CONTENT-BASED INFORMATION RETRIEVAL BY GROUP THEORETICAL METHODS

Michael Clausen
Department of Computer Science III
*University of Bonn, Germany**
clausen@cs.uni-bonn.de

Frank Kurth
Department of Computer Science III
University of Bonn, Germany
frank@cs.uni-bonn.de

Abstract This paper presents a general framework for efficiently indexing and searching large collections of multimedia documents by content. Among the multimedia information retrieval scenarios that fit into this framework are music, audio, image and 3D object retrieval. Combining the technique of inverted files with methods from group theory we obtain space efficient indexing structures as well as time efficient search procedures for content-based and fault-tolerant search in multimedia data. Several prototypic applications are discussed demonstrating the capabilities of our new technique.

Keywords: multimedia retrieval, content-based retrieval, music and audio retrieval.

1. Introduction

The last few years have seen an increasing importance of multimedia databases for a wide range of applications. As one major reason, the availability of affordable high-performance hardware now allows for efficient processing and storage of the huge amounts of data which arise, e.g., in video, image, or audio applications. A key philosophy to accessing data in multimedia databases is *content-based retrieval* ([Yoshitaka

*This work was supported in part by Deutsche Forschungsgemeinschaft under grant CL 64/3

and Ichikawa, 1999]), where the content of the multimedia documents is processed rather than just some textual annotation describing the documents. Hence, a content-based query to an image database asking for all images showing a certain person would basically rely on a suitable feature extraction mechanism to scan images for occurrences of that person. On the other hand, a classical query based on additional textual information would rely on the existence of a suitable textual annotation of the contents of all images. Unfortunately, in most cases such an annotation is neither available nor may it be easily extracted automatically, emphasizing the demand for feasible content-based retrieval methods.

It turns out that many content-based retrieval problems share essential structural properties. We briefly sketch two of those problems.

Let us first consider a problem from music information retrieval. Assume that a music database consists of a collection of scores. That is, each database document is a score representation of a piece of music containing the notes of that piece as well as additional information such as meter or tempo. Now we consider the following database search problem: Given a melody or, more generally, an arbitrary excerpt of a piece of music, we are looking for all occurrences of that *query* or slight variations thereof in the database documents. The result of such a query could for example help music professionals to discover plagiarism. A simpler, yet very active application area is to name a tune that is whistled or hummed into a microphone. As a query result, a user could expect information on title, composer, and consumer information on an audio CD containing the corresponding piece of music.

The second problem is concerned with content-based image retrieval. Consider a database consisting of digital copyrighted images. Assume that we are interested, e.g., for some legal reasons, in finding all web pages on the Internet containing at least one of the copyrighted images or fragments thereof. This problem may be again considered as a database search problem: Given a (query) image taken from some web page, we are looking for an occurrence of that image as a subimage of one of the database images, including the case that the query matches one of those images as a whole. Extensions to this problem include that we are also interested in finding rotated, resized, or lower-quality versions of the original images.

Both of the above problems may be viewed in the following general setting: A query to a database consisting of a collection of multimedia documents has to be answered in the sense that certain transformations (or generalized shift operations) have to be found which transport the query to its location within the database document. In this tutorial paper, we systematically exploit this principle for the case that admissible

transformations are taken from a *group* acting on a *set* which in turn constitute the database documents. It turns out that this approach leads to very efficient search algorithms for a large class of content-based search problems, which are in particular applicable to spatial-, temporal-, or spatio-temporal retrieval settings ([Yoshitaka and Ichikawa, 1999]).

We briefly summarize the main contributions of our approach:

- We develop a general framework for retrieval of multimedia documents by example. Our technique's flexibility has been demonstrated by prototypes in various fields (e.g., music, audio, image, and (relational) object retrieval).
- We propose generic algorithms for query evaluation together with efficient algorithms for fault-tolerant retrieval which consequently exploit the structure inherent in the retrieval problems.
- In contrast to previously reported approaches, query evaluation becomes *more efficient* when the complexity of a query increases.
- The concept of partial matches (i.e., a query is only matched to a part of a document) is an integral part of our technique and requires no additional storage.

The proposed technique has been successfully tested on a variety of content-based retrieval problems. We summarize some figures on those prototypes:

- Our PROMS system, for the first time, allowed for efficient polyphonic search in polyphonic scores ([Clausen et al., 2000]). E.g., queries to a database of 12,000 pieces of music containing 33 million notes can be answered in about 50 milliseconds.
- Our system for searching large databases of audio signals allows for both identifying *and* precisely locating short fragments of audio signals w.r.t. the database. The sizes of our search indexes are very small, e.g., only 1:1,000–1:15,000 the size of the original audio data depending on the required retrieval granularity. As an example, a database of 180 GB of audio material can be indexed using about 50 MB only, while still allowing audio signals of only several seconds of length to be located within fractions of seconds ([Ribbrock and Kurth, 2002]).
- Index construction may be performed very efficiently. For the PROMS system, index construction takes only a few minutes, hence allowing for indexing on the fly. Indexing PCM audio data

may be performed several times faster than real-time on standard PC hardware.

- In our prototypic image retrieval system containing 3,300 images, exact sub-image queries require about 50ms response time ([Röder, 2002]). The search index is compressed to 1:6 compared to the original (JPEG) data.

The paper is organized as follows. In the next section, we discuss several motivating examples and give an informal overview on the concepts of our approach to content-based multimedia retrieval. Those concepts are introduced formally in Sections 3 and 4. Section 3 deals with a formal specification of documents, queries, the notion of matches, and the derivation of fast retrieval algorithms. Section 4 introduces two general mechanisms to incorporate fault tolerance: mismatches and fuzzy queries. In Section 5 we present prototypic applications from the fields of music-, audio-, image-, and object retrieval. Finally, Section 6 gives a brief overview on related work and suggest some future research directions.

2. Motivating Examples

The goal of this section is to present three motivating examples that illustrate the desirability of a unified approach to content-based retrieval. The first two are concerned with text retrieval tasks, whereas the third discusses content-based retrieval in score-based music data.

As a first example, suppose we have a collection $\mathcal{T} = (T_1, \ldots, T_N)$ of text documents each consisting of a finite sequence of *words* where each word is contained in a set (i.e., a dictionary) W of all admissible words. Taking a coarse level of granularity, each text document is preprocessed in order to extract a set of *terms* occurring in that document. If T denotes the set of all conceivable terms, then this preprocessing step produces a sequence $\mathcal{D} = (D_1, \ldots, D_N)$ of finite subsets of T, where D_i is the set of terms extracted from the ith document. We may think of the term set T as a reduced version of W, where unimportant words have been left out and verbs, nouns, etc. in W have been replaced by their principal forms, e.g., by means of a stemming algorithm (e.g., *sitting* $\mapsto$ *sit*).

Next, let us think about content-based queries. A usual way of formulating a query is to input a finite set Q of terms to the retrieval system. Then one possible task is to compute the set $H_{\mathcal{D}}(Q)$ of all *exact partial matches*, which is the set of all documents containing all of the terms

specified by Q. More formally, with $[1:N] := \{1, \ldots, N\}$,

$$H_{\mathcal{D}}(Q) := \{i \in [1:N] \mid Q \subseteq D_i\}. \tag{1}$$

Note that this is a rather strict decision about the Q-relevance of a document: if just one $q \in Q$ does not occur in D_i, then D_i is already considered irrelevant. A more realistic way to estimate relevance would be a ranking of the documents along the quotients $|Q \cap D_i|/|Q| \in [0,1]$. This quotient equals 1 if and only if Q is a subset of D_i.

From the viewpoint of logic, presenting a set $Q = \{q_1, \ldots, q_n\}$ of terms to the retrieval system is equivalent to the boolean query $q_1 \wedge \ldots \wedge q_n$ which asks for all documents that contain all these terms. More generally, we can form boolean expressions like $(t_1 \wedge t_2) \vee (t_3 \wedge \neg t_4)$, the latter asking for all documents containing both the terms t_1 and t_2 or the term t_3 but not the term t_4.

After this brief introduction to content-based queries and different text retrieval tasks, we are now going to discuss efficient algorithms for text retrieval. For each term $t \in \cup_{i=1}^{N} D_i$ one establishes a so-called inverted file, which is the (linearly ordered) set $H_{\mathcal{D}}(t) := \{i \in [1:N] \mid t \in D_i\}$. Then, for a query $Q \subseteq T$, the set of all exact partial matches is obtained by intersecting the inverted files of all elements in Q:

$$H_{\mathcal{D}}(Q) = \bigcap_{t \in Q} H_{\mathcal{D}}(t).$$

This generalizes to boolean queries. For example, $H_{\mathcal{D}}((t_1 \wedge t_2) \vee (t_3 \wedge \neg t_4)) = (H_{\mathcal{D}}(t_1) \cap H_{\mathcal{D}}(t_2)) \cup (H_{\mathcal{D}}(t_3) \setminus H_{\mathcal{D}}(t_4))$.

