Peachnote Music Ngram Viewer [48] was then developed, relying like our approach on *n*-grams and a symbolic input (with a piano keyboard interface), though the description of their method is not detailed. Note that the idea of splitting musical sequences in *n*-grams has been experimented in several earlier proposals [49–52], although not in the context of indexing. Other projects, like Probado or Vocalsearch, seem to have shared some features with our framework, but most of their details are no longer available. Modulo7 [53] is a promising search engine (currently under development), also offering an abstract representation of the music content. An index structure based on *n*-grams is described in [54], and extended in [55] with ranking procedures. The present paper further extends [55] with a full study that addresses all the aspects of the envisioned framework, along with a complete and publicly available implementation.

3. The Music Content Model

We now present the *Music Content Model* (MDM) which relies heavily on principles taken from music notation, seen as an expressive formal language that provides a powerful basis for modeling music content. The MDM gives an abstract vision of digital music documents as structured objects, and supports indexing and search functionalities developed in the forthcoming sections.

To state it in a nutshell, we model music information as a mapping from a structured temporal domain to a set of value domains, and call *music descriptor* a representation of this mapping as a structured object. The temporal domain is a hierarchical structure, called *rhythmic tree*, that partitions a finite time range in non-overlapping intervals. Each interval corresponding to a leaf of the rhythmic tree is associated to an atomic music event. The mapping therefore associates to each such interval the event value, taken from a domain which can be the domain of sounds, of syllables, or actually any domain that makes sense with respect to music description concerns: intensity, timbre, texture, or high-level concepts obtained from music analysis.

We start with the main domain of interest, the domain of sounds, and continue with the definition of music descriptors, along with the main operations. The model is illustrated with a first example: the German anthem, *Das Lied der Deutschen*, whose music was composed by *Joseph Haydn* in 1797 [56]. The notation of this example is shown on Fig. 2. Note that in the presentation that follows, we introduce the basic material from music theory, necessary and sufficient to understand the rationale of our design. References to authoritative sources are given for the interested reader.

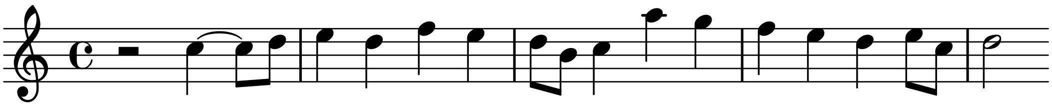


Figure 2. First notes of the German anthem, *Das Lied der Deutschen* by *Joseph Haydn* (1797).

3.1. The Domain of Sounds: Pitches and Intervals

The main domain to consider is that of sounds. A sound can be characterized by many properties, including intensity, timbre and frequency [57]. We only consider the characterization of sounds by their frequency in the modeling of our domain.

In the language of music notation, the frequency ranges approximately from 20 to 20,000 Hz. In Western music, a finite set of frequencies, or *pitches* [58], 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. In this designation, each pitch is referred to by a pitch class P (a letter A, B, C, D, E, F, or G) [59], an index I in $[1, 7]$, and an optional accidental a in $\{\sharp, \flat\}$. One obtains a set of *pitch symbols* of the form $P[a]I$.

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 Fig. 2 is a C4, followed by a D4, an E4, 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 rectangle.

Finally, a sound can be represented by one or several consecutive pitches, representing the same frequency level, which is then "tied" (graphically represented as curves over the heads, such as in the first measure of Fig. 2), we add the *continuation symbol* _ to our domain. We obtain the domain of musical symbols.

Definition 1 (Domain of musical symbols)**.** *The domain* ***Mus*** *of musical symbols consists of:*

1. *The set of pitch symbols* $P[a]I$, $P \in \{A, B, C, D, E, F, G\}, a \in \{\sharp, \flat\}, I \in [1, 7]$,
2. *The rest symbol, noted* r,
3. *The continuation symbol, noted* _ .

We will need some derived notions in our model. An *interval* is a distance between two pitches [60], physically characterized by the ratio of their respective frequencies. A ratio of 1 denotes a *unison*, a ratio of 2 an *octave*. The octave is the fundamental interval that structures symbolic music representation. Indeed, a pitch class contains all the pitches that are one or several octaves apart from one another: A4 is one octave above A3, and one octave below A5. The second component of a pitch designation, the index I, refers to a specific octave in the whole frequency range. The ISO standard assumes seven octave ranges numbered from 1 to 7.

In Western music notation, an octave range is divided in 12 *semi-tones*. This defines a scale, called *chromatic*, with 12 *steps*, corresponding each to exactly one semi-tone. The definition of *chromatic intervals* is therefore based on the number of semi-tones between the two pitches.

Another scale, called *diatonic* relies on the identification of 7 "natural" pitches [61] within and octave. Octave 4, for instance, contains the (natural) pitches {A4, B4, C4, D4, E4, F4, G4}. In the diatonic perspective, the twelve chromatic pitches are obtained by altering the natural ones. An alteration either adds (symbol ♯) of subtracts (symbol ♭) a semi-tone. Therefore A♯4 is one semi-tone above A4, and B♭4 one semi-tone below B4.

In the diatonic perspective, the distance between two pitches is nominal and based on the number of steps between the pitches in the diatonic scale, regardless of possible alterations. One obtains unisons (0 steps), seconds (1 step), thirds (2 steps), etc. The list of interval names (lower than an octave) is $\{unison, second, third, fourth, fifth, sixth, seventh, octave\}$.

To summarize, we can (and will) consider two definitions of intervals:

- A *chromatic interval* is the number of steps, negative (descending) or positive (ascending), in the chromatic scale, between two pitches.
- A *diatonic interval* is a nominal distance measuring the number of steps, descending or ascending, in the diatonic scale, between two pitches.

Diatonic and chromatic intervals are partially independent from one another: if we take two pairs of pitches, they might coincide in terms of their respective diatonic intervals, and differ on the chromatic ones, and conversely. Searches interpreted with respect to either of those two concepts gives distinct results.

3.2. Music Content Descriptors

We model music is a temporal organization of sounds inside a bounded time range. Notes cannot be assigned to any timestamp but fall on a set of positions that defines a discrete partitioning of this range. More precisely, this partition results from a recursive decomposition of temporal intervals, yielding a rhythmic organization which is inherently hierarchical.

In Western music notation, a music piece is divided in *measures* (graphically represented as vertical bars on Fig. 2), and a measure contains one or more *beats*. Beats can in turn be divided into equal units (i.e., sub-beats) [62,63]. Further recursive divisions often occur, generating a hierarchy of pulses called *metrical structure*. The *time signature*, a rational number (in our example, 4/4) determines the preferred decomposition. A 4/4 measure consists of 4 beats, and each beat is one quarter (graphically, a black note ♩) long. Still in the context of a 4/4 time signature, the preferred decomposition of a measure, is into 4 sub-intervals (some other partitions are possible, although less likely), beats are preferably partitioned in two quavers (graphically, a ♪), themselves (generally) partitioned in semi-quavers (𝅘𝅥𝅯), etc.

For other meters (*e.g.*, 3/4, 6/8), temporal decomposition follows different patterns. In all cases, the rhythmic decomposition rules can be expressed in a well-known formal language, namely *Context-Free Grammars* (CFG). In order to express decomposition preferences, they can be extended to *Weighted Context-Free Grammars* [26]. As an illustration, the following grammar $\mathcal{G} = (V, \mathbf{Mus}, R, S)$ is sufficient to model the rhythmic organization of our example, with time signature 4/4. The set of non-terminal symbols is $V = \{S, m, b, q\}$, where S (the initial symbol) denotes a whole music piece, m a measure, b a beat and q a quaver. The terminal symbols belong to **Mus**, the set of music symbols (Def. 1), and R is the following set of rules:

1. $R_0 : S \to m|m, S$ (a piece of music is a sequence of measures)
2. $r_1 : m \to b, b, b, b$ (a measure is decomposed in four quarter notes / beats)
3. $r_2 : b \to q, q$ (a beat is decomposed in two quavers / eighth note)
4. A set $\mathcal{R}^m$ of rules $R^v_e : v \to e$ where $e \in \mathbf{Mus}$ is a musical symbol.

Rule R_0 and the set $\mathcal{R}^m$ together determine the temporal structure of music: i) a time range in divided in equal-sized measures, and ii) events only occur at timestamps determined by a parse tree of the grammar. Unambiguous grammars that feature R_0 and $\mathcal{R}^m$ are called *music content grammars* in the following. Given a music content grammar, we can use its rules to build a hierarchical structure (a parse tree) that models the rhythmic organization of a sequence of musical events.

Definition 2 (Monodic content descriptor)**.** *Let $\mathcal{G} = (V, \textbf{Mus}, R, S)$ be a music content grammar. A (monodic)* content descriptor *is a parse tree of G. The inner nodes constitute the* rhythm tree*, and the leaves are the (musical)* events*.*

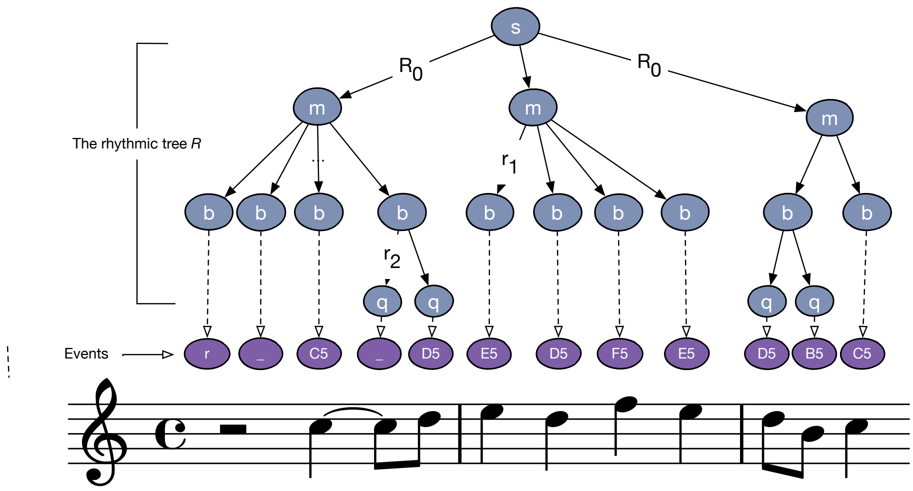


Figure 3. The content descriptor for the German anthem, with its events and the rhythmic tree.

