5

content annotation

Current technology does not allow us to extract information automatically from arbitrary media objects. In these cases, at least for the time being, we need to assist search by annotating content with what is commonly referred to as meta-information. In this chapter, we will look at two more media types, in particular audio and video. Studying audio, we will learn how we may combine feature extraction and meta-information to define a data model that allows for search. Studying video, on the other hand, will indicate the complexity of devising a knowledge representation scheme that captures the content of video fragments. Concluding this chapter, we will discuss an architecture for feature extraction for arbitrary media objects.

5.1 audio

The audio media type covers both spoken voice and musical material. In this section we will discuss audio signal, stored in a raw or compressed (digital) format, as well as similarity-based retrieval for musical patterns.

In general, for providing search access to audio material we need, following [MMDBMS], a data model that allows for both mdeta-data (that is information about the media object) and additional attributes of features, that we in prinicple obtain from the media object itself, using feature extraction.

audio data model

- *meta-data* – describing content
- *features* – using feature extraction

As an example of audi meta-data, consider the (meta-data) characterization that may be given for opera librettos.

example

singers – (Opera,Role,Person)
score – ...

transcript – ...

For signal-based audio content, we have to perform an analysis of the audio signal for which we may take parameters such as frequency, velocity and amplitude. For the actual analysis we may have to break up the signal in small windows, along the time-axis. Using feature extraction, we may characterize (signal-based) properties such as indicated below.

feature extraction

- *intensity* – watts/m^2
- *loudness* – in decibels
- *pitch* – from frequency and amplitude
- *brightness* – amount of distortion

For a more detailed treatment of signal-based audio content description, consult [MMDBMS].

In the following we will first give an overview of musical search facilities on the web and then we will discuss similarity-based retrieval of musical patterns in somewhat more depth in the section on *research directions*, In section 6.3, we will have a closer look at feature extraction for arbitrary media types.

research directions – *musical similarity*

In this section on research directions for audio information retrieval, we will study how to provide content-based retrieval facilities based on *similarity* in the musical domain. This material comes from our previous research, part of which has been reported in [OO]. However, here we will look primarily at work that has been done in this field by others.

As concerns musical content, at least for most genres, it appears that we should focus primarily on *melody*, since, as phrased in [Concepts]:

> *"It is melody that makes music memorable: we are likely to recall a tune long after we have forgotten its text."*

Other features, content-based as well as descriptive, may however be used as additional filters in the proces of retrieval.

Melodic searching and matching has been explored mainly in the context of bibliographic tools and for the analysis of (monophonic) repertories [Similarity]. As described in section **??**, many of these efforts have been made available to the general public through the Web. Challenges for the near future are, however, to provide for melodic similarity matching on polyphonic works, and retrieval over very large databases of musical fragments.

In this section we will look in somewhat more detail at the problem of melodic similarity matching. In particular, we will discuss representational issues, matching algorithms and additional analysis tools that may be used for musical information retrieval.

melodic similarity Consider the musical fragment *Twinkle, twinkle little star* (known in the Dutch tradition as *"Altijd is Kortjakje ziek"*), which has been used by Mozart for a series of variations [Mozart]. Now, imagine how you would approach establishing the similarity between the original theme and these variations. As a matter of fact, we discovered that exactly this problem had been tackled in the study reported in [Compare], which we will discuss later. Before that, we may reflect on what we mean by the concept of a *melody*. In the aforementioned variations the original melody is disguised by, for example, decorations and accompaniments. In some variations, the melody is distributed among the various parts (the left and right hand). In other variations, the melody is only implied by the harmonic structure. Nevertheless, for the human ear there seems to be, as it is called in [Concepts], a *'prototypical'* melody that is present in each of the variations.

When we restrict ourselves to pitch-based comparisons, melodic similarity may be established by comparing profiles of pitch-direction (up, down, repeat) or pitch contours (which may be depicted graphically). Also, given a suitable representation, we may compare pitch-event strings (assuming a normalized pitch representation such as position within a scale) or intervallic contours (which gives the distance between notes in for example semitones). Following [Concepts], we may observe however that the more general the system of representation, the longer the (query) *string* will need to be to produce meaningful discriminations. As further discussed in [Concepts], recent studies in musical perception indicate that pitch-information without durational values does not suffice.

representational issues Given a set of musical fragments, we may envisage several reductions to arrive at the (hypothetical) prototypical melody. Such reductions must provide for the elimination of confounds such as rests, repeated notes and grace notes, and result in, for example, a pitch-string (in a suitable representation), a duration profile, and (possibly) accented note profiles and harmonic reinforcement profiles (which capture notes that are emphasized by harmonic changes). Unfortunately, as observed in [Concepts], the problem of which reductions to apply is rather elusive, since it depends to a great extent on the goals of the query and the repertory at hand.