The above discussion immediately leads to the following computational problems: given (linearly ordered) finite sets A and B of integers, compute their intersection, union and difference, again linearly ordered. If a and b denote the cardinality of A and B, respectively, then $A \cap B$, $A \cup B$ and $A \setminus B$ may be computed with at most $a+b$ comparisons using a merge technique. If $a \ll b$ then comparing the elements of A one after another by means of the binary search method (see, e.g., [Cormen et al., 1990]) with the elements of B causes at most $a \log b$ comparisons to solve each of these problems.

So far we have discussed a rather coarse notion of a match. If for example the document ID i is an exact partial match w.r.t. the query Q, then we only know that all terms in Q do also occur in the set of terms extracted from the ith document. However, we do not know *where* these terms occur in the original text document T_i. This in turn is the main goal of full-text retrieval, where questions are allowed that ask, e.g., for a document containing the text passage "to be or not to be." We discuss this kind of retrieval next.

In our second example we consider full-text retrieval where we use a finer level of granularity. For this, we consider the word sequences constituting a text document. If W denotes the set of all words then the ith document is viewed as a sequence $D_i = (w_{i0}, \ldots, w_{in_i})$ over W. To make this example more comparable to the previous one, we will identify this sequence with the set $\{[j, w_{ij}] \mid j \in [0 : n_i]\}$. (Note that replacing sequences by their corresponding sets means a switch from an implicit to an explicit specification of the words' places.) Similarly, a query like "to be or not to be" would then be identified with the set $\{[0, \text{to}], [1, \text{be}], [2, \text{or}], [3, \text{not}], [4, \text{to}], [5, \text{be}]\}$.

In general, for a query $Q = (q_0, \ldots, q_n) \equiv \{[j, q_j] \mid j \in [0 : n]\}$, we are looking for the set $H_{\mathcal{D}}(Q)$ of all pairs (t, i) such that $t+Q := \{[t+j, q_j] \mid j \in [0 : n]\}$ is a subset of D_i. To obtain all those pairs efficiently we use, in analogy to the first example, an inverted file $H_{\mathcal{D}}(w) := \{(j, i) \mid w = w_{ij}\}$, for each word w that occurs in any of the documents. We claim that

$$H_{\mathcal{D}}(Q) = \cap_{k=0}^{n}(H_{\mathcal{D}}(q_k) - k), \tag{2}$$

where $H_{\mathcal{D}}(q_k) - k := \{(j-k, i) \mid (j, i) \in H_{\mathcal{D}}(q_k)\}$. In fact, $(j, i) \in H_{\mathcal{D}}(Q)$ iff $(q_0, \ldots, q_n) = (w_{ij}, \ldots, w_{i,j+n})$. This is equivalent to $(j + k, i) \in H_{\mathcal{D}}(q_k)$, for all $k \in [0 : n]$, i.e., $(j, i) \in H_{\mathcal{D}}(q_k) - k$, for all k, which proves our claim. Note that we use square brackets (such as in $[1, \text{be}]$) to denote the elementary objects constituting our documents in order to avoid confusion with elements of the inverted files (such as (j, i) above) which are denoted by round brackets.

According to this formula, the list of all solutions is again the intersection of all relevant and properly adjusted inverted files. However, for stop words like *the, a, is, to, be*, the lists are rather long. To avoid long lists and to improve the query response time substantially, we consider instead of W the set of all pairs $W2$ as our new universe of elementary data objects. For each pair $[v, w]$ of words we generate the inverted file $H_{\mathcal{D}}(v, w) := \{(j, i) \mid [v, w] = [w_{ij}, w_{i,j+1}]\}$. On the one hand, this simple trick drastically increases the number of inverted files, on the other hand, the new lists are typically much smaller than the original inverted files. The query "to be or not to be" can now be processed by the following intersection (check this!):

$$H_{\mathcal{D}}(to, be) \cap (H_{\mathcal{D}}(or, not) - 2) \cap (H_{\mathcal{D}}(to, be) - 4).$$

Thus instead of intersecting six long lists when working with W, we now have to intersect only three small lists. Obviously, this generalizes to k-tuples of words, for any k.

After these rather classical examples let us now turn to our third example which deals with content-based music information retrieval. The reader should notice the analogy to full-text retrieval.

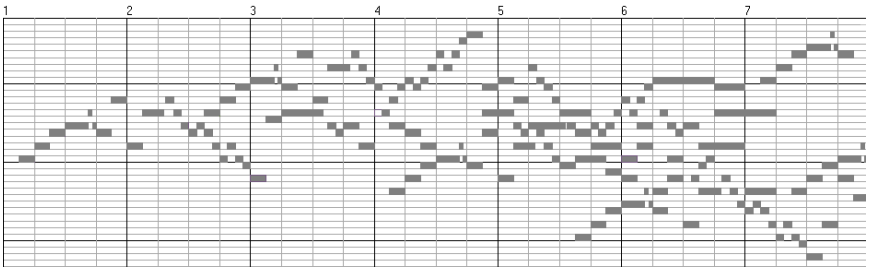

Figure 1. A part of J.S. Bach's Fugue in C major, BWV 846, in the piano roll notation.

Suppose we have a collection $\mathcal{T} = (T_1, \ldots, T_N)$ of scores. That is, each T_i is a score representation of a piece of music containing the notes of that piece as well as additional information such as meter, tempo or specifications of dynamics. Musical scores in the conventional staff notation may be visualized using the so-called *piano roll* representation. Fig. 1 shows the piano roll representation of the beginning of Johann Sebastian Bach's Fugue in C major, BWV 846. In this figure, the horizontal axis represents time whereas the vertical axis describes pitches. Each rectangle of width d located (w.r.t. its left lower corner) at coordinates (t, p) represents a note of pitch p, onset time t, and duration d. Such a note will be denoted by the triple $[p, t, d]$. (After a suitable quantization, we can assume w.l.o.g. that p, t and d are integers.)

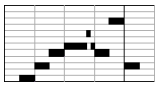

Figure 2. A query to the database in the piano roll notation.

Now we consider the following music information retrieval task which has already been sketched in the introduction: given a fragment of a melody or, more generally, an arbitrary excerpt of a piece of music, we are looking for all occurrences of that content-based query in the collection. As an example consider the query depicted in Fig. 2. In fact, this query is a part of the fugue's theme. Assume that we are looking for all positions where this theme or a pitch-transposed version thereof occurs. Then, Fig. 3 shows all occurrences of the query within the excerpt of the fugue given in Fig. 1.

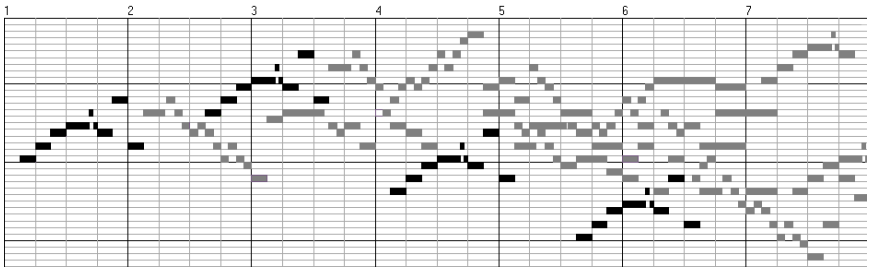

Figure 3. Bach fugue of Fig. 1. All occurrences of the query in Fig. 2 are highlighted.

To solve such matching problems, we first have to decide which parts of the complete score information are actually needed. By our experience, the most important parameters for recognizing a piece of music are pitch and onset time. In other words, a person will recognize a piece of music even if it is played in staccato throughout, i.e., disregarding the notes' durations. According to this experimental result, the first step is to extract from each score the pitches and onset times. This results in a new collection $\mathcal{D} = (D_1, \ldots, D_N)$ of finite subsets D_i of $\mathbb{Z}^2$. A content-based query is—after a possible preprocessing step—again a finite subset Q of $\mathbb{Z}^2$. The set $H_{\mathcal{D}}(Q)$ of all exact partial matches—when time- and pitch-shifts are allowed—is thus given by

$$H_{\mathcal{D}}(Q) := \{(\pi, \tau, i) \in \mathbb{Z}^2 \times [1:N] \mid (\pi, \tau) + Q \subseteq D_i\},$$

where $(\pi, \tau) + Q := \{[\pi + p, \tau + t] \mid [p, t] \in Q\}$. If we construct for every pair $[p, t] \in \mathbb{Z}^2$ an inverted file $H_{\mathcal{D}}([p, t]) := \{(\pi, \tau, i) \in \mathbb{Z}^2 \times [1:N] \mid [\pi + p, \tau + t] \in D_i\}$, then

$$H_{\mathcal{D}}(Q) = \bigcap_{[p,t] \in Q} H_{\mathcal{D}}([p, t]).$$

As in the case of text retrieval, the set of all exact partial matches is the intersection of all inverted files corresponding to the elements of Q. Obviously, there are infinitely many inverted files. Fortunately, from just *one* list one can recover the other inverted files according to the formula

$$H_{\mathcal{D}}([p, t]) = H_{\mathcal{D}}([0, 0]) - (p, t), \tag{3}$$

where $H_{\mathcal{D}}([0, 0]) - (p, t) := \{(\pi - p, \tau - t, i) \mid (\pi, \tau, i) \in H_{\mathcal{D}}([0, 0])\}$. In fact, $(\pi, \tau, i) \in H_{\mathcal{D}}([p, t])$ iff $[\pi + p, \tau + t] \in D_i$ iff $(\pi + p, \tau + t, i) \in H_{\mathcal{D}}([0, 0])$ iff $(\pi, \tau, i) \in H_{\mathcal{D}}([0, 0]) - (p, t)$, which proves our claim. Due to this formula, we only need to store the inverted file $H_{\mathcal{D}}([0, 0])$. The other lists can easily be computed from it if required. Remarkably, if $H_{\mathcal{D}}([0, 0])$ is linearly ordered, then so is $H_{\mathcal{D}}([0, 0]) - (p, t)$.