Figure 3 shows the content descriptor of the initial measures of the German anthem. From a content descriptor it is easy to infer the following properties that will serve as a basis for the indexing process: pitch sequence, temporal partition, and event sequence.

Definition 3 (Pitch sequence)**.** *Let D be a content descriptor. The sequence of leaf nodes values in D is a string in $\textbf{Mus}^*$ called the* pitch sequence *of D and noted $PSeq(D)$.*

Given a time range I, a content descriptor D defines a partitioning of I as a set of non-overlapping temporal intervals defined as follows.

Definition 4 (Temporal partition)**.** *Let $I = [\alpha, \beta[$ be a time range and D a content descriptor. The* temporal partitioning *$P(I, D)$ of I with respect to D is defined as follows. Let N be a node in the rhythmic tree of D (recall that the rhythmic tree is D without the leaves level).*

1. *If N has no children, $P(I, N) = \{I\}$*
2. *If N is of the form $N(N_1, \cdots, N_i)$, I is partitioned in n sub-intervals of equal size $s = \frac{\beta - \alpha}{n}$ each: $P(I, N) = \{I_1, \cdots, I_n\}$ with $I_i = [\alpha + (i-1) \times s, \alpha + i \times s[$*

This partitioning associates to each internal node N of a content descriptor a non-empty interval denoted $itv(I, N)$ in the following and a duration denoted $dur(I, N)$. Each event (leaf node) covers the time interval of its parent in the rhythmic tree.

We will adopt the following convention to represent temporal values: the duration of a measure is 1, and the music piece range is n, the number of measures. Both the duration and interval of a node result from the recursive division defined by the rules. The duration of a half note for instance is $\frac{1}{2}$, the duration of a quaver is $\frac{1}{4}$, etc. The duration of a leaf node (event) is that of its parent in the rhythmic tree.

One can finally obtain the *event sequence* by combining both information.

Definition 5 (Event sequence)**.** *Let D be a content descriptor and $[L_1, \cdots, L_n]$ be the pitch sequence of D. Then the sequence $[(L_1, dur(L_1)), \cdots, (L_n, dur(L_n))]$ where we associate to each leave its duration is the* event sequence *of D, denoted $ESeq(D)$.*

Each element in $ESeq(D)$ associates a symbol from **Mus** and a duration. One obtains the sequential representation commonly found in music notation. An explicit representation of the hierarchical structure is, however, much more powerful than the sequential one. We can use the tree structure for various simplifications, compute similarity measures (see below), or infer strong or weak timestamps from their corresponding branch in the tree. More generally, this general framework allows deriving *features* from content descriptors by extracting, transforming, normalizing specific aspects pertaining to rhythm, domain values, or both.

3.3. Non-Musical Domains

This modeling perspective can be extended to other value domains beyond the class of music symbols. Consider the example shown on Fig. 4, the same German anthem enriched with lyrics. We can model this mixed content with two content descriptors over distinct values domains (i.e., terminal symbols sets). The first is derived from a grammar where terminal symbols taken from **Mus**, as before, and the second one taken from syllables.

Figure 4. The German anthem, with lyrics associated with the music.

The content descriptor for the lyrics part might be different from that of the melodic part (Fig. 5). Indeed, a same syllable may extend over several notes (a feature called *melism*, see 'al–les' Fig. 5). Less commonly, but also possible, several syllables may be sung on a single note.

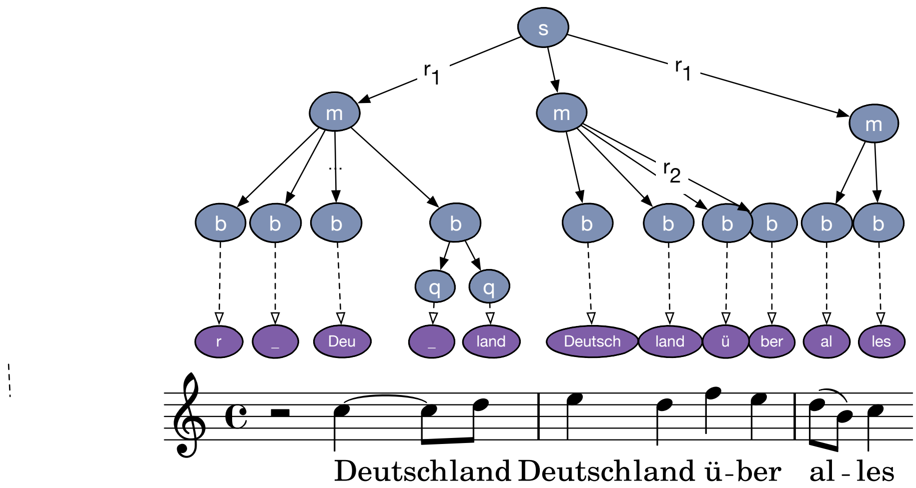


Figure 5. The content descriptor of the syllabic part of the German anthem.

This generalized model therefore covers any mapping of a time range structured by a CFG to a value domain: we illustrated it so far with pitches and syllables, but chords, textures, or other types of annotation can fit in this framework.

3.4. Polyphonic Music

So far we only considered monodic music (a single flow of events). The representation of polyphonic music simply consists of a set of monodic content descriptors sharing a same grammar.

Definition 6 (Polyphonic content descriptor)**.** *Given a music content grammar $\mathcal{G}$, a* (polyphonic) content descriptor *is a set of parse trees of $\mathcal{G}$ such that the number of derivations of rule R_0 (in other words, the number of measures) is constant.*

Fig. 6 gives an illustration (the same theme, with a bass part added). In terms of music content, it can be represented by two content descriptors derived from the same grammar, and with the same number of measures. *Synchronization* properties (the fact for instance that the time range of two events overlap) can easily be inferred. Harmonic features (*e.g.,* chord names) could therefore be obtained from the content descriptors, and added to the framework. The same holds for musical properties such as, *e.g.,* timbre [64] or texture [65], as long as they can be modeled and derived computationally.

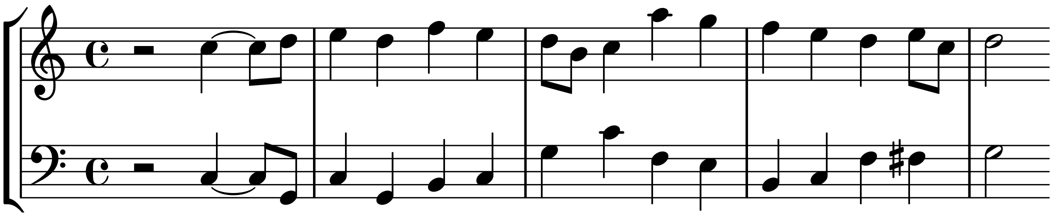


Figure 6. German anthem, with two voices.

From now on, we will assume that a polyphonic descriptor can be obtained from every music document (we refer to Section 6 that describes our implementation, and to the state-of-the-art in Section 2). Content descriptors constitute the input for the features production described in the next section. A generalization to polyphonic descriptors as sets of features is immediate.

4. Offline Operations: Features and Text-Based Indexing

We now present a list of features that can be produced from a music content descriptor: a chromatic interval feature (CIF), a diatonic interval feature (DIF), a rhythm feature (RF), and a lyric feature (LF). This list is not closed. As explained above, features pertaining to other aspects of music representation (*e.g.,* harmonic) or features obtained from an analytic process may be added, as long as they can be derived from our description model.

The features presented below are designed to be integrated in a text-based search engine. This requirement is motivated by easiness of implementation. Should a multimedia search engine be available off-the-shelf with metric-based access methods (for instance multidimensional search trees [2]), this constraint could be relaxed. Each feature type must therefore fulfill the following requirements:

- There exists an *analyzer* that takes a content descriptor as input and produces a feature as output.
- There must exist a *serialization* of a feature as a character string, which makes possible the transposition of queries to standard text-based search supported by the engine.
- Finally, each feature type must be equipped with a *scoring function* that can be incorporated into the search engine for ranking purposes.

We will use the famous song *My way* [66] as an example to illustrate our features (see Fig. 7). The song is the English version of the French song *Comme d'habitude* [67], written by Claude François and Jacques Revaux (1967). The English lyrics are by Paul Anka (1969).

Figure 7. Main example (*My way*, first phrase)

The content descriptor of this fragment is illustrated by Fig. 8.

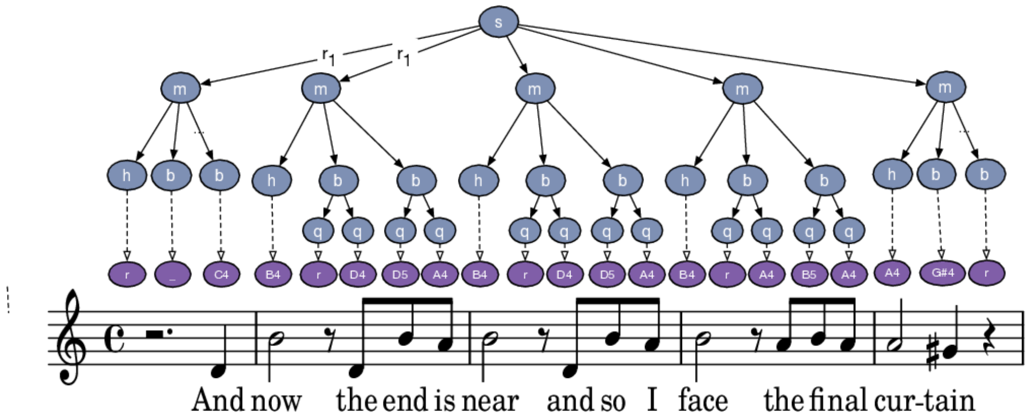


Figure 8. Content descriptor of *My way*

4.1. *Chromatic interval feature*

The feature analyzer A_{CIF} relies on the following simplification of a pitch sequence:

1. All repeated values from $PSeq(D)$ are merged in a single one.
2. Rest and continuation symbols are removed.

One obtains a simplified descriptor that essentially keeps the sequence of non-null intervals. Fig. 9 shows such a sequence, resulting from the analysis of *My way*. Note that the two consecutive `A4`s near the end have been merged, and all rests removed.

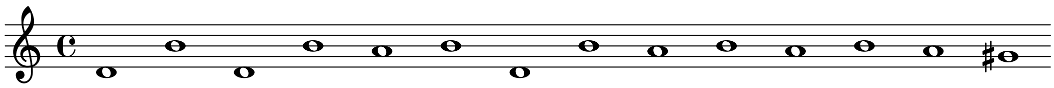


Figure 9. *My way*, after the feature extraction by the appropriate analyzer.