As concerns the representation of pitch information, there is a choice between a base-7 representation, which corresponds with the position relative to the tonic in the major or minor scales, a base-12 representation, which corresponds with a division in twelve semitones as in the chromatic scale, and more elaborate encodings, which also reflect notational differences in identical notes that arise through the use of accidentals. For MIDI applications, a base-12 notation is most suitable, since the MIDI note information is given in semitone steps. In addition to relative pitch information, octave information is also important, to establish the rising and falling of melodic contour.

When we restrict ourselves to directional profiles (up, down, repeat), we may include information concerning the slope, or degree of change, the relation of the current pitch to the original pitch, possible repetitions, recurrence of pitches after intervening pitches, and possible segmentations in the melody. In addition,

however, to support relevant comparisons it seems important to have information on the rhythmic and harmonic structure as well.

similarity matching An altogether different approach at establishing melodic similarity is proposed in [Compare]. This approach has been followed in the Meldex system [Meldex], discussed in section **??**. The approach is different in that it relies on a (computer science) theory of finite sequence comparison, instead of musical considerations. The general approach is, as explained in [Compare], to search for an optimal correspondence between elements of two sequences, based on a distance metric or measure of dissimilarity, also known more informally as the *edit-distance*, which amounts to the (minimal) number of transformations that need to be applied to the first sequence in order to obtain the second one. Typical transformations include *deletion*, *insertion* and *replacement*. In the musical domain, we may also apply transformations such as *consolidation* (the replacement of several elements by one element) and *fragmentation* (which is the reverse of consolidation). The metric is even more generally applicable by associating a weight with each of the transformations. Elements of the musical sequences used in [Compare] are pitch-duration pairs, encoded in base-12 pitch information and durations as multiples of 1/16th notes.

The matching algorithm can be summarized by the following recurrence relation for the dissimilarity metric. Given two sequences $A = a_1, \ldots, a_m$ and $B = b_1, \ldots, b_n$ and $d_{ij} = d(a_i, b_j)$, we define the distance as

$$d_{ij} = min \begin{cases} d_{i-1,j} + w(a_i, 0) & \text{deletion} \\ d_{i,j-1} + w(0, b_j) & \text{insertion} \\ d_{i-1,j-1} + w(a_i, b_j) & \text{replacement} \\ d_{i-k,j-1} + w(a_{i-k+1}, \ldots, a_i, b_j).\ 2 \leqslant k \leqslant i & \text{consolidation} \\ d\_i - 1, j - k + 1 + w(a\_i, b\_j - k + 1, \ldots b\_j)\ 2 <= k <= j & \\ & \text{fragmentation} \end{cases}$$

with

$$\begin{array}{ll} d_{i0} = d_{i-1,0} + w(a_i, 0),\ i \geqslant 1 & \textit{deletion} \\ d_{0j} = d_{0,j-1} + w(0, b_i),\ j \geqslant 1 & \textit{insertion} \end{array}$$

and $d_{00} = 0$. The weigths $w(\_, \_)$ are determined by the degree of dissonance and the length of the notes involved.

The actual algorithms for determining the dissimilarity between two sequences uses dynamic programming techniques. The algorithm has been generalized to look for matching phrases, or subsequences, within a sequence. The complexity of the algorithm is $O(mn)$, provided that a limit is imposed on the number of notes involved in consolidation and fragmentation.