Let us analyze the computational cost of computing all exact partial matches w.r.t. the query Q using the formula

$$H_{\mathcal{D}}(Q) = \bigcap_{[p,t]\in Q} (H_{\mathcal{D}}([0,0]) - (p,t)). \tag{4}$$

As the list $H_{\mathcal{D}}([0,0])$ contains information about *all* notes from *all* scores in the collection, its size is proportional to the total number of notes in the collection. At least for large corpora this leads to unacceptable query response times. So we have to look for alternatives to get shorter lists.

Although Formula (4) looks very similar to Formula (2), there exists a significant difference concerning the number of independent inverted files: in our music retrieval scenario this number is one whereas in the full-text retrieval scenario this number equals the number of different terms in all documents. Thus text retrieval has the advantage of many independent inverted files that are typically small as compared to the storage requirements of the whole collection. Hence it is advantageous to make sure that many independent inverted files exist also in the music retrieval scenario.

There are different ways to achieve this goal. The first way is to reconsider the durations of the notes, resulting in documents D_i and Q over $\mathbb{Z}^3$ where the additional component represents the note duration. Using shifts $(\pi,\tau) + Q := \{[\pi + p, \tau + t, d] \mid [p,t,d] \in Q\}$ and inverted files $H_{\mathcal{D}}([p,t,d]) := \{(\pi,\tau,i) \in \mathbb{Z}^2 \times [1:N] \mid [\pi + p, \tau + t, d] \in D_i\}$, we note that now

$$H_{\mathcal{D}}([p,t,d]) = H_{\mathcal{D}}([0,0,d]) - (p,t). \tag{5}$$

Hence, the number of independent inverted lists equals the number of different durations occuring in the score collection. Besides fundamental difficulties of quantifying the duration, e.g., for staccato or grace notes, there will be extremely long lists for whole, half, quarter and eighth notes and these lists will typically be needed in most queries. So this alternative will generally not be suitable.

A second way of obtaining shorter independent inverted files is to exploit a user's prior knowledge. To avoid technicalities, assume that all pieces of the score collection are in 4/4 meter. Assume furthermore that each meter is subdivided into 16 metrical positions. When a user posing a query knows about the metrical position of the query, e.g., that the query starts at an offbeat, this may be incorporated as follows. We work with modified inverted files of the form

$$H'_{\mathcal{D}}([\pi,\tau]) := \{(p,k,i) \mid [p + \pi, 16k + \tau] \in D_i\}$$

and note that $H'_{\mathcal{D}}([\pi,\tau]) = H'_{\mathcal{D}}([0,\tau+16\ell]) - (\pi,-\ell,0)$ (Check this!). Thus, instead of one independent inverted file we now have 16 independent files

$$H'_{\mathcal{D}}([0,t]) := \{(\pi,k,i) \mid [\pi,16k+t] \in D_i\}$$

for $t \in [0:15]$. For a query Q containing correct indications of the metrical positions, our task is to compute the set $H'_{\mathcal{D}}(Q) := \{(p,k,i) \mid (p,16k)+Q \subseteq D_i\}$ which is equal to the intersection of all $H'_{\mathcal{D}}(q)$, q ranging over all elements of Q.

However, this only works if the user provides the retrieval system with accurate metrical information. If, in addition, the user knows about the exact pitches, then assuming 128 different pitches (like in the MIDI format), we now have 2048 independent inverted files

$$H''_{\mathcal{D}}([p,t]) := \{(p,k,i) \mid [p,16k+t] \in D_i\}$$

for $[p,t] \in [0:127] \times [0:15]$. In this case, a user has to know both the exact pitches and the metrical positions. So this second alternative requires a high degree of user knowledge. One could decrease this requirement by incorporating fault tolerance. We briefly discuss two mechanisms for achieving this: mismatches and fuzzy queries.

The first mechanism considers the case that a query is not completely contained in a document D_i. The elements of $Q \setminus D_i$ are called mismatches. Using a variant of an algorithm (see Section 4) for computing the ratios $|Q \cap D_i|/|Q|$, one may efficiently determine all documents with at most k mismatches.

Another possibility is to allow a user to pose fuzzy queries. In our music scenario, it could be the case that one is unsure about a certain pitch interval or the exact rhythm. Here, a user can specify alternatives for one note. Then a fuzzy query is a sequence $\mathbf{F} = (F_1,\ldots,F_n)$ of n finite sets F_i of alternatives. Such an $\mathbf{F}$ is the shorthand for a family of ordinary queries $Q = \{q_1,\ldots,q_n\}$ where for each i, q_i is allowed to take arbitrary values of F_i. A document D_i is an exact partial match for a fuzzy query $\mathbf{F}$ if $Q \subseteq D_i$, for some Q in this bunch of ordinary queries corresponding to $\mathbf{F}$. If the F_i are pairwise disjoint, then $\mathbf{F}$ consists of $\prod_{i=1}^{n} |F_i|$ ordinary queries. Although this number might grow rapidly, there are efficient algorithms to compute all matches, see Section 4.

Finally, let us discuss a third alternative. Instead of notes $[p,t]$ we now consider *pairs* of notes $([p,t],[p',t'])$ as our basic objects. As a query Q in our score scenario typically refers to a contiguous part of the score, we do not need to store all pairs of $D_i \times D_i$ but only those which are close to another. Noticing that most rows of a score correspond to one particular voice each, the storage complexity is still linear in the total length of the collection.

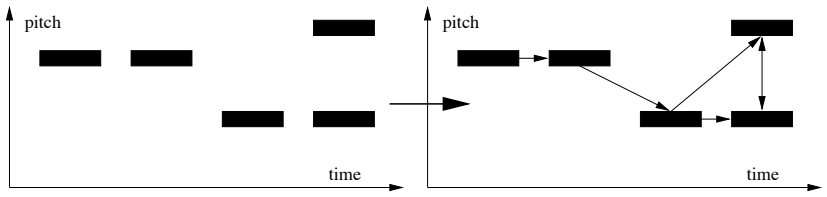

Figure 4. Replacing a document D (left) with a subset of $D2$ (right) consisting of neighboring objects of D. Elements of $D2$ are specified by pairs of notes joined by an arrow.

So documents D_i and queries Q are replaced by suitable subsets of D_i2 and $Q2$, respectively. Figure 4 illustrates this procedure. Considering pairs of notes is advantageous: we now have many independent inverted files: these are indexed by the pairs $([0,0],[p,t])$ with $p \in [-127:127]$ and $t \in \mathbb{Z}$. For details see the next section.

Prepared by these motivating examples, we are going to describe a general concept for multimedia information retrieval. This concept, worked out in detail in the next section, combines classical full-text retrieval techniques with methods from group theory.

3. General Concept

Let M be a set whose elements model the elementary data objects. A *document* over M is just a finite subset D of M. A *collection* (or *data base*) over M is a finite sequence $\mathcal{D} = (D_1, \ldots, D_N)$ of documents.

In the classical text retrieval example, $M = T$ is the set of all terms, whereas D_i equals the set of terms extracted from the ith text document. In the full-text retrieval scenario, $M = \mathbb{Z} \times W$ is the set of all words together with positional information. The ith text document T_i is viewed as a sequence of words. This in turn may be described by the subset $D_i = \{[j, w_{ij}] \mid j \in [0 : n_i]\}$ of M. Finally, in our score retrieval example, $M = [0:127] \times \mathbb{Z}$ or $M = ([0:127] \times \mathbb{Z})2$ is the set of all (pairs of) notes. The ith score document is then a finite subset of M. (For technical reasons, in this last example we will replace M by the supersets $\mathbb{Z}^2$ and $(\mathbb{Z}^2)2$, respectively.)

A content-based query—after a possible preprocessing step—is again a finite subset Q of M. Hence, in our three examples, Q is a set of terms, a sequence of words, or a set of (pairs of) notes, respectively.

To define more generally shifted versions of a query Q we use the concept of a group G acting on a set M. Recall that the (multiplicative) group G acts on the set M if there is a map $G \times M \ni (g, m) \mapsto gm \in M$ satisfying for all $g, h \in G$ and $m \in M$: $g(hm) = (gh)m$ and $1_G m = m$.

In this case, M is also called a *G-set*. Such a group action defines an equivalence relation on M:

$$m \sim m' \quad \text{iff} \quad \exists g \in G : gm = m'.$$

The equivalence class containing $m \in M$ is the so-called *G-orbit*

$$Gm := \{gm \mid g \in G\}.$$

Hence the G-set M decomposes into the disjoint union of G-orbits. If R is a *transversal* of the G-orbits, i.e., a set of representatives containing exactly one element of each orbit, then

$$M = \bigsqcup_{r \in R} Gr.$$

By definition, the G-action on M is *transitive* if there is only one G-orbit, otherwise the G-action is *intransitive*.