Definition 7 (Chromatic Interval Feature)**.** *Given a content descriptor D, the* chromatic interval feature *(CIF) $A_{CIF}(D)$ is the sequence of the chromatic intervals values (number of chromatic steps) between two consecutive pitches in the simplification of $PSeq(D)$.*

When the CIF analyzer A_{CIF} is applied to the sequence of Fig. 9, one obtains the following feature.

$$< 9, -9, 9, -2, 2, -9, 9, -2, 2, -2, 2, -2, -1 >$$

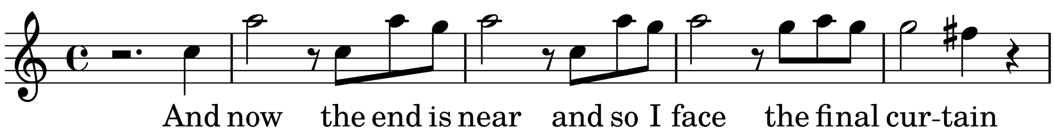


Figure 10. *My way*, transposed.

It is worth mentioning that we obtain the same CIF from initially distinct music descriptors. Fig. 10 shows a *transposed* version of *My way*, more suitable to a female voice (say, Céline Dion rather than Franck Sinatra). The CIF is invariant. The feature is also robust with respect to rhythmic variants. Fig. 11 shows the initial – French – version of the tune, sung by Claude François. The lyrics in French imply a slightly distinct rhythmic structure. However, the sequence of intervals remains identical, and so does the CIF.

Figure 11. French version of *My way* (*Comme d'habitude*, first phrase).

We can therefore conclude that the descriptors shown in Figures 7, 10, and 11 *match* with respect to their respective chromatic features. The matching of two descriptors is highly dependent on the analyzer. Among other possible features, we could have taken the sequence of pitch names, in which case transposed scores would not match. The precision would likely be higher but we would miss results that seem intuitive.

Another feature would accept unisons (i.e., repeated notes yielding intervals with 0 semi-tones). Then, in our example, the French version (Fig. 11) would no longer match with the English version of *My way*.

Each analyzer determines a balance between precision and recall. Fig. 12 shows another example of descriptor that matches with the previous ones with respect to the CIF feature. It seems clear that it is quite *rhythmically* far from the standard tune and that, at the very least, it should not be given the same score in the result set than the previous ones. The ranking function should yield a low similarity factor for such descriptors that match at the value (melodic) level but highly differ at the rhythmic level. We propose such a ranking function in Section 5.3.

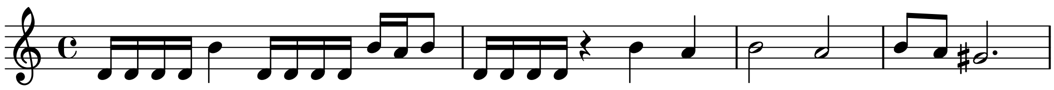


Figure 12. *My way*, rhythmically distorted.

4.2. Diatonic Interval Feature

Let us continue with our favorite tune, *My Way*. We keep the same simplification phase already used for A_{CIF}. Fig. 13 shows the second phrase, which slightly differs from the first one. If we compute the chromatic interval feature, one obtains the following sequence:

$$< 8, -8, 8, -1, 1, -8, 8, -1, 1, 4, -2, -5, 3, -1 >$$

which is distinct from that of the first phrase (see above).

Figure 13. *My way*, second phrase.

If we adopt the diatonic perspective, we observe that the second phrase starts with a 5-steps diatonic interval (from E4 to C5), continues with a descending one-step (from C5 to B4), etc. Therefore, the first interval of the first phrase and of the second phrase do match in a diatonic interpretation context: they are both sixths, major in the first case, minor in the second one. So does the second interval (a second, minor in the first case, major the second case). We can conclude that both phrases, in the diatonic perspective, are similar, and we introduce the Diatonic Interval Feature to capture this interpretation.

Definition 8 (Diatonic Interval Feature)**.** *Given a content descriptor D, the* diatonic interval feature *(DIF) $A_{DIF}(D)$ is the sequence of diatonic interval names between two consecutive pitches in the simplification of $PSeq(D)$.*

Assuming that the set of interval names is {U(nison), S(e)c(ond), T(hird), Fo(urth), Fi(fth), Si(xth), Se(venth) and O(ctave)} and that an ascending interval is coded with a $+$, a descending one with a $-$, one may apply this definition to the descriptor of Fig. 9. The following sequence is obtained:

$$< Si+, Si-, Si+, Sc-, Sc+, Si-, Si+, Sc-, Sc+, Si-, Si+, Si-, Si- >$$

The first and second phrases of *My way* match with respect to this feature, and continue to match with any transposition (Fig. 10) or rhythmic variants (Fig. 11).

4.3. Rhythmic Feature

So far we have built the features on the event values associated to the leaves of content descriptors. We now focus on the rhythmic information provided by the rhythmic tree in a music content descriptor.

An immediate thought would be to serialize the rhythmic tree using some nested word representation. There are at least two downsides in doing so:

1. Rhythmic perception is essentially invariant to homomorphic transformations: doubling both the note durations *and* the tempo results in the same music being played.
2. The rhythmic tree provides a very elaborated representation of the rhythmic organization: putting all this information in a feature would favor a very high precision but a very low recall.

As in the case on melodic description, we therefore adopt a simplified rhythmic representation, and resort to the ranking step to favor the result items that are closer to the query pattern.

Given a content descriptor R, its *temporal partition* (see Def. 4) gives the respective durations of the events. Consider once again the first phrase of *My way* (Fig. 7), ignoring the initial rest. It starts with a quarter note, followed by a half-note: the ratio (i.e., the multiplication to obtain the second duration value from the first one) is 2. Then comes a 1-eighth duration, hence a ratio equal to $\frac{1}{8}$, followed by three eight-notes, hence three times a neutral ratio of 1, etc. We adopt the sequence of these ratio as the description of rhythm.

Definition 9 (Rhythmic feature)**.** *Given a content descriptor D and its leaves* $[L_1, L_2, \cdots, L_n]$, *the* rhythmic feature *(RF)* $A_{RF}(D)$ *is a sequence* $[r_1, \cdots, r_{n-1}]$ *such that* $r_i = \frac{dur(L_{i+1})}{dur(L_i)}, \forall i \in [1, n-1]$.

The rhythmic feature of the first phrase of *My way* (ignoring the initial rest) is

$$< 2, \frac{1}{8}, 1, 1, 8, \frac{1}{8}, 1, 1, 8, \frac{1}{8}, 1, 1, 8, \frac{1}{2} >$$

4.4. Lyrics Feature

The lyrics feature (LF) is the simplest one: it consists of the text of the tune (if any exists). Since the feature contains purely textual information, it is subject to the traditional transformations (tokenization, lemmatization, etc.) operated by search engines.

4.5. Text-Based Indexing

Each of the previous feature is a (potentially long) sequence of values $[v_1, v_2, \cdots, v_k]$. In order to adapt this representation to the encoding expected by a search engine, we compute the list of n-grams $\{< v_i, \cdots, v_{i+n-1} >, i \in [1, k-n+1]\}$, where n, the n-gram size, is an index configuration parameter (we use $n = 3$ in our implementation). If, for instance, the sequence of values is <6,-3,-3,1,2,-2>, the list of 3-grams is [<6,-3,-3>, <-3,-3,1>, <-3,1,2>, <1,2,-2>].

Each n-gram is then encoded as a character string which constitutes a *token*. These tokens are finally concatenated in a text, separated by white spaces. Some additional encoding might be necessary, depending on the specific restrictions of the search engine, to avoid misinterpretation of unusual characters (for instance, the minus sign can be encoded as `m`), and value separators in n-grams must be chosen with care.

For instance, assuming that i) the character `m` is substituted to the minus sign, and ii) `X` is used as a separator, one would submit the following text to the engine:

```
6Xm3Xm3 m3Xm3X1 m3X1X2 1X2Xm2
```

One obtains a standard textual representation that can be right away submitted to the indexing module of the search engine.

4.6. A Short Discussion

So far, we presented a set of features that all relate to the monodic aspect of a music content. Some are mathematically founded (chromatic, rhythm), others are application-dependent (diatonic). They illustrate the design of our search framework as a producer of features derived from a normalized, high-level music content description. They all result in a linear representation, akin to be assimilated to textual data in a standard search engine.

Another design choice is to simplify the feature representation so that it favors recall over precision. Since a feature captures only one aspect of the content (either rhythmic, or melodic-based), two descriptions that are close with respect to this aspect, but highly different with respect to another, might match in spite of important differences. The matching-based retrieval is designed as a first step operated to filter out a large part of the collection, and completed with a scoring function (to be described next) that top-ranks relevant music documents.

5. Online Operations: (Scalable) Searching, Ranking, Highlighting

We now turn to the operations that occur during the query processing phase. *Searching* operates by applying to the query pattern the same analyzer as those used for the targeted feature. The matching is then computed thanks to the scalable text-search mechanisms supplied by standard text search engines.

The difficult part of the process is the *ranking* of the query result. The default ranking functions of a text-based information retrieval system would yield meaningless results if applied to our features. We therefore define and plug our own set of ranking functions, the description of which constitutes the major part of the present section.

5.1. Searching

A *query pattern* q (or pattern in short) is a pair (P, FT) where P is either a content descriptor or a set of keywords, and FT is the feature type (CIF, DIF, RF, or LF – the latter being required when P consists of keywords). In the following, we focus on musical patterns since lyrics can be treated as standard text.

Definition 10 (Matching)**.** *Let* $q = (P, FT)$ *be a query pattern, with* $FT \in \{CIF, DIF, RF\}$*,* A_{FT} *be the analyser associated to* FT*, and* D *be a content descriptor. Then* q matches D *if and only if there exists at least one substring* F *of* $A_{FT}(D)$ *(called* fragment *thereafter) such that* $A_{FT}(P) = F$*.*

Assume for instance that the user searches for *My way* and submits the search pattern P of Fig. 14 with the feature type CIF. The sequence $A_{CIF}(P)$ is:

$$< 9, -2, 2, -2 >$$

which (after n-gram encoding) is a sub-string of the CIF for the descriptors of Fig. 7, 11 and 12.

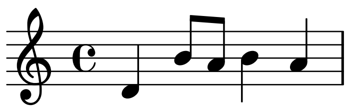


Figure 14. A pattern, matching a fragment of *My way*.

Definition 10 extends naturally to polyphonic music: a polyphonic descriptor M *matches* a query pattern (P, FT) if and only if, for *at least* a content descriptor D in M, and *at least* a substring F of $A_{FT}(D)$, $A_{FT}(P) = F$. The matching fragments are called the *matching occurrences* of M.

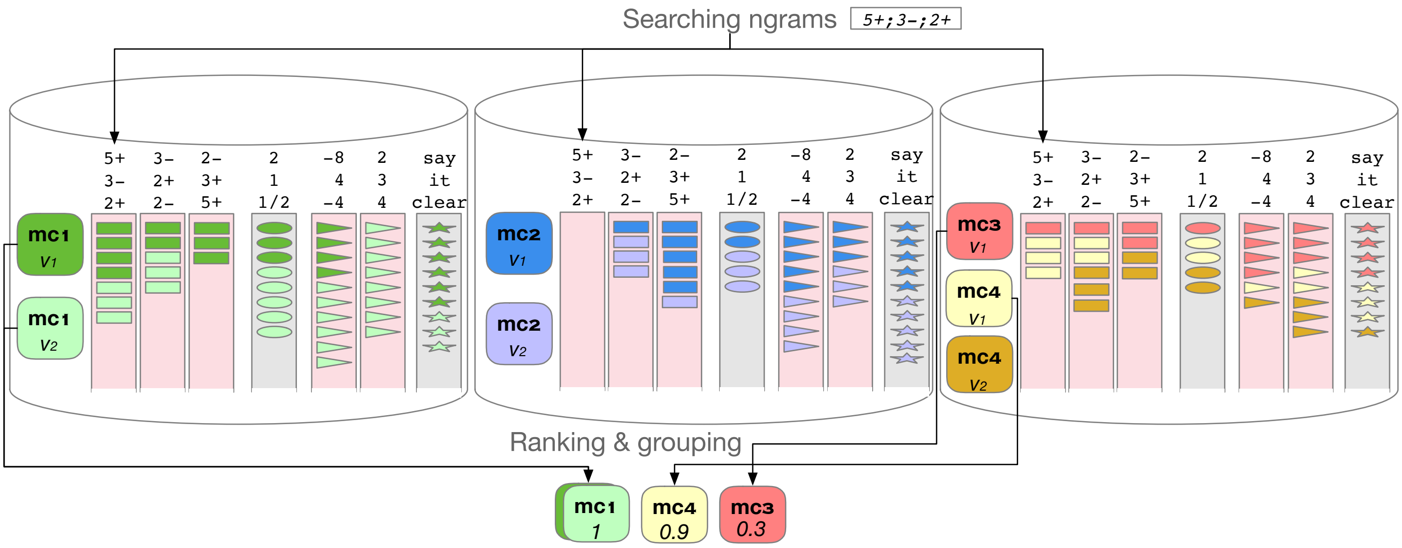


Figure 15. Data organization in a distributed search index and query processing.

5.2. Scalability

It follows from the previous definition that the matching operation is natively supported by text-based search engines. Furthermore, carrying out this operation is *scalable* because it can be processed in parallel over the participating server nodes in a distributed setting.

We illustrate the distributed processing with the example of Fig. 15, assuming a distributed setting with three servers. The figure shows four musical document $\{mc_1, mc_2, mc_3, mc_4\}$. Each document except mc_3 is polyphonic, with two monophonic descriptors. Documents are spread on the 3 servers: mc_1 on server 1, mc_2 on server 2 and $\{mc_3, mc_4\}$ on server 3.

On each server, for each type of descriptor, there is a list L_{ng} for each indexed n-gram ng: on Fig. 15 we show rectangles for CIF, circles for DIF, triangles for RF and stars for LF. Each list L_{ng} stores, in order, the position of ng in each descriptor of the local documents. For document mc_1 for instance, n-gram `5+;3-;2+` appears four times for the first descriptor, and three times for the second one.

At search time, the pattern is n-gram encoded as described above, and this encoding is submitted as a *phrase queries* to the search engine. A so-called "phrase query" retrieves the documents that contain a list of tokens (n-grams) appearing in a specific order. The query is sent to each server, and the servers carry out in parallel the following operations: 1) scan the list for each n-gram of the query and retrieve the matching descriptors (here $[mc_1, 1), (mc_1, 2), (mc_3, 1), (mc_4, 1)]$), 2) check that the positions correspond to the n-gram order in the pattern, 3) apply the ranking function locally, and 4) groups descriptors by documents to keep the best score.

All these steps, except the last one, operate at the document level and can therefore be processed in parallel on each participating server. The final ranked list is obtained by merging (in time linear in the size of the global result) the local results.

5.3. Ranking for Interval-Based Search

We first describe the ranking for interval-based features (i.e., Chromatic Interval Feature and Diatonic Interval Feature). Given a set of descriptors that match a pattern P, we now want to sort them according to a *score*, and rank first the ones that are closest to P. Since matching occurs on the melodic part, we want to rank on the rhythmic one. Referring to the pattern of Fig. 14, fragments from Fig. 7, 11 should be ranked first, whereas that of Fig. 12 should be ranked last because it greatly differs from the formers rhythmically.

We therefore compute a score based on the rhythmic similarity between the query pattern P and the matching subtree(s) in each descriptor of the result set. We base the computation on the following important observation: since the pattern and the descriptor share a common part of their melodic features, they have a similar structure that can be exploited. To state it more formally, since $A_{FT}(P) = F$, F being a fragment of $A_{FT}(D)$, there exists a sequence of identical non-null intervals in both P and D. Each interval is represented in D or P by a list of events that we call a *block*. More precisely:

Definition 11 (Block). *Let $F =< I_1, \cdots, I_n >$ be a fragment of $A_{FT}(D)$ for some descriptor D. By definition of the analyzer A_{FT}, each interval $I_i, i \in [1, n]$ in F corresponds to a sub-sequence $< p^i_1, e^i_2, \cdots, e^i_{k-1}, p^i_k >$ of $ESeq(D)$ such that:*

- *p^i_1 and p^i_k are two distinct non-rest values, and $interval(p^i_1, p^i_k) = I_i$*
- *each $e^i_l, l \in [2, k-1]$ is either a rest, or a pitch such that $e^i_l = p^i_1$*

We call $B_i =< p^i_1, e^i_2, \cdots, e^i_{k-1} >$ the block *of I_1 in D.*

The concept of block is illustrated by Fig. 16 for the descriptors of Fig. 7, 11 and 12 matching the pattern of Fig. 14.

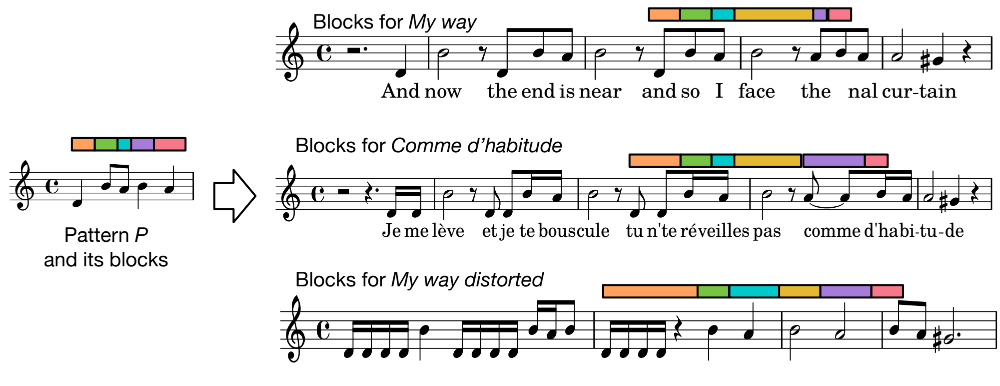


Figure 16. Blocks, in several fragments matching the pattern displayed in Fig. 14.

If a descriptor D matches a pattern P with respect to a feature type $FT \in \{CIF, DIF\}$, we can constitute a sequence of pairs $(B^P_i, B^D_i), i \in [1, n]$ of blocks representing the same part of the melodic query. We can therefore reduce the scoring problem to the evaluation of the rhythmic similarity internal to each pair.

Rhythmic similarity is a specific area of computational musicology which have been the subject of many studies [68–70]. A prominent trend is to rely on *edit distances* [71] applied to rhythms represented as sequences. Since we represent rhythm as trees, we rather use a *tree-edit distance* that operates on the rhythmic tree part of a music descriptor.

A tree-edit distance between two trees T_1 and T_2 is based on a set of transformations (called "edit operations"), each associated to a cost. The distance is defined as the sequence of transformations from T_1 and T_2 that minimizes the overall cost. Standard operations are *insert* (a node), *delete* (a node) or *replace*.

In our case, we are restricted to the parse trees of the music content grammar $\mathcal{G}$. Any transformation applied to a parse tree must yield a parse tree. We therefore accept the following list of edit operations.

1. *For children of the root*: insert / delete / replace a measure.
2. *For all other nodes N*: either insert a subtree by applying a rule from $\mathcal{G}$ to the non-terminal symbol N, or delete the subtree rooted at N.

The *cost* of each operation is the duration of the modified node. The cost of replacing / inserting / deleting a measure is 1, the cost of inserting / deleting a subtree rooted at a node labeled h (half note) is $\frac{1}{2}$, etc. Intuitively, the cost of an operation if the duration of the interval modified by the operation: the smaller the modification, the smaller the cost.

Definition 12 (Rhythmic similarity)**.** *Given two descriptors D_1 and D_2, the* rhythmic similarity *$Rsim(D_1, D_2)$ between D_1 and D_2 is the tree-edit distance is the minimal cost sequence of parse-tree edit operations that transforms the rhythmic tree of D_1 to the rhythmic tree of D_2.*

The rhythmic distance between D_1 and D_2 is $Rdist(D_1, D_2) = 1 - Rsim(D_1, D_2)$.

Computing the tree-edit distance is usually achieved with a dynamic programming algorithm. The two best known algorithms [72,73] run in time quadratic in the input size.

Fig. 17 shows the rhythmic trees for the pattern and the first block of the three matching fragments. The edit operations to obtain the rhythmic tree of the descriptors consist of one node insertion (*My way*), two insertions (one beat and two quavers, *Comme d'habitude*), and finally five insertions for *My way distorted*.