If G acts on M then G also acts on $P(M) := \{Q \mid Q \subseteq M\}$, the power set of M, via

$$gQ := \{gq \mid q \in Q\},$$

where $g \in G$ and $Q \subseteq M$. As Q and gQ have the same cardinality, G acts on the set of all finite subsets of M, i.e., on the documents and queries over M. Thus the group elements precisely allow us to specify what is meant by a G-shift of a query. Furthermore, as we will see below, we can vary the group G to exploit the user's prior knowledge. With this concept we can define a first information retrieval task.

Definition 1 *(Exact partial $(G, \mathcal{D})$-matches)*
Let $\mathcal{D} = (D_1, \ldots, D_N)$ be a collection over the G-set M. For a query Q over M the set of all exact $(G, \mathcal{D})$-matches is defined as

$$G_{\mathcal{D}}(Q) := \{(g, i) \in G \times [1 : N] \mid gQ \subseteq D_i\}. \quad \bullet$$

In the text retrieval example, G is the trivial group. Hence there is no loss of information if we simply replace $(g, i) \in G_{\mathcal{D}}(Q)$ by i. Thus $G_{\mathcal{D}}(Q) \equiv \{i \in [1 : N] \mid Q \subseteq D_i\}$ equals the set $H_{\mathcal{D}}(Q)$ from equation (1).

In the full-text retrieval example, the additive group $G = \mathbb{Z}$ acts on $\mathbb{Z} \times W$ by $(t, [j, w]) \mapsto [t + j, w]$. In this case, $G_{\mathcal{D}}(Q)$ consists of all pairs (t, i) such that the time-shifted version $t + Q$ of Q is completely contained in D_i.

Finally, in the score retrieval scenario, the additive group $G = \mathbb{Z}^2$ acts on $M = \mathbb{Z}^2$ by vector addition. By the way, this action is transitive. Again, $G_{\mathcal{D}}(Q)$ is the set of all pairs $((p, t), i)$ such that $(p, t) + Q$ is completely contained in the ith document.

So far, in all examples a commutative group is acting. Here is a classical example of a noncommutative group action.

A 2D color image I is modeled as a finite subset of $\mathbb{R}^2 \times C$, where $[(x,y),c] \in I$ describes the color information $c \in C$ of I at position $(x,y) \in \mathbb{R}^2$. Let G denote the group of similarity transformations in $\mathbb{R}^2$. This noncommutative group is generated by rotations, translations and uniform scalings. If $Q \subset \mathbb{R}^2 \times C$ is a fragment of a 2D color image, then G acts on such fragments of images by $gQ := \{[g(x,y),c] \mid [(x,y),c] \in Q\}$. Thus gQ is a rotated, translated and rescaled version of Q.

If $\mathcal{D}$ is a collection over the G-set M then our index will consist of $(G,\mathcal{D})$-*inverted files* or *lists*, defined by

$$G_{\mathcal{D}}(m) := \{(g,i) \in G \times [1:N] \mid gm \in D_i\},$$

for $m \in M$. One easily shows that for a query Q the corresponding set of all exact partial $(G,\mathcal{D})$-matches may be computed as the intersection of all $(G,\mathcal{D})$-inverted files specified by Q:

$$G_{\mathcal{D}}(Q) = \bigcap_{q \in Q} G_{\mathcal{D}}(q).$$

Thus to obtain all matches, it suffices to have access to all inverted lists $G_{\mathcal{D}}(q)$. If M or G are infinite or of large finite cardinality, it might be impossible or impractical to store all inverted lists. However, in a number of cases, we can overcome this problem. The crucial observation is that the inverted lists of all elements in one G-orbit are closely related.

LEMMA 2 $G_{\mathcal{D}}(gm) = G_{\mathcal{D}}(m)g^{-1} := \{(hg^{-1},i) \mid (h,i) \in G_{\mathcal{D}}(m)\}$.

Proof. The following chain of equivalences proves our claim:

$$\begin{aligned}(h,i) \in G_{\mathcal{D}}(m) \quad &\iff \quad hm \in D_i \iff hg^{-1}(gm) \in D_i \\ &\iff \quad (hg^{-1},i) \in G_{\mathcal{D}}(gm). \quad \bullet\end{aligned}$$

Thus if the multiplication in the group G is not too involved, we can quickly recover $G_{\mathcal{D}}(gm)$ from $G_{\mathcal{D}}(m)$, which might lead to dramatic storage savings. To be more precise, let R be a transversal of the G-orbits of M. Then every element $m \in M$ can be written as $m = g_m r_m$ with a uniquely determined $r_m \in R$ and an element $g_m \in G$ which is unique modulo $G_m := \{g \in G \mid gm = m\}$, the stabilizer subgroup of m. By the last lemma it is sufficient to store only the inverted lists corresponding to the transversal R. Then the remaining lists can be computed on demand according to the following result.

THEOREM 3 *For the set of all exact partial $(G,\mathcal{D})$-matches w.r.t. the query $Q \subseteq M$ the following formula holds:*

$$G_{\mathcal{D}}(Q) = \bigcap_{q\in Q} G_{\mathcal{D}}(r_q)g_q^{-1}.$$

Proof. Combine $G_{\mathcal{D}}(Q) = \cap_{q\in Q} G_{\mathcal{D}}(q)$ and $q = g_q r_q$ with the last lemma. •

How time and space consuming is the computation of all exact partial $(G,\mathcal{D})$-matches along the formula of the last theorem? Let us first discuss storage requirements.

THEOREM 4 *With the notation of the last theorem suppose that the stabilizer subgroups G_m for all $m \in M$ are trivial. Then the sum of the lengths of all $(G,\mathcal{D})$-inverted lists corresponding to a transversal R of the G-orbits of M equals the sum of the cardinalities of all documents:*

$$\sum_{r\in R} \text{length}(G_{\mathcal{D}}(r)) = \sum_{i\in[1:N]} |D_i|.$$

Proof. As all stabilizers are trivial, each $m \in M$ has a unique decomposition $m = g_m r_m$ with $g_m \in G$ and $r_m \in R$. Thus each $m \in D_i$ contributes exactly one entry to exactly one list: $(g_m, i) \in G_{\mathcal{D}}(r_m)$. •

If $m \in D_i$ has a nontrivial stabilizer G_m and if $m = g_m r$ with $g_m \in G$ and $r \in R$, then m contributes exactly $|G_m|$ entries to the inverted list $G_{\mathcal{D}}(m)$, namely all pairs of the form $(g_m g, i)$ with $g \in G_r$. So if all stabilizers are small ($|G_m| \leqslant c$, say) then a slight modification of the above reasoning shows that the above equality can be replaced by $\sum_{r\in R} \text{length}(G_{\mathcal{D}}(r)) \leqslant c\cdot\sum_{i\in[1:N]} |D_i|$. If there are large stabilizers, then the above formula should not be applied directly. Instead, a redesign of the set M of elementary data objects might be helpful to force trivial stabilizers. This trick will be discussed below.

In what follows we concentrate on the case that all stabilizers are trivial. How fast can we compute an intersection like $\cap_{q\in Q} G_{\mathcal{D}}(r_q)g_q^{-1}$? To settle this question let us first discuss the problem of generating the inverted lists. To accelerate the intersection task, we first define a linear ordering on the group G and suppose that we have an algorithm that efficiently decides for $g, h \in G$ whether $g = h$, $g < h$, or $g > h$. For the moment, we also assume that we know a transversal R of the G-orbits of M and that we know an efficient algorithm for computing $M \ni m \mapsto (g_m, r_m) \in G \times R$, where $m = g_m r_m$. Under these assumptions we may, for each element in all the documents, efficiently compute the corresponding inverted file: $m = g_m r_m \in D_i$ yields the entry (g_m, i) in $G_{\mathcal{D}}(r_m)$. Finally, we linearly order the entries in each inverted file

according to their first component, which is an element of the linearly ordered group G. Note that not all G-orbits have to be involved in the documents of a particular database $D_1, \ldots, D_N$. Suppose that $R' \subseteq R$ is minimal with $\cup_{r\in R'} Gr \supseteq \cup_{i\in[1:N]} D_i$, then we have a total of $|R'|$ inverted files. If the group operation is compatible with the linear ordering of the group, i.e., $x < y$ implies $xg < yg$, for all $x, y, g \in G$, then the linear ordering of the inverted files $G_{\mathcal{D}}(r)$ for $r \in R$ can be used directly when computing $G_{\mathcal{D}}(Q)$ using the formula $G_{\mathcal{D}}(Q) = \cap_{q\in Q} G_{\mathcal{D}}(r_q)g_q^{-1}$. According to our assumption, each $G_{\mathcal{D}}(r_q)$ is linearly ordered, and hence so is $G_{\mathcal{D}}(r_q)g_q^{-1}$. Consequently, we have to compute the intersection of n linearly ordered lists of length $\ell_1 \leqslant \ldots \leqslant \ell_n$, say. If λ_i is the cardinality of the intersection of the first i lists, then computing $G_{\mathcal{D}}(Q)$ with the binary method requires at most $\sum_{i=1}^{n-1} \lambda_i \log \ell_{i+1} \leqslant (n-1)\ell_1 \log \ell_n$ comparisons in G as well as at most n inversions and $\ell_1 + \ldots + \ell_n$ multiplications in G.