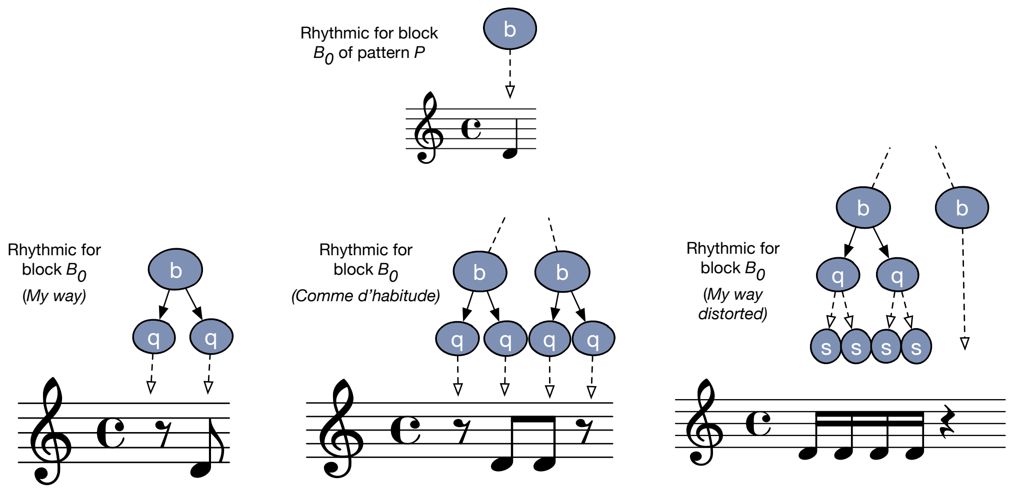


Figure 17. The rhythmic trees for the first block of each descriptor

The ranking function takes as input a pair of descriptors and outputs a score. It computes the alignment of blocks and sums up the distance between their rhythmic trees, obtained by the rhythmic distance *Rdist* (Def. 12).

Algorithm 1 Rhythmic Ranking

procedure RHYTHMRANKING(D_1, D_2)
Input: D_1, D_2, such that $A_{FT}(D_1)$=$A_{FT}(D_2)$
Output: a score $s \in [0, 1]$
    $s \leftarrow 0; < (B_0^1, B_0^2), \cdots, (B_n^1, B_n^2) > \leftarrow getBlocks(D_1, D_2)$
    for $i := 0$ **to** n **do** ▷ Loop on the pairs of blocks
        $s \leftarrow s + Rdist(B_i^1, B_i^2)$
    return s/n

The cost of the *getBlocks* part is linear in the size of D_1 and D_2, but computing the tree-edit distance is quadratic. There exists a simplified function that we use in our implementation: it simply cumulates the delta of the block duration within a pair. This simplified version reflects the insertion or deletion of nodes; however it ignores the internal structural changes. Applied to the trees of the pattern and *My way distorted* for instance, the simplified version does not measure the difference in the first beat (one quaver versus 4 repeated 16th-notes).

Algorithm 2 Simplified Rhythmic Ranking

1: **procedure** SIMPLERHYTHMRANKING(D_1, D_2)
2: **Input:** D_1, D_2, such that $A_{FT}(D_1)$=$A_{FT}(D_2)$
3: **Output:** a score s
4: $s \leftarrow 0; < (B_0^1, B_0^2), \cdots, (B_n^1, B_n^2) > \leftarrow getBlocks(D_1, D_2)$
5: **for** $i := 0$ **to** n **do** ▷ Loop on the pairs of blocks
6: $s \leftarrow s + |dur((B_i^1) - dur(B_i^2)|$
7: **return** s

The running time of the approximate ranking function is linear in the size of the descriptors.

5.4. Ranking for Rhythmic-Based Search

Let $(P,' RT')$ be a query on rhythmic features. A set of descriptors matching with P are retrieved. Since matching occurs on the rhythmic part, we want to sort the result set on their melodic similarity. Our implementation relies on the classical Levenshtein distance [74], as it is one of the most commonly used metrics to measure melody similarity in state-of-the-arts [75]. Note that edit distance is our current choice for this paper, yet the ranking algorithm could be substituted in another version of implementation.

Let F_1 be the CIF extracted from the query pattern P, and F_2 be the CIF extracted from matching parts in the descriptor, and F_i^1 represents for the ith element in F_1, while F_j^2 refers to the jth element in F_2. Thus, the *score* represents for the cost of converting F_2 into F_1, with three types of operations: deletion, insertion, and replacement. Since that each operation edits only one element in a CIF sequence, the alignment cost are all considered as 1.

The alignment cost of converting the sequence of first i elements of F_1 into the sequence of first j elements of F_2 is:

$$aligncost(F_i^1, 0) = i \qquad \text{if } 1 \leq i \leq n$$

$$aligncost(0, F_j^2) = j \qquad \text{if } 1 \leq j \leq m$$

$$aligncost(F_{i-1}^1, F_{j-1}^2) \qquad \text{if } F_i^1 = F_j^2$$

$$aligncost(F_i^1, F_j^2) = \min \begin{cases} aligncost(F_{i-1}^1, F_j^2) + 1 & \\ aligncost(F_i^1, F_{j-1}^2) + 1 & \text{if } F_i^1 \neq F_j^2 \\ aligncost(F_{i-1}^1, F_{j-1}^2) + 1 & \end{cases} \tag{1}$$

If there are n elements in F_1 and m elements in F_2, the *score* is $align(F_n^1, F_m^2)$. The final *score* is divided by n to normalize to the range $[0, 1]$.

Algorithm 3 Interval-based Ranking

procedure ItvRanking(F_1, F_2)
Input: F_1, F_2 ▷ A pair of CIF
Output: a score $s \in [0, 1]$
  for $i := 0$ **to** n **do** ▷ Loop on the elements of F_1
    for $j := 0$ **to** m **do** ▷ Loop on the elements of F_2
      if i = 0 **then**
        $cost[i, j] \leftarrow j;$
      else if j = 0 **then**
        $cost[i, j] \leftarrow i;$
      else if $F_1[i-1] = F_2[j-1]$ **then**
        $cost[i, j] \leftarrow cost[i-1, j-1];$
      else
        $cost[i, j] \leftarrow 1 + min(cost[i-1, j], cost[i, j-1], cost[i-1, j-1]);$
  return $cost[n, m]/n$

5.5. Finding Matching Occurrences

Once matching descriptors have been extracted from the repository, it is necessary to identify the sequences of events that match the pattern. Since both the pattern P and the feature are encoded as n-grams, the matching operator is able to return the sequence of n-grams in the feature that match P. This functionality is actually natively supplied by search engines, and commonly called *highlighting*.

Assuming that we get the sequence of matching n-grams, the problem is therefore reduced to identifying the events that yielded each n-gram during the analysis phase. Since we generally cannot inverse the analyzer, we must keep a correspondence table that associates to each n-gram the sequence of events it originates from.

Definition 13 (Reverse Analysis Table)**.** *Let D be a descriptor and FT a feature type. The Reverse Analysis Table RAT) (RAT) of D is a 2d table which gives for each n-gram ngr the list of events $e_i \in [1, n]$ in D for which $A_{FT}(e_1, \cdots, e_n) = ngr$.*

The RAT must be stored in the system and used on the result set. Given the sequence of matching n-grams $[g_1, \cdots, g_k]$ obtained from the search engine, we compute the union $RAT[g_1] \cup RAT[g_2] \cdots \cup RAT[g_k]$ and get the sequence of events matching the patterns.

6. Implementation

In this section, we detail an implementation of our proposed framework for symbolic music collections, *i.e.*, music in a notation-based format such as MIDI, XML and MEI. Since the core musical elements such as structure, melody and rhythm are represented in symbolic music, it is straightforward to develop an extraction of music content descriptors.

We offer a publicly available Docker image at https://hub.docker.com/repository/docker/traversn/scoresim, for the community to experience the proposed search engine. The code is in open access on Github (Components implementation: https://github.com/cedric-cnam/scoresim) under the GNU General Public License v3.0.

In the remainder of the section, we first present the global architecture, before delving into some specific components: descriptor extraction and feature production, integration into a standard text-based Information Retrieval system (centered around ElasticSearch) with several search modes available, customized highlighting and ranking procedures. We also showcase some functionalities of our system, taking advantage of the existing Neuma platform [76] (*e.g.*, Graphical User Interface, and large corpora).

6.1. Global Architecture

The architecture of our Docker server is illustrated in Figure 18. The main components are: i) an ETL (Extract/Transform/Load) process that receives music documents and produces their musical features, ii) an *Elasticsearch* server that indexes music features,

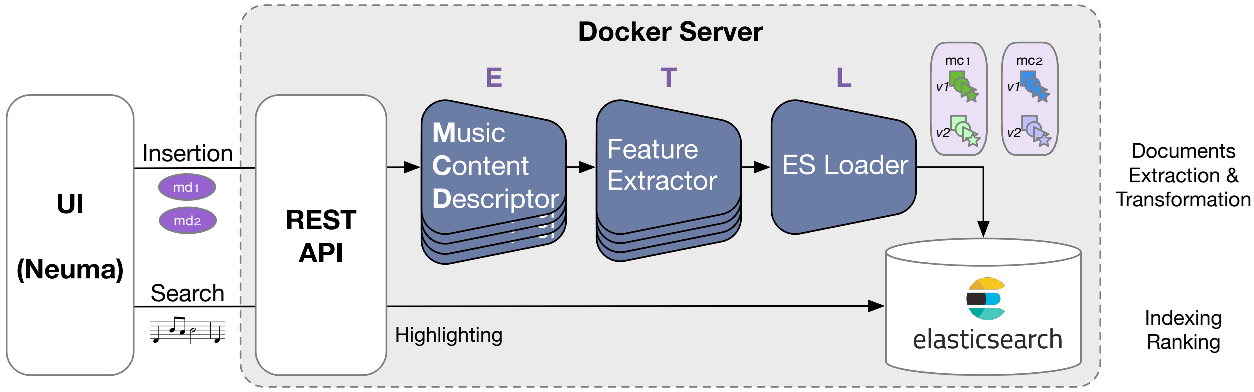


Figure 18. A global view of our music search architecture, for symbolic music.

supports searches and ranks results, and iii) implementation of several utility functions, including the matching occurrence identification. All these modules are written in Python. Once instantiated, the server communicates via a REST API which supplies insertion and search services. An external application (such as Neuma) can rely on this API to integrate a search module.

Elasticsearch [77] is a tunable search engine which provides several interesting features fitting our needs. It supports scalable data management based on a multi-server architecture with collections sharding, a rich query language, and the capability to tune the scoring function. Note that these features are shared with other search engines such as for instance Solr [78] or Sphinx [79]. The design of our framework relies on its ability to exploit these standard functionalities. Any of the above search engine would be a suitable candidate for supporting our solution and supply a scalable search operation without any further implementation. The main difficulty lies in the integration of an ad-hoc scoring function.

The server receives music documents via its REST API. Each document is then submitted to a pre-processing phase composed of three steps: Extract, Transform and Load. The **Extract** phase produces the music content descriptors (see Section 3). A specific extractor is required for each input format. In the case of XML-encoded scores, there exists ready-to-use toolkits such as Musicc21 [80] for parsing the input, accessing relevant data, and structuring this data according to our model. A content descriptor itself is an implementation of our tree-based representation, along with the production of derived representations (pitch sequences, distance operators, etc.). In general, the music document is polyphonic, and we obtain a set of monodic content descriptors.

The **Transformation** step produces features from each monodic content descriptor. We implemented all the features described in Section 4. Finally, the **Loading** steps sends the n-gram encoded features to Elasticsearch. For scalability reasons, we create one individually indexed document for each single monodic descriptor. This favors parallelism, but requires an aggregation at the document level at the end of the search process.

An example of indexed document is given below. It is identified by the pair `(doc_id, descr_id)`, the latter being typically the *voice ID* found in an XML encoding for such pieces. All features are encoded as 3-grams in our implementation. With each feature comes a *RAT* field that keeps the correspondence between each n-gram and the list of elements in the original document.

```
{
    "_id" : "doc_id:descr_id",
    "chromatic" : "7+;3-;2+; 3-;2+;1+; 2+;1+;1-; 1+;1-;5+;...",
    "RAT_chromatic" : {[...]},
    "diatonic" : "Fi+;T-;Se+ T-;Se+;Se+ Se+;Se+;Se- Se+;Se-;Fo+...",
    "RAT_diatonic" : {
        "Fi+;T-;Se+": [1, ...],
        "T-;Se+;Se+": [2, ...],
```

```
        ... },
    "rhythmic" : "(1)(1/2)(1) (1/2)(1)(2) (1)(2)(1/2) (2)(1/2)(1)...",
    "RAT_rhythmic" : {[...]},
    "lyrics" : "And now, ....",
    "RAT_lyrics" : {[...]}
}
```

Indexed documents are sent to Elasticsearch which builds the full-text indexes on features, and supplies text-based search operations. New feature extractors could easily be integrated to the system by adding new fields for each indexed document.

6.2. Query Processing

A query is submitted to the server as a pattern P along with the feature type T. We accept the *Plaine and Easie* coding formats for P. From this encoding a content descriptor $D = extract(P)$ is built using a dedicated extractor, and a feature of type T is obtained through the standard feature production function. This feature is n-gram encoded and submitted to ElasticSearch as part of a "*match_phrase*" query. This is illustrated by the following query from the ElasticSearch Domain Specific Language (https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl.html), showing a *match_phrase* query with a chromatic feature with two 3-grams encoding the following chromatic fragment: two ascending semitones, 2 descending, 1 ascending and 5 descending.

```
{
    "query": {
       "match_phrase": {"chromatic": "2+;2-;1+ 2-;1+;5-"}
      },
    "highlight": {"fields" : {"chromatic" : {}}}
}
```

ElasticSearch carries out the search operation, and at this point we benefit from all the capacities of a top-level indexing system: a set of all the matching documents is retrieved.

The non-standard part then occurs: we must rank this set according to the relevant distance (which is not the default one supplied by ElasticSearch for textual data). Elasticsearch is a tunable search engine that can be extended with a specific ranking algorithm. We implemented our own *SearchScript* (https://www.elastic.co/guide/en/elasticsearch/reference/current/modules-scripting.html), as a Java implementation of the proposed ranking procedures (Sections 5.3 and 5.4).

The following example shows how to use our custom *ScoreSim* ranking function, for a diatonic search and requiring a ranking on the rhythmic part. *SearchScript* requires to specify the custom plugin name (here "*lang:ScoreSim*"), and input parameters (here "*params*") that will be used in the procedure. Here two parameters are given, the first one gives the searched pattern "*query*" and the type of similarity that is applied "*similarity*".

```
{
    "query": {
        "function_score": {
            "functions":[
                {"script_score": {"script": {
                    "lang": "ScoreSim",
                    "params": {
                        "query": "(0|1/2)(1|1)(2|1)",
                        "similarity":"rhythmic"
                        }
        }}}
    ]}}
}
```

6.3. Distribution and Aggregation

A major feature of Elasticsearch is its ability to scale up by distributing indexes in a cluster. The fact that we split polyphonic music descriptors as individual monodic documents in the system allows to homogeneously distribute the computation of *ScoreSim* all over the repository. One obtains a matching score for each monodic descriptor in a distributed context.

However it requires to recompose the global score. This is done by applying an aggregate function (the "grouping" phase on Fig. 15). The following example applies three aggregate functions on grouped documents on "*doc_id*" and the final result is sorted according to the maximum score ("`max_scoresim`") over all descriptors.

```
{
    "query": {},
    "aggs":{"group_score":{
            "terms":{
                "field":"doc_id",
                "order" : { "max_scoresim" : "desc" }
            },
            "aggs":{
                "max_scoresim": {"max" : {"script":"_score"}},
                "min_scoresim": {"min" : {"script":"_score"}},
                "avg_scoresim": {"avg" : {"script":"_score"}}
    }}}
}
```

6.4. Highlighting

Alongside documents identifiers, Elasticsearch provides some information about the matching parts in the selected documents. The following example shows an ElasticSearch JSON result document, featuring the `highlight` field with two matching occurrences, enclosed in *windows* delimited by `<em>` tags.

```
{
    "_id" : "doc_id:descr_id",
    "_score" : 0.8301817,
    "highlight" : {
        "chromatic" : ["7+;3-;2+; <em>3-;2+;1+; 2+;1+;1-;</em> 1+;1-;5+;...",
        "2+;3-;2+; <em>3-;2+;1+; 2+;1+;1-;</em> 1+;1-;7+;"
        ]
      }
}
```

From each window, one obtains the n-grams positions and then uses the RAT table (see Section 5.5) to determine the position of each occurrence in the original document.

6.5. Interacting with the Server

We briefly illustrate how the search server can be integrated in an application managing large collections of scores with Neuma digital library. Neuma [76] maintains corpuses of music scores, encoded in MusicXML and MEI. It features a Graphical User Interface (GUI) to communicate with the search server. Patterns and search mode can be entered with an interactive virtual keyboard (Fig. 19). Search modes correspond to the feature types presented in Section 4.

6.6. Data and Performance Evaluation

In order to study the performances of our approach, we compare our architecture implemented with Elasticsearch to a traditional pattern search, based on regular expressions. We have imported in the Neuma platform [76] a corpus of 14,637 scores from various

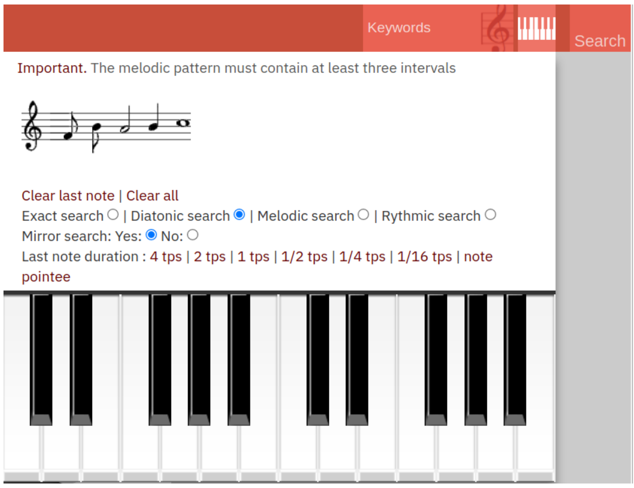
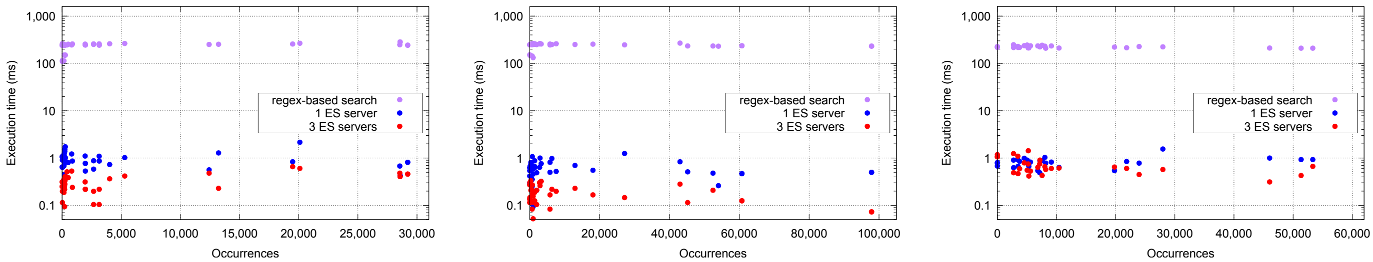


Figure 19. Interactive piano keyboard, for query inputs on the Neuma platform.



(a) Chromatic **(b)** Diatonic **(c)** Rhythmic

Figure 20. Execution time per query wrt. pattern occurrence for the chromatic, diatonic and rhythmic features. Regular-expressions searches are consistently orders of magnitude more costly than our ElasticSearch-based implementation.

sources, including Kern@HumDrum (https://kern.humdrum.org/). We then apply different queries on the whole corpora and report the computation times. We especially focus on applying various patterns, from infrequent to more frequent ones, based on the popularity of the indexed *n*-grams.

We sampled 40 patterns from each of the three chromatic, diatonic and rhythmic feature domains, with different distributions of pattern occurrences. The patterns exhibit different lengths, from *3*-grams to *11*-grams (which are rather infrequent). In the rhythmic domain, for instance, `(1)(1)(1)` appears 469,222 times in the whole corpus, while `(1/2)(2)(1/2)(1)(1)(2)(3/4)(1/3)(2)` is found only once. The goal of this test set is to evaluate the scalability of our system and its robustness to various pattern sizes and selectivity.

Figure 20 show the execution time in log-scale of each pattern query on the whole corpus. It gives the time spent on 1) a traditional pattern search with regular expressions [49–52] (purple dots), 2) our implementation in a single server of Elasticsearch (blue dots) and 3) on a cluster of 3 servers (red dots).