Surprisingly, a similar complexity result can even be obtained when the linear ordering of G is not compatible with the multiplications in G. This is based on the following simple but very useful observation.

LEMMA 5 *Let a, b be elements and A, B subsets of the group G. Then $Aa \perp Bb = (Aab^{-1} \perp B)b = (A \perp Bba^{-1})a$, for $\perp \in \{\cap, \cup, \setminus\}$.*

Proof. Straightforward. •

In the worst case, both Aa and Bb are not linearly ordered in contrast to A and B. Nevertheless, when computing, e.g., the intersection $Aa \cap Bb$ we can use either the linear ordering of A by computing $(A \cap Bba^{-1})a$ or that of B by computing $(Aab^{-1} \cap B)b$.

THEOREM 6 *Let Q be a query of size n, $Q = \{q_1, \ldots, q_n\}$. Suppose that the lists $G_{\mathcal{D}}(q_i)$ are ordered according to their lengths $\ell_1 \leqslant \ldots \leqslant \ell_n$, where $\ell_i := |G_{\mathcal{D}}(q_i)|$. For $j \in [1:n]$ let λ_j denote the cardinality of the intersection $\cap_{i\in[1:j]} G_{\mathcal{D}}(q_i)$. Given the linearly ordered lists $G_{\mathcal{D}}(r)$ for all $r \in R$, the list $G_{\mathcal{D}}(Q)$ of all exact partial $(G, \mathcal{D})$-matches of Q can be computed with $\sum_{i=1}^{n-1} \lambda_i \log \ell_{i+1} \leqslant (n-1)\ell_1 \log \ell_n$ comparisons in G, along with n decompositions $q_i \mapsto (g_i, r_i) \in G \times R$ satisfying $q_i = g_i r_i$, n inversions $g_i \mapsto g_i^{-1}$, $n-1$ multiplications $g_i^{-1} g_{i+1}$, and $\sum_{i=1}^{n} \lambda_i \leqslant n\ell_1$ multiplications in G.*

Proof. Let $q_i = g_i r_i$ with $g_i \in G$ and $r_i \in R$. Then

$$G_{\mathcal{D}}(Q) = \bigcap_{i=1}^{n} G_{\mathcal{D}}(q_i) = \bigcap_{i=1}^{n} G_{\mathcal{D}}(r_i) g_i^{-1}.$$

According to the last lemma we compute $G_{\mathcal{D}}(Q)$ as follows (here shown for the case $n = 3$ using the shorthand $H_i := G_{\mathcal{D}}(r_i)$):

$$H_1 g_1^{-1} \cap H_2 g_2^{-1} \cap H_3 g_3^{-1} = ((H_1 g_1^{-1} g_2 \cap H_2) g_2^{-1} g_3 \cap H_3) g_3^{-1}.$$

This formula suggests the following procedure: first, compute all decompositions $q_i \mapsto (g_i, r_i)$. Then invert all g_i and compute all $g_i^{-1} g_{i+1}$. Afterwards, inductively compute linearly ordered sets

$$\Lambda_j := (\bigcap_{i \in [1:j]} H_i g_i^{-1}) g_j.$$

Note that $\Lambda_1 = H_1$ is already linearly ordered. As $\Lambda_{j+1} = \Lambda_j g_j^{-1} g_{j+1} \cap H_j$ and H_j is linearly ordered, we can compute this intersection using the binary method. We finish the proof by mentioning that Λ_j has cardinality λ_j and $\Lambda_n g_n^{-1}$ equals $G_{\mathcal{D}}(Q)$. •

Roughly speaking, the last theorem tells us that the number of operations to compute $G_{\mathcal{D}}(Q)$ is at most the product of the cardinality of Q, the length of the shortest list and the logarithm of the longest list among all lists corresponding to Q. Hence it is profitable to have many short lists. But what do we do if G acts transitively on M? In this case one should look at intransitive G-sets, closely related to M. Here are two examples: if $n \geqslant 2$ then G acts on $\mathcal{P}_{\leqslant n}(M) := \{X \subseteq M \mid 0 < |X| \leqslant n\}$ by $gX := \{gx \mid x \in X\}$ and on M^n by $g(m_1, \ldots, m_n) := (gm_1, \ldots, gm_n)$. Both actions are intransitive. The next result shows a close connection between exact partial matches in various G-sets.

THEOREM 7 *Let $\mathcal{D} = (D_1, \ldots, D_N)$ denote a document collection over the G-set M. This induces new collections $\mathcal{D}^n := (D_1^n, \ldots, D_N^n)$. Similarly, for a query $Q \subseteq M$ and $\mathcal{P}_{\leqslant n}(\mathcal{D}) := (\mathcal{P}_{\leqslant n}(D_1), \ldots, \mathcal{P}_{\leqslant n}(D_N))$ we associate the queries $\mathcal{P}_{\leqslant n}(Q) \subseteq \mathcal{P}_{\leqslant n}(M)$ and $Q^n \subseteq M^n$. The corresponding sets of exact partial matches are equal:*

$$G_{\mathcal{D}}(Q) = G_{\mathcal{P}_{\leqslant n}(\mathcal{D})}(\mathcal{P}_{\leqslant n}(Q)) = G_{\mathcal{D}^n}(Q^n).$$

Proof. For $(g, i) \in G \times [1 : N]$ we have

$$\begin{aligned}
(g,i) \in G_{\mathcal{D}}(Q) &\iff gQ \subseteq D_i \\
&\iff \forall X \in \mathcal{P}_{\leqslant n}(gQ) = g\mathcal{P}_{\leqslant n}(Q) :\ X \in \mathcal{P}_{\leqslant n}(D_i) \\
&\iff g\mathcal{P}_{\leqslant n}(Q) \subseteq \mathcal{P}_{\leqslant n}(D_i) \\
&\iff (g,i) \in G_{\mathcal{P}_{\leqslant n}(\mathcal{D})}(\mathcal{P}_{\leqslant n}(Q)).
\end{aligned}$$

This proves the first equality. The second equality follows in a similar way using $(gQ)^n = gQ^n$. •

It should be clear that the above intransitive G-actions are just two generic constructions which are available in every case. Depending on a particular application one will possibly succeed with smaller G-sets, as, e.g., the action on $\mathcal{P}_n(M) = \{X \subseteq M \mid |X| = n\}$ instead of $\mathcal{P}_{\leqslant n}(M) = \{X \subseteq M \mid 0 < |X| \leqslant n\}$.

Here are some specific examples. $G = (\mathbb{R}^3, +)$ acts transitively on $M := \mathbb{R}^3$ by vector addition. The induced actions of G on both $\mathcal{P}_2(M)$ and $M2$ are intransitive. In fact, the G-orbit of $\{x, y\} \in \mathcal{P}_2(M)$ is characterized by $\pm(x - y)$, whereas the G-orbit of $(x, y) \in M2$ is characterized by $x - y$. Thus if the documents consist of finite subsets D_i of $\mathbb{R}^3$ and the group of all translations is the group in question, then it is profitable to view pairs or 2-sets of vectors in $\mathbb{R}^3$ as the new elementary data objects. If, however, the group G_s of all similarity transformation in 3-space (generated by all translations, rotations and uniform scalings) is the group of choice, then $M2$ decomposes only into two G_s-orbits represented by $([0, 0, 0], [0, 0, 0])$ and $([0, 0, 0], [0, 0, 1])$, respectively. Furthermore, G_s acts transitively on $\mathcal{P}_2(M)$. However, $\mathcal{P}_3(M)$ decomposes into many G_s-orbits. These orbits correspond to the classes of congruent triangles in Euclidean 3-space. Thus if $G = G_s$ then one should switch from $M = \mathbb{R}^3$ to $\mathcal{P}_3(M)$. Besides many G_s-orbits by this switch we also obtain small stabilizers. In fact, all stabilizers are isomorphic to a subgroup of the dihedral group of order 6.

A switch from M to $\mathcal{P}_k(M)$ has the drawback that a document $D_i \subseteq M$ with d_i elements has to be replaced by the new document $\mathcal{P}_k(D_i)$ which consists of $\binom{d_i}{k}$ elements. Fortunately, in most applications a query Q will typically be not a random but a structured subset of M. For example, if (M, d) is a metric space and $\theta > 0$ a prescribed constant, a possible property of Q could be its θ-connectedness, i.e., there is an enumeration of Q, $Q = \{q_1, \ldots, q_n\}$, such that $d(q_{i-1}, q_i) \leqslant \theta$, for all $i \in [2 : n]$. Thus instead of $\mathcal{P}_k(D_i)$ in this case it is sufficient to work with the typically much smaller set $D_{i,k,\theta} := \{X \in \mathcal{P}_k(D_i) \mid X \text{ is } \theta\text{-connected}\}$. A query $Q = \{q_1, \ldots, q_n\}$ with $0 < d(q_{i-1}, q_i) \leqslant \theta$ should then be replaced by $Q' = \{\{q_{\lambda k+1}, \ldots, q_{(\lambda+1)k}\} \mid 0 \leqslant \lambda \leqslant \lfloor n/k \rfloor\} \cup \{\{q_{n-k+1}, \ldots, q_n\}\}$.

4. Fault Tolerance

So far, we have mainly discussed *exact* partial matches. Typically, there are many sources of impreciseness. Think, e.g., of a non professional user humming a melody into a microphone. Another example could be the search in a database of signals where the queries consist of noisy or lossy compressed versions of the original signals. In all of those

cases, a certain degree of fault tolerance is required. Now we will discuss two general kinds of fault tolerance: mismatches and fuzzy queries.

We start with mismatches. Let Q be a query and D a document over M. The elements of $Q \cap D$ form the matching part, whereas the elements in Q that do not belong to D are called the *mismatches*. Thus for a fixed non-negative integer k and a query Q the set

$$G^k_{\mathcal{D}}(Q) := \{(g,i) \in G \times [1:N] \mid |gQ \setminus D_i| \leqslant k\}$$

specifies all partial $(G,\mathcal{D})$-matches with at most k mismatches. Obviously, $G^0_{\mathcal{D}}(Q)$ equals $G_{\mathcal{D}}(Q)$. For $k > 0$ and $(g,i) \in G^k_{\mathcal{D}}(Q)$ the transformed query gQ is contained in D_i with up to k mismatching elements. Fig. 5 illustrates the concept of mismatches.

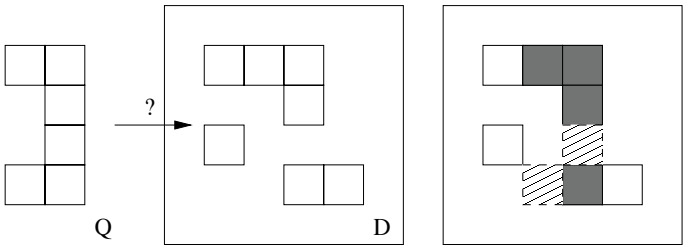

Figure 5. A query Q (left) is matched to a certain location of a document D (middle) using a shift by g. On the right, the matching positions $D \cap gQ$ are represented by gray boxes, whereas the two mismatches $gQ \setminus D$ are marked by dashed boxes.

To determine $G^k_{\mathcal{D}}(Q)$, we use a dynamic programming approach. Let $Q = \{q_1, \ldots, q_n\}$ and define $G_j := G_{\mathcal{D}}(q_j)$ as well as $\Gamma_j := G_1 \cup \ldots \cup G_j$. We inductively define credit functions $C_j \colon \Gamma_j \to \mathbb{Z}$ as follows. $\Gamma_1(\gamma) := k+1$, for every $\gamma \in \Gamma_1$. For $2 \leqslant j \leqslant n$ we define

$$C_j(\gamma) := \begin{cases} C_{j-1}(\gamma) & \text{if } \gamma \in \Gamma_{j-1} \cap G_j \\ C_{j-1}(\gamma) - 1 & \text{if } \gamma \in \Gamma_{j-1} \setminus G_j \\ k+2-j & \text{if } \gamma \in G_j \setminus \Gamma_{j-1}. \end{cases}$$

THEOREM 8 *The elements of Γ_n with a positive credit are just all partial $(G,\mathcal{D})$-matches with at most k mismatches:*

$$G^k_{\mathcal{D}}(Q) = \{\gamma \in \Gamma_n \mid C_n(\gamma) > 0\}.$$

Proof. Let $(g,i) \in \Gamma_j$ and $Q_j := \{q_1, \ldots, q_j\}$. By induction on j we show that $C_j(g,i) = k + 1 - |gQ_j \setminus D_i|$. The start $j = 1$ is clear. To prove the inductive step $(j-1 \to j)$ we distinguish three cases.

Case 1: $(g,i) \in \Gamma_{j-1} \cap G_j$. As $gq_j \in D_i$, $gQ_j \setminus D_i = gQ_{j-1} \setminus D_i$. Thus $C_j(g,i) := C_{j-1}(g,i) = k + 1 - |gQ_{j-1} \setminus D_i| = k + 1 - |gQ_j \setminus D_i|$.

Case 2: $(g,i) \in \Gamma_{j-1} \setminus G_j$. As $gq_j \notin D_i$, $|gQ_j \setminus D_i| = |gQ_{j-1} \setminus D_i| + 1$. Thus $C_j(g,i) := C_{j-1}(g,i) - 1 = k + 1 - (|gQ_{j-1} \setminus D_i| + 1) = k + 1 - |gQ_j \setminus D_i|$.

Case 3: $(g,i) \in G_j \setminus \Gamma_{j-1}$. Then $gq_j \in D_i$, but $gq_\ell \notin D_i$, for all $\ell \in [1 : j-1]$. Hence $|gQ_j \setminus D_i| = j - 1$. Thus $C_j(g,i) := k + 2 - j = k + 1 - (j-1) = k + 1 - |gQ_j \setminus D_i|$.

As $Q_n = Q$, we get $C_n(g,i) = k + 1 - |gQ \setminus D_i|$, for every $(g,i) \in \Gamma_n$. Thus $C_n(g,i) > 0$ iff $|gQ \setminus D_i| \leqslant k$, i.e., $(g,i) \in G^k_{\mathcal{D}}(Q)$. •

To perform a complexity analysis of k-mismatch search, we first note that $C_j(\gamma) \leqslant C_{j-1}(\gamma)$, for all $\gamma \in \Gamma_{j-1}$. Hence in a practical implementation we can replace Γ_j by $\Gamma^j := \Gamma_j \setminus \{\gamma \in \Gamma_{j-1} \mid C_{j-1}(\gamma) \leqslant 0\}$. Then $\Gamma_j = \Gamma^j$, for all $j \in [1 : k+1]$ and $\Gamma 1 \subseteq \Gamma 2 \subseteq \ldots \subseteq \Gamma^{k+1} \supseteq \Gamma^{k+2} \supseteq \ldots \supseteq \Gamma^n$. Let ℓ_j and μ_j denote the lengths of G_j and Γ^j, respectively. With Lemma (3.5) and a similar technique as in the proof of Theorem (3.6) we can compute $G^k_{\mathcal{D}}(Q)$ with at most $\sum_{j=1}^{n-1} \min\{\ell_{j+1} \log \mu_j, \mu_j \log_{j+1}\} \leqslant L \log \mu_{k+1}$ comparisons, where L denotes the total length of all involved lists. In addition, we have to perform $2n + \sum_{j=1}^{n-1} \min\{\ell_{j+1}, \mu_j\}$ multiplications or inversions in the group.

Let us now turn to fuzzy queries. Recall that a fuzzy query over M consists of a sequence $\mathbf{F} = (F_1, \ldots, F_n)$ of n finite sets F_i of alternatives. We associate to such an $\mathbf{F}$ a family of ordinary queries

$$Q(\mathbf{F}) := \{\{q_1, \ldots, q_n\} \mid \forall i : \ q_i \in F_i\}.$$

The set $G_{\mathcal{D}}(\mathbf{F}) := \{(g,i) \mid \exists Q \in Q(\mathbf{F}) : \ gQ \subseteq D_i\}$ specifies the set of all exact partial $(G,\mathcal{D})$-matches w.r.t. the fuzzy query $\mathbf{F}$. Obviously, $G_{\mathcal{D}}(\mathbf{F})$ is the union of all $G_{\mathcal{D}}(Q)$, where Q ranges over all ordinary queries associated to $\mathbf{F}$. As already mentioned, if the F_i are pairwise disjoint, then $\mathbf{F}$ consists of $\prod_{i=1}^n |F_i|$ many ordinary queries. So, the naive algorithm that separately computes sets $G_{\mathcal{D}}(Q)$ for each $Q \in Q(\mathbf{F})$ and finally merges those sets is rather inefficient. Fortunately, the following result indicates an efficient algorithm to compute $G_{\mathcal{D}}(\mathbf{F})$.

THEOREM 9 *Let $\mathcal{D} = (D_1, \ldots, D_N)$ be a collection of documents over the G-set M. If $\mathbf{F} = (F_1, \ldots, F_n)$ is a sequence of subsets of M then the set $G_{\mathcal{D}}(\mathbf{F})$ of all exact partial $(G,\mathcal{D})$-matches w.r.t. the fuzzy query $\mathbf{F}$ may be computed using the following formula*

$$G_{\mathcal{D}}(\mathbf{F}) = \bigcap_{j \in [1:n]} \Big(\bigcup_{q \in F_j} G_{\mathcal{D}}(q) \Big) = \bigcap_{j \in [1:n]} \Big(\bigcup_{q \in F_j} G_{\mathcal{D}}(r_q) g_q^{-1} \Big).$$

Proof. $(g,i) \in G_{\mathcal{D}}(\mathbf{F})$ iff $gQ \subseteq D_i$, for some $Q \in Q(\mathbf{F})$. Equivalently, there exists an element $q_j \in F_j$ satisfying $gq_j \in D_i$, for all $j \in [1 : n]$, i.e., $(g,i) \in \cup_{q_j \in F_j} G_{\mathcal{D}}(q_j)$, for all j, proving our claim. •

The complexity analysis for computing $G_{\mathcal{D}}(\mathbf{F})$ is straightforward and left to the reader. It turns out that $G_{\mathcal{D}}(\mathbf{F})$ may be computed with a number of comparisons which is—modulo logarithmic factors—linear in the total number of entries in all involved lists.