The evaluation of regular expressions (regex), without index, must process the whole corpus and scan each extracted feature to find matching occurrences. Consequently, the time is dependent on the size of the corpus. Thus, in all experiments (see Table 1) we obtain a mean time of 246 ms (with a standard deviation of 41 ms) for chromatic features, 250 ms (resp. 21 ms) for diatonic, and 219 ms (resp. 10 ms) for rhythmic.

As presented in Section 5.2, a scalable search engine relies on inverted lists. Execution times are much lower, since indexes help to find the proper *n*-gram and are less dependent on the sizes of the corpora. We can see for the three indexed features that the execution

Table 1. Global execution time

	Feature	Mean time	Standard deviation
Regex-based search	**Chromatic**	246 ms	41 ms
	Diatonic	250 ms	21 ms
	Rhythmic	219 ms	10 ms
1 server	**Chromatic**	0.869 ms	0.269 ms
	Diatonic	0.583 ms	0.194 ms
	Rhythmic	0.804 ms	0.136 ms
3 servers	**Chromatic**	0.312 ms	0.110 ms
	Diatonic	0.166 ms	0.064 ms
	Rhythmic	0.700 ms	0.216 ms

time is around 1 ms to process queries. This is more than 200 times faster compared to regular expressions.

On a single server, the execution time increases with the number of occurrences (see the rhythmic feature on Figure 20c for instance, at around 0.8 ms). This is explained by the fact that inverted lists are longer. This effect can be seen also in a cluster of servers with an execution time of 0.3 ms for chromatic and 0.7 ms for rhythmic features. The gain from a central to a distributed environment is dependent on the distribution of n-grams over the servers. The rhythmic domain has more highly frequent patterns, which explains why execution time does not vary much between 1 and 3 servers (Figure 20c). Conversely, the gain is higher when the patterns are more distributed, which is the case for chromatic and diatonic features. We obtain as much as a 3.5 times speed improvement (three servers *vs* one).

7. Conclusion and future work

We presented in this paper a practical approach to the problem of indexing a large library of music documents. Our solution fulfills three major requirements for an information retrieval system: i) it supports search with a significant part of flexibility, ii) it proposes a ranking method consistent with the matching definition, and iii) it brings scalability thanks to its compatibility with the features of state-of-the-art search engines. We believe that our design is complete, robust, and covers most of the functionalities expected from a scalable search system tailored to the specifics of music information.

We fully implemented our solution for the specific situation of XML-encoded music scores, and supply a packaged Docker image for any institution wishing to use a ready-to-use music-oriented search engine. Our solution is also available as a component of the Neuma platform [76], with a user-friendly interface (patterns are input with a piano keyboard) and a large collection of scores to illustrate the operation of the framework.

There exists many directions of research to extend the current work: integration to other music representations; extension of the features set and refinement of the core information retrieval modules (searching and ranking).

If we turn to alternative representation, the most important seem audio and digitized score sheets (massively found in patrimonial archives). In both cases, the focus is on the development of a specific extractor for the considered format, the rest of the framework being unchanged. For audio documents, extracting a music content descriptor is akin to Automatic Music Transcription (AMT) [81]. As detailed in Section 2, this is an active area of research. Satisfying result are obtained (in research labs) for monophonic music, whereas polyphonic music transcription is still a challenging problem. Regarding digitized score sheets, the tool of choice is Optical Music Recognition (OMR). OMR modules are proposed as part of commercial music notation editors. The result depends on the quality of the image supplied to the system. In general, it is still difficult to avoid a manual post-correction.

Our team is active in both directions, and in both cases we target a goal which is less ambitious and more specific than full fledged AMT or OMR. Indeed, both aim at producing a complete music score, featuring an adequate placement of graphical elements (notes, staves, clefs). In our approach, we would satisfy ourselves with the mere extraction of a core info-set sufficient to build our content descriptor, avoiding therefore the burden of dealing with complex graphic representation issues. This simplifies the target, but does not keep from addressing the other important issues regarding in particular the quality of the result.

The search engine could also be extended with a faceting capability, to enhance filtering the search result page and organize relevant documents. Another future direction is to add new features. Some could be extracted from symbolic music data, such as harmony and tonality. Some may require data in audio format, like timbre, since certain types of features are only available for extraction in audio. The major challenge of this task is to rank the search result of queries targeted on such features.

Finally, the ranking part of the search engine could be more versatile. In the future, ranking with geometric measures [82], transposition distance [83] or Dynamic Time Warping based approaches [84,85] could be integrated in the system.

Acknowledgments: This work is part of a project that has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101004746 (Polifonia: a digital harmoniser for musical heritage knowledge, H2020-SC6- TRANSFORMATIONS).

References

1. Manning, C.D.; Raghavan, P.; Schutze, H. *Introduction to Information Retrieval*; Cambridge University Press, 2008. Online version at *http://informationretrieval.org/*.
2. Samet, H. *Foundations of Multidimensional and Metric Data Structures*; Morgan Kaufmann, 2006.
3. Zhou, W.; Li, H.; Tian, Q. Recent advance in content-based image retrieval: A literature survey. *arXiv preprint arXiv:1706.06064* **2017**.
4. Chen, W.; Liu, Y.; Wang, W.; Bakker, E.; Georgiou, T.; Fieguth, P.; Liu, L.; Lew, M.S. Deep Image Retrieval: A Survey, 2021, [arXiv:cs.CV/2101.11282].
5. Rothstein, J. *MIDI: A Comprehensive Introduction*; Computer Music and Digital Audio Series, A-R Editions, 1995.
6. Apel, W. *The Notation of Polyphonic Music, 900-1600*; The medieval academy of America, 1961.
7. Gould, E. *Behind Bars*; Faber Music, 2011; p. 676.
8. Huron, D. The Humdrum Toolkit: Software for Music Research, 1994-.
9. Good, M., "MusicXML for Notation and Analysis". In *The Virtual Score: Representation, Retrieval, Restoration*; W. B. Hewlett and E. Selfridge-Field, MIT Press, 2001; pp. 113–124.
10. MNX 1.0 draft specification, 2021, [https://github.com/w3c/mnx]. Accessed 2021-11.
11. Rolland, P. The Music Encoding Initiative (MEI). Proceedings of the International Conference on Musical Applications Using XML, 2002, pp. 55–59.
12. Music Encoding Initiative, 2015, [http://www.music-encoding.org]. Accessed 2021-09.
13. Huron, D. Music Information Processing Using the Humdrum Toolkit: Concepts, Examples, and Lessons. *Computer Music Journal* **2002**, *26*, 11–26. doi:10.1162/014892602760137158.
14. Sapp, C.S. Online Database of Scores in the Humdrum File Format . Proceedings of the International Conference on Music Information Retrieval (ISMIR), 2005.
15. Rigaux, P.; Abrouk, L.; Audéon, H.; Cullot, N.; Davy-Rigaux, C.; Faget, Z.; Gavignet, E.; Gross-Amblard, D.; Tacaille, A.; Thion-Goasdoué, V. The design and implementation of Neuma, a collaborative Digital Scores Library - Requirements, architecture, and models. *Int. J. on Digital Libraries* **2012**, *12*, 73–88.
16. MuseScore, [https://musescore.org/]. Accessed: 2021-12.
17. Carterette, B.; Jones, R.; Jones, G.F.; Eskevich, M.; Reddy, S.; Clifton, A.; Yu, Y.; Karlgren, J.; Soboroff, I. Podcast Metadata and Content: Episode Relevance and Attractiveness in Ad Hoc Search. *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval* **2021**. doi:10.1145/3404835.3463101.
18. Discogs: Music Database and Marketplace, [https://www.discogs.com/]. Accessed: 29.09.2021.
19. AllMusic, [https://www.allmusic.com/]. Accessed: 03.12.2021.
20. Musixmatch, [https://www.musixmatch.com/]. Accessed: 03.12.2021.
21. Wang, A. The Shazam music recognition service. *Communications of the ACM* **2006**, *49*, 44–48.
22. SoundHound, [https://www.soundhound.com/]. Accessed: 29.09.2021.

23. Kotsifakos, A.; Papapetrou, P.; Hollmén, J.; Gunopulos, D.; Athitsos, V. A Survey of Query-by-Humming Similarity Methods. Proceedings of the 5th International Conference on Pervasive Technologies Related to Assistive Environments; Association for Computing Machinery: New York, NY, USA, 2012; PETRA '12. doi:10.1145/2413097.2413104.
24. Cuthbert, M.e.a. Music21, 2006-.
25. Foscarin, F.; Jacquemard, F.; Rigaux, P.; Sakai, M. A Parse-based Framework for Coupled Rhythm Quantization and Score Structuring. MCM 2019 - Mathematics and Computation in Music; Springer: Madrid, Spain, 2019; Vol. Lecture Notes in Computer Science, *Proceedings of the Seventh International Conference on Mathematics and Computation in Music (MCM 2019)*. doi:10.1007/978-3-030-21392-3_20.
26. Foscarin, F.; Fournier-S'niehotta, R.; Jacquemard, F. A diff procedure for music score files. Proceedings of the 6th International Conference on Digital Libraries for Musicology (DLfM); ACM: The Hague, Netherlands, 2019; pp. 7–11.
27. Antila, C.; Treviño, J.; Weaver, G. A hierarchic diff algorithm for collaborative music document editing. Third International Conference on Technologies for Music Notation and Representation (TENOR), 2017.
28. Simonetta, F.; Carnovalini, F.; Orio, N.; Rodà, A. Symbolic music similarity through a graph-based representation. In *Proceedings of the Audio Mostly 2018 on Sound in Immersion and Emotion*; Association for Computing Machinery: New York, NY, USA, 2018. doi:10.1145/3243274.3243301.
29. Salamon, J.; Gomez, E. Melody Extraction From Polyphonic Music Signals Using Pitch Contour Characteristics. *IEEE Transactions on Audio, Speech, and Language Processing* **2012**, *20*, 1759–1770. doi:10.1109/TASL.2012.2188515.
30. Kim, J.W.; Salamon, J.; Li, P.; Bello, J.P. Crepe: A Convolutional Representation for Pitch Estimation. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2018, pp. 161–165. doi:10.1109/ICASSP.2018.8461329.
31. Bainbridge, D.; Bell, T. The Challenge of Optical Music Recognition. *Computers and the Humanities* **2001**, *35*, 95 – 121. doi:10.1023/A:1002485918032.
32. Rebelo, A.; Fujinaga, I.; Paszkiewicz, F.; Marçal, A.R.S.; Guedes, C.; Cardoso, J.S. Optical music recognition: state-of-the-art and open issues. *International Journal of Multimedia Information Retrieval* **2012**, *1*, 173–190.
33. Choi, K.Y.; Coüasnon, B.; Ricquebourg, Y.; Zanibbi, R. Music Symbol Detection with Faster R-CNN Using Synthetic Annotations. First International Workshop on Reading Music Systems, 2018.
34. Ríos Vila, A.; Rizo, D.; Calvo-Zaragoza, J. Complete Optical Music Recognition via Agnostic Transcription and Machine Translation. Intl. Conf. on Document Analysis and Recognition (ICDAR 2021), 2021, pp. 661–675. doi:10.1007/978-3-030-86334-0_43.
35. Knopke, I. The Perlhumdrum and Perllilypond Toolkits for Symbolic Music Information Retrieval. Proceedings of the International Conference on Music Information Retrieval (ISMIR), 2008, pp. 147–152.
36. Kornstaedt, A. Themefinder: A Web-based Melodic Search Tool. *Computing in Musicology* **1998**, *11*, 231–236.
37. Stuart, C.; Liu, Y.W.; Selfridge-Field, E. Search-effectiveness measures for symbolic music queries in very large databases. Proceedings of the International Conference on Music Information Retrieval (ISMIR), 2004.
38. Prechelt, L.; Typke, R. An Interface for Melody Input. *ACM Transactions on Computer-Human Interaction (TOCHI)* **2001**, *8*, 133–149.
39. Fournier-S'niehotta, R.; Rigaux, P.; Travers, N. Is There a Data Model in Music Notation? Intl. Conf. on Technologies for Music Notation and Representation (TENOR'16); Hoadley, R.; Nash, C.; Fober, D., Eds. Anglia Ruskin University, 2016, pp. 85–91.
40. Fournier-S'niehotta, R.; Rigaux, P.; Travers, N. Querying XML Score Databases: XQuery is not Enough! Proceedings of the International Conference on Music Information Retrieval (ISMIR), 2016.
41. Fournier-S'niehotta, R.; Rigaux, P.; Travers, N. Modeling Music as Synchronized Time Series: Application to Music Score Collections. *Information Systems* **2018**, pp. 35–49.
42. Casey, M.; Veltkamp, R.; Goto, M.; Leman, M.; Rhodes, C.; Slaney, M. Content-based music information retrieval: current directions and future challenges. *Proceedings of the IEEE* **2008**, *96*, 668–696.