To incorporate prior user knowledge, we consider subgroups $U < G$. Then, for a transversal $R \ni 1_G$ of U's cosets in G we have $G = \sqcup_{r \in R} Ur$. Using this decomposition, G- and U-inverted lists are connected by

$$G_{\mathcal{D}}(m) = \bigsqcup_{r \in R} U_{\mathcal{D}}(rm)r,$$

for each $m \in M$. Then, a speed-up in query processing when restricting ourselves to subgroups $U < G$ may result from the fundamental property

$$G_{\mathcal{D}}(Q) = \bigsqcup_{r \in R} U_{\mathcal{D}}(rQ)r,$$

for all $Q \subseteq M$. Hence, $G_{\mathcal{D}}(Q)$ consists of many lists of the form $U_{\mathcal{D}}(rQ)r$. Now, assuming prior knowledge about the coset of a match $g \in G$ w.r.t. U, as in the above example where a user knows about the exact metrical position of a query, we only need to determine the list for the case $r = 1_G$, i.e., $U_{\mathcal{D}}(Q)$.

5. Applications, Prototypes, and Test Results

In this section we give an overview on several applications of the proposed indexing and search technique. A more detailed treatment for the case of music retrieval may be found in our related work ([Clausen and Kurth, 2003]). We describe prototypic implementations and give some test results demonstrating time- and space-efficiency of the proposed algorithms. Recall that for each application we have to specify an underlying set M of elementary objects, a group G operating on this set, and a transversal $R \subset M$ of G-orbits to specify the inverted file index $\{G_{\mathcal{D}}(r) \mid r \in R\}$.

5.1 Content–Based Music Retrieval

Score-based polyphonic search has been used as a running example within this paper. In our PROMS-system ([Clausen et al., 2000]) we used the set $M := \mathbb{Z} \times [0 : 127]$ of notes consisting of onset-times and MIDI-pitches. The search is carried out w.r.t. the groups $G := (\mathbb{Z}, +)$ and $V := (16\mathbb{Z}, +)$ of time-shifts, where G shifts by metrical positions and V by whole measures each consisting of 16 metrical positions. As discussed above, the latter models prior knowledge about the metrical

position of a query within a measure. The corresponding transversals R are defined as above.

Our database consists of 12,000 classical pieces of music given in the MIDI format. The pieces consist of a total of about 33 million notes. Response times for queries of various lengths are summarized in Table 1. The response times for each query length were averaged over 100 randomly generated queries. As the table demonstrates, our query processing is very fast. In addition, the index requires only a small amount of disk space and indexing of 330 MB of polyphonic music takes only 40 seconds. The uncompressed index requires 110 MB of disk space. Compressing the inverted lists using Golomb coding results in reducing the space requirement to 22 MB.

a	4	8	12	16	20	30	50	100
b	51	86	92	97	100	107	125	159
c	1	5	7	10	12	19	31	64

Table 1. Average total system response time (row b) in ms for different numbers of notes per query (row a). Row c: Disk access time for fetching inverted lists. (Pentium II, 333 MHz, 256 MB RAM).

5.2 Audio Identification

The task of audio identification may be described as follows. Given a short part q of an audio track and a database $x_1, \ldots, x_N$ of full-size audio tracks, locate all occurrences of q within the database tracks. In this, a pair (t, i) determines an occurrence of q in x_i iff $q = x_i[t : t + |q| - 1]$. Note that this problem may be stated in terms of our group-based approach since an audio signal $s : \mathbb{Z} \to \mathbb{R}$ may be interpreted via its graph as a subset $S \subset \mathbb{Z} \times \mathbb{R}$ where $G = (\mathbb{Z}, +)$ operates by addition in the $\mathbb{Z}$- (time) component. For robustness- and space-efficiency reasons, audio signals are preprocessed using a G-invariant feature extractor $F : \mathbb{R}^{\mathbb{Z}} \to \mathcal{P}(\mathbb{Z} \times X)$, where X denotes a set of feature classes (in this case G-invariance denotes the usual *time*-invariance). As an illustration we sketch a feature extractor which extracts significant local maxima from a smoothed version of a signal. For a detailed treatment of more robust feature extractors, we refer to [Ribbrock and Kurth, 2002]. F will be composed from several elementary *operators* which are each maps on the signal space, i.e., maps $\mathbb{R}^{\mathbb{Z}} \to \mathbb{R}^{\mathbb{Z}}$. First, an input signal s is smoothed by linear filtering. The corresponding operator is $C_f[s] : n \mapsto \sum_{k \in \mathbb{Z}} f(k)s(n - k)$, where f denotes a signal of finite support. Next,

K-significant local maxima are extracted by an operator

$$M_K[x] : n \mapsto \begin{cases} x(n) & \text{if } x(n-K) < \ldots < x(n) \wedge \\ & x(n) > \ldots > x(n+K), \\ 0 & \text{otherwise.} \end{cases}$$

The resulting signal is again processed by an operator $M'_{K'}$ extracting local maxima. $M'_{K'}$ is defined exactly as M_K, but regards only the support of the input signal. $M'_{K'}$ usually returns a very sparse output signal. The operator Δ assigns to each non-zero position of an input sequence the distance to the previous nonzero position (provided existence), and zero otherwise. Finally, let $\mathcal{Q}_{|X|}$ denote an, e.g., linear, quantizer which reduces a signal's amplitude to $|X|$ feature classes. Note that $\mathcal{Q}_{|X|}$ is an operator $\mathbb{R}^{\mathbb{Z}} \to \mathcal{P}(\mathbb{Z} \times X)$ which in addition to quantization discards zero-positions of a signal. Then our feature extractor may be written as $F := \mathcal{D}_c \circ \Delta \circ M'_{K'} \circ M_K \circ C_f$. In an example we could choose $K = 5$, $K' = 3$, and $X = [1 : 50]$. To construct our search index, we calculate $F[x_i] \subseteq \mathbb{Z} \times X$ for each signal x_i of our database. In this $\{[0, x] \mid x \in X\}$ serves as a transversal for index construction.

We briefly summarize some results of our extensive tests in the audio identification scenario. Our database consists of 4500 full-size audio tracks. This approximately amounts to 180 GB of original data or 13 days of high quality audio. Using the above significant maxima as features, we obtain an (uncompressed) index size of about 128 MB which is a compression ratio of about 1:1,400 as compared to the original data. Using different feature extraction methods, the index size may be further reduced to sizes of 1:5,000 or even lower. The query times range from only a few milliseconds (higher quality queries) to about one second. The required length of a query signal depends on the feature extractor and ranges from a few fractions of a second (significant maxima features) to 5-15 seconds (robust features and low quality queries) ([Clausen and Kurth, 2003]).

5.3 Content-Based Image Retrieval

In content-based 2D- or 3D-retrieval we are interested in finding possibly translated, rotated, or (uniformly) scaled versions of a query object in an underlying database. In this overview we shall consider translations and rotations only. Hence, the groups of interest are the group T_n of translations in $\mathbb{R}^n$, the orthogonal group O_n, and the group of *Euclidean motions* $\mathcal{E}_n := T_n \rtimes O_n$.

In content-based image retrieval, we are working with 2D images $D \subset \mathbb{R}^2 \times \mathbb{N} = P$ and are hence interested in the groups T_2 and $\mathcal{E}_2$. Assume a suitable feature extractor yielding a set of features $F(D) \subset \mathbb{R}^2 \times X$ for an

image D. T_2 acts on P's first two components as described above. Hence, after feature extraction, we may create an index based on the transversal $\{[0,0,x] \mid x \in X\} \subset P$. In our extensive tests described in [Röder, 2002] we investigated several kinds of feature extractors including corner detectors, gray value statistics and histograms.

Looking at retrieval under the group $\mathcal{E}_2$ acting on 2D points from P, we face the problem that each point $x \in \mathbb{R}^2$ is mapped to any other point $y \in \mathbb{R}^2$ by infinitely many elements from $\mathcal{E}_2$, resulting in inverted lists of infinite size. Hence we resort to indexing line segments, which are modeled by the set $\mathcal{P}_2(M) := \{L \subset M \mid |L| = 2\}$ of all two–element subsets of $\mathbb{R}^2$. Using $M' := \mathcal{P}_2(\mathbb{R}^2) \times X$ the line segments $\{[(0,a),(0,-a),x] \mid x \in X, a \geqslant 0\}$ serve as a transversal for indexing. In [Röder, 2002], several types of features were tested in the latter setting.