43. Jones, M.C.; Downie, J.S.; Ehmann, A.F. Human Similarity Judgments: Implications for the Design of Formal Evaluations. Proceedings of the International Conference on Music Information Retrieval (ISMIR), 2007, pp. 539–542.
44. Ens, J.; Riecke, B.E.; Pasquier, P. The Significance of the Low Complexity Dimension in Music Similarity Judgements. Proceedings of the International Conference on Music Information Retrieval (ISMIR), 2017, pp. 31–38.
45. Velardo, V.; Vallati, M.; Jan, S. Symbolic Melodic Similarity: State of the Art and Future Challenges. *Computer Music Journal* **2016**, *40*, 70–83.
46. Typke, R.; Wiering, F.; Veltkamp, R.C. A survey of music information retrieval systems. Proceedings of the International Conference on Music Information Retrieval (ISMIR), 2005, pp. 153–160.
47. Nanopoulos, A.; Rafailidis, D.; Ruxanda, M.M.; Manolopoulos, Y. Music search engines: Specifications and challenges. *Information Processing & Management* **2009**, *45*, 392–396. doi:10.1016/j.ipm.2009.02.002.
48. Viro, V. Peachnote: Music Score Search and Analysis Platform. Proceedings of the International Conference on Music Information Retrieval (ISMIR), 2011, pp. 359–362.
49. Downie, J.S. Evaluating a simple approach to music information retrieval: Conceiving melodic n-grams as text. PhD thesis, Univ. Western Ontario, 1999.
50. Neve, G.; Orio, N. Indexing and Retrieval of Music Documents through Pattern Analysis and Data Fusion Techniques. Proceedings of the International Conference on Music Information Retrieval (ISMIR), 2004, pp. 216–223.
51. Doraisamy, S.; Rüger, S.M. A Polyphonic Music Retrieval System Using N-Grams. Proceedings of the International Conference on Music Information Retrieval (ISMIR), 2004, pp. 204–209.
52. Chang, C.W.; Jiau, H.C. An Efficient Numeric Indexing Technique for Music Retrieval System. Proceedings of the IEEE International Conference on Multimedia and Expo (ICME), 2006, pp. 1981–1984.
53. Sanyal, A.; others. Modulo7: A Full Stack Music Information Retrieval and Structured Querying Engine. PhD thesis, Johns Hopkins University, 2016.
54. Constantin, C.; du Mouza, C.; Faget, Z.; Rigaux, P. The Melodic Signature Index for Fast Content-based Retrieval of Symbolic Scores Camelia Constantin. Proceedings of the 12th International Society for Music Information Retrieval Conference, ISMIR 2011, Miami, Florida, USA, October 24-28, 2011, 2011, pp. 363–368.
55. Rigaux, P.; Travers, N. Scalable Searching and Ranking for Melodic Pattern Queries. Intl. Conf. of the International Society for Music Information Retrieval (ISMIR); , 2019.
56. Haydn, J. Das Lied der Deutschen, 1797. Lyrics by August Heinrich Hoffmann von Fallersleben.
57. Taylor, C.; Campbell, M. Sound. Grove Music Online, 2001. doi:10.1093/gmo/9781561592630.article.26289
58. Haynes, B.; Cooke, P. Pitch. Grove Music Online, 2001. doi:10.1093/gmo/9781561592630.article.40883.
59. Roeder, J. Pitch class. Grove Music Online, 2001. doi:10.1093/gmo/9781561592630.article.21855.
60. Lindley, M. Interval. Grove Music Online, 2001. (revised by Murray Campbell and Clive Greated), doi:10.1093/gmo/9781561592630.article.13865.
61. Drabkin, W. Diatonic. Grove Music Online, 2001. doi:10.1093/gmo/9781561592630.article.07727.
62. Mullally, R. Measure. Grove Music Online, 2001. doi:10.1093/gmo/9781561592630.article.18226.
63. Greated, C. Beats. Grove Music Online, 2001. doi:10.1093/gmo/9781561592630.article.02424.
64. Campbell, M. Timbre. Grove Music Online, 2001. doi:10.1093/gmo/9781561592630.article.27973.
65. Texture. Grove Music Online, 2001. doi:10.1093/gmo/9781561592630.article.27758.
66. Anka, P. My Way, 1969. Interpret: Frank Sinatra.
67. Revaux, J.; François, C. Comme d'habitude, 1967.
68. Orpen, K.; Huron., D. Measurement of Similarity in Music: A Quantitative Approach for Non-Parametric Representations. *Computers in Music Research* **1992**, *4*, 1–44.
69. Toussaint, G. A comparison of rhythmic similarity measures. Proceedings of the International Conference on Music Information Retrieval (ISMIR), 2004, ISMIR'04, pp. 242–245.
70. Beltran, J.; Liu, X.; Mohanchandra, N.; Toussaint, G. Measuring Musical Rhythm Similarity: Statistical Features versus Transformation Methods. *Journal of Pattern Recognition and Artificial Intelligence* **2015**, *29*, 1–23.
71. Post, O.; Toussaint, G. The Edit Distance as a Measure of Perceived Rhythmic Similarity. *Empirical Musicology Review* **2011**, *6*, 164–179.
72. Demaine, E.D.; Mozes, S.; Rossman, B.; Weimann, O. An optimal decomposition algorithm for tree edit distance. *ACM Trans. Algorithms* **2009**, *6*, 2:1–2:19.

73. Zhang, K.; Shasha, D. Simple Fast Algorithms for the Editing Distance between Trees and Related Problems. *J. Comput.* **1989**, *18*, 1245–1262. doi:10.1137/0218082.
74. Ristad, E.; Yianilos, P. Learning string-edit distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **1998**, *20*, 522–532. doi:10.1109/34.682181.
75. Habrard, A.; Iñesta, J.M.; Rizo, D.; Sebban, M. Melody Recognition with Learned Edit Distances. Structural, Syntactic, and Statistical Pattern Recognition; da Vitoria Lobo, N.; Kasparis, T.; Roli, F.; Kwok, J.T.; Georgiopoulos, M.; Anagnostopoulos, G.C.; Loog, M., Eds.; Springer Berlin Heidelberg: Berlin, Heidelberg, 2008; pp. 86–96.
76. Cnam. Neuma on-line digital library, 2009-, [http://neuma.huma-num.fr/].
77. ElasticSearch. The Elastic search engine, [https://www.elastic.co/]. Accessed: 2021-12.
78. Solr. The Apache Solr search engine, [https://solr.apache.org/]. Accessed: 2021-12.
79. Sphinx. The Sphinx search engine, [https://sphinxsearch.com/]. Accessed: 2021-12.
80. Cuthbert, M.S.; Ariza, C. Music21: A Toolkit for Computer-Aided Musicology and Symbolic Music Data. Proceedings of the International Conference on Music Information Retrieval (ISMIR), 2010, pp. 637–642.
81. Benetos, E.; Dixon, S.; Giannoulis, D.; Kirchhoff, H.; Klapuri, A. Automatic music transcription: challenges and future directions. *Journal of Intelligent Information Systems* **2013**, *41*, 407–434.
82. Aloupis, G.; Fevens, T.; Langerman, S.; Matsui, T.; Mesa, A.; Rodríguez, Y.; Rappaport, D.; Toussaint, G. Algorithms for Computing Geometric Measures of Melodic Similarity. *Computer Music Journal* **2006**, *30*, 67–76. doi:10.1162/comj.2006.30.3.67.
83. Veltkamp, R.; Typke, R.; Giannopoulos, P.; Wiering, F.; Oostrum, R. Using Transportation Distances for Measuring Melodic Similarity. Proceedings of the International Conference on Music Information Retrieval (ISMIR), 2003.
84. Keogh, E.J.; Ratanamahatana, C.A. Exact Indexing of Dynamic Time Warping. *Knowl. Inf. Syst.* **2005**, *7*, 358–386.
85. Frieler, K.; Höger, F.; Pfleiderer, M.; Dixon, S. Two web applications for exploring melodic patterns in jazz solos. Proceedings of the International Conference on Music Information Retrieval (ISMIR), 2018, pp. 777–783.