5.4 Searching 3D-Scenes

We investigated content-based search in 3D scenes for the case of a database of VRML (Virtual Reality Modeling Language) documents ([Mosig, 2001]). To obtain feasible inverted lists, the elementary objects were chosen to be all 3-sets in $\mathbb{R}^3$, i.e., $M = \mathcal{P}_3(\mathbb{R}^3)$, interpreted as the sets of all triangles in $\mathbb{R}^3$. Hence for indexing, all VRML documents were converted into documents consisting of triangles only. Indexing was performed for the groups T_3 of 3D translations and for the group $\mathcal{E}_3$ of Euclidean motions in $\mathbb{R}^3$. As a transversal of the $\mathcal{E}_3$-orbits we chose all sets of triangles with the origin as the center of gravity. Additionally, each representative is rotated such that one specific edge runs in parallel to the x-axis, this edge depending on the triangle having one, two, or three different side lengths. This way, one obtains a finite set of inverted lists for this application.

Fig. 6 illustrates the concept of 3D-retrieval using an underlying toy-database of 3D-objects (on the left). The top of the figure shows a part of an object which is used as a query to the database. An index-based search results in one object of the database matching the query (on the right). The position of that object matching the query is highlighted.

6. Related Work and Future Research

The techniques presented in this paper are related to work from across several communities. In this section we try to establish the most important relations to previous as well as ongoing research efforts.

From a classical database point of view, multimedia data may be modeled using a relational framework. In the usual approach, mul-

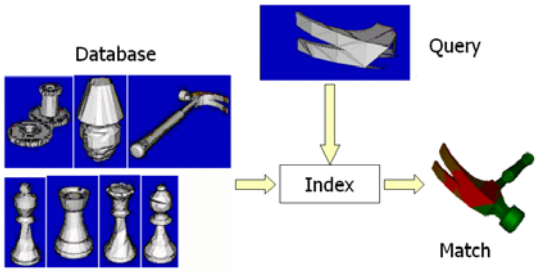

Figure 6. Toy database of 3D-objects (left), query object (top), and matching database object (right). The matching position is highlighted.

timedia documents are preprocessed yielding certain feature vectors. The extracted features are then suitably stored in tables of a relational database ([Santini and Gupta, 2002]). Using relations, it is possible to model complex object dependencies like spatial constraints on regions in an image. Efficient retrieval methods have been proposed using approximate search like hill climbing ([Papadias, 2000]). Although our approach may be extended to a relational setting by introducing permutation groups, the methods proposed in this paper were primarily developed to exploit the structure of the underlying object set M and the group G acting on M (e.g., musical documents are structured by specific time- and pitch- intervals between single notes which are not changed by the group action). In this light, our model constitutes a special case of the general relational setting which, however, allows for very efficient query evaluation.

An important issue in multimedia indexing is the use of multidimensional access structures like k-d- or R*-Trees ([Lu, 2002]). Popular indexing approaches map multimedia documents such as time series to higher dimensional feature sequences and use multidimensional access structures for searching in those structures ([Faloutsos, 1996]). In our approach we, as far as possible, tried to avoid higher dimensional features in order to avoid problems resulting from the dimensionality curse. In indexing audio this became, e.g., possible by exploiting the fixed temporal relationships between the features. On the other hand, when dealing with more complex groups (e.g., the group of Euclidean motions in 3D leads to a 6-parameter representation for each element), our approach is also dependent on efficient algorithms for higher dimensional range and nearest neighbor search ([Agarwal, 1997]).

Specializing to time-series, there has been considerable recent interest in searching a query sequence in large sets of times–series w.r.t. various

distance measures. Examples are Euclidean, ℓ^p, or dynamic time warping distances ([Keogh, 2002, Moon et al., 2002]). Whereas our approach has up to now only been applied to audio signals, its high performance suggests an extension to include general time-series search under the latter distance measures. For the particular case of audio identification, several methods suitable for large data collections have been proposed recently. Among those, the hashing algorithms proposed by ([Haitsma et al., 2001]) are the most similar to our approach, as hash tables are related to inverted files. Our approach has the advantage of needing significantly less memory for storing the index (about 100 MB as compared to 1–2 GB) with otherwise comparably (fast) performance data.

In music retrieval, most of the early work has concentrated on similarity based retrieval of melodies, see, e.g., [Uitdenbogerd and Zobel, 1999]. For a long time, retrieval in *polyphonic* music (see, e.g., [Lemström and Perttu, 2000]) already suffered from a lack of suitable data modeling. Our technique led to a breakthrough in allowing to model as well as efficiently search polyphonic music ([Clausen et al., 2000]). As our approach is up to now mainly focused on modeling and efficient retrieval, the use of music similarity measures is a natural challenge for future work.

An approach which is similar in spirit to our general technique is geometric hashing for object recognition proposed in Computer Vision, see [Wolfson and Rigoutsos, 1997]. This approach shares the modeling of shift operations which are considered for 2D/3D settings as well as the exploitation of the data's structural (geometric) properties. In our approach the data modeling is more general, which yields advantages in designing more efficient fault-tolerant retrieval algorithms.

An extension of our approach ([Clausen and Mosig, 2003]) including general distance measures between query and matching position leads to (shape-) matching problems which have been extensively treated in the area of computational geometry ([Veltkamp, 2001]). It will be a great challenge for future work to try to combine our approach, which is more tuned for use with larger datasets, with the sophisticated geometric matching techniques.

Many approaches to content-based image retrieval have been proposed in the last years, among which perhaps IBM's QBIC system (Query by Image Content) is the most popular. In comparison to those, a considerable advantage of our technique is the natural integration of partial matches and the ability to locate queries as subimages of database images at no additional cost. An interesting direction is the Blobworld approach by [Carson et al., 1999], of region-based image representation and retrieval, where image descriptors are created from a prior segmen-

tation into similarly textured regions. It would be very interesting to combine such texture descriptors with our approach to spatial indexing.

References

Agarwal, P. K. (1997). Geometric range searching. In *Handbook of Comp. Geometry*. CRC.

Carson, C., Thomas, M., Belongie, S., Hellerstein, J. M., and Malik, J. (1999). Blobworld: A system for region-based image indexing and retrieval. In *Third International Conference on Visual Information Systems*. Springer.

Clausen, M., Engelbrecht, R., Meyer, D., and Schmitz, J. (2000). PROMS: A Web-based Tool for Searching in Polyphonic Music. In *Proceedings Intl. Symp. on Music Information Retrieval 2000, Plymouth, M.A., USA*.

Clausen, M. and Kurth, F. (2003). A Unified Approach to Content-Based and Fault Tolerant Music Recognition. IEEE Transactions on Multimedia, Accepted for Publication.

Clausen, M. and Mosig, A. (2003). Approximately Matching Polygonal Curves under Translation, Rotation and Scaling with Respect to the Fréchet-Distance. In *19th European Workshop on Computational Geometry*.

Cormen, T. H., Leiserson, C. E., and Rivest, R. L. (1990). *Introduction to Algorithms*. MIT Press, Cambridge, MA.

Faloutsos, C. (1996). *Searching Multimedia Databases by Content*. Kluwer.

Haitsma, J., Kalker, T., and Oostven, J. (2001). Robust Audio Hashing for Content Identification. In *2nd Intl. Workshop on Content Based Multimedia and Indexing, Brescia, Italy*.

Keogh, E. (2002). Exact indexing of dynamic time warping. In *28th International Conference VLDB, Hong Kong*, pages 406–417.

Lemström, K. and Perttu, S. (2000). SEMEX - An Efficient Music Retrieval Prototype. In *Proceedings Intl. Symp. on Music Information Retrieval 2000, Plymouth, M.A., USA*.

Lu, G. (2002). Techniques and Data Structures for Efficient Multimedia Retrieval Based on Similarity. *IEEE Trans. on Multimedia*, 4(3):372–384.

Moon, Y.-S., Whang, K.-Y., and Han, W.-S. (2002). General Match: A Subsequence Matching Method in Time-Series Databases Based on Generalized Windows. In *SIGMOD Conference*, pages 382–393.

Mosig, A. (2001). Algorithmen und Datenstrukturen zur effizienten Konstellationssuche. Masters Thesis, Department of Computer Science III, University of Bonn, Germany.

Papadias, D. (2000). Hill Climbing Algorithms for Content-Based Retrieval of Similar Configurations. In *Proc. SIGIR, Greece*, pages 240–247.

Ribbrock, A. and Kurth, F. (2002). A Full-Text Retrieval Approach to Content-Based Audio Identification. In *Proc. 5. IEEE Workshop on MMSP, St. Thomas, Virgin Islands, USA*.

Röder, T. (2002). A Group Theoretical Approach to Content-Based Image Retrieval. Masters Thesis, Department of Computer Science III, University of Bonn, Germany.

Santini, S. and Gupta, A. (2002). Principles of Schema Design in Multimedia Data Bases. *IEEE Trans. on Multimedia*, 4(2).

Uitdenbogerd, A. L. and Zobel, J. (1999). Melodic Matching Techniques for Large Music Databases. In *Proc. ACM Multimedia*.

Veltkamp, R. C. (2001). Shape Matching: Similarity Measures and Algorithms. In *Shape Modelling International*, pages 188–199.

Wolfson, H. and Rigoutsos, I. (1997). Geometric Hashing: An Overview. *IEEE Computational Science and Engineering*, 4(4):10–21.

Yoshitaka, A. and Ichikawa, T. (1999). A Survey on Content-Based Retrieval for Multimedia Databases. *IEEE Transactions on Knowlegde and Data Engineering*, 11(1):81–93.