Nevertheless, as indicated in experiments for the Meldex database, the resulting complexity is still forbidding when large databases are involved. The Meldex system offers apart from the (approximate) dynamic programming algorithm also a state matching algorithm that is less flexible, but significantly faster. The Meldex experiments involved a database of 9400 songs, that were used to

investigate six musical search criteria: (1) exact interval and rhythm, (2) exact contour and rhythm, (3) exact interval, (4) exact contour, (5) approximate interval and rhythm, and (6) approximate contour and rhythm. Their results indicate that the number of notes needed to return a reasonable number of songs scales logarithmically with database size [Meldex]. It must be noted that the Meldex database contained a full (monophonic) transcription of the songs. An obvious solution to manage the complexity of searching over a large database would seem to be the storage of prototypical themes or melodies instead of complete songs.

indexing and analysis There are several tools available that may assist us in creating a proper index of musical information. One of these tools is the Humdrum system, which offers facilities for metric and harmonic analysis, that have proven their worth in several musicological investigations [Humdrum]. Another tool that seems to be suitable for our purposes, moreover since it uses a simple pitch-duration, or *piano-roll*, encoding of musical material, is the system for metric and harmonic analysis described in [Meter]. Their system derives a metrical structure, encoded as hierarchical levels of equally spaced beats, based on preference-rules which determine the overall likelihood of the resulting metrical structure. Harmonic analysis further results in (another level of) *chord spans* labelled with roots, which is also determined by preference rules that take into account the previously derived metrical structure. As we have observed before, metrical and harmonic analysis may be used to eliminate confounding information with regard to the 'prototypical' melodic structure.

5.2 video

Automatic content description is no doubt much harder for video than for any other media type. Given the current state of the art, it is not realistic to expect content description by feature extraction for video to be feasible. Therefore,to realize content-based search for video, we have rely on some knowledge representation schema that may adequately describe the (dynamic) properties of video fragments.

In fact, the description of video content may reflect the story-board, that after all is intended to capture both time-independent and dynamically changing properties of the objects (and persons) that play a role in the video.

In developing a suitable annotation for a particular video fragment, two questions need to be answered:

video annotation

- what are the interesting aspects?
- how do we represent this information?

Which aspects are of interest is something you have to decide for yourself. Let's see whether we can define a suitable knowledge representation scheme.

One possible knowledge representation scheme for annotating video content is proposed in [MMDBMS]. The scheme proposed has been inspired by knowledge

representation techniques in Artificial Intelligence. It captures both static and dynamic properties.

video content

video v, frame f
f has associated objects and activities
objects and activities have properties

First of all, we must be able to talk about a particular video fragment v, and frame f that occurs in it. Each frame may contain objects that play a role in some activity. Both objects and activities may have properties, that is attributes that have some value.

property

property: name = value

As we will see in the examples, properties may also be characterized using predicates.

Some properties depend on the actual frame the object is in. Other properties (for example sex and age) are not likely to change and may considered to be frame-independent.

object schema

(fd,fi) – frame-dependent and frame-independent properties

Finally, in order to identify objects we need an object identifier for each object. Summing up, for each object in a video fragment we can define an *object instance*, that characterizes both frame-independent and frame-dependent properties of the object.

object instance: (oid,os,ip)

- *object-id* – oid
- *object-schema* – os = (fd,fi)
- *set of statements* – ip: name = v and name = v IN f

Now, with a collection of object instances we can characterize the contents of an entire video fragment, by identifying the frame-dependent and frame-independent properties of the objects.

Look at the following example, borrowed from [MMDBMS] for the *Amsterdam Drugport* scenario.

frame	objects	*frame-dependent properties*
1	Jane	has(briefcase), at(path)
-	house	door(closed)
-	briefcase	
2	Jane	has(briefcase), at(door)
-	Dennis	at(door)
-	house	door(open)
-	briefcase	

In the first frame Jane is near the house, at the path that leads to the door. The door is closed. In the next frame, the door is open. Jane is at the door, holding a briefcase. Dennis is also at the door. What will happen next?

Observe thatwe are using predicates to represent the state of affairs. We do this, simply because the predicate form *has(briefcase)* looks more natural than the other form, which would be *has = briefcase*. There is no essential difference between the two forms.

Now, to complete our description we can simply list the frame-independent properties, as illustrated below.

object	*frame-independent properties*	value
Jane	age	35
	height	170cm
house	address	...
	color	brown
briefcase	color	black
	size	40 x 31

How to go from the tabular format to sets of statements that comprise the object schemas is left as an (easy) exercise for the student.

Let's go back to our *Amsterdam Drugport* scenario and see what this information might do for us, in finding possible suspects. Based on the information given in the example, we can determine that there is a person with a briefcase, and another person to which that briefcase may possibly be handed. Whether this is the case or not should be disclosed in frame 3. Now, what we are actually looking for is the possible exchange of a briefcase, which may indicate a drug transaction. So why not, following [MMDBMS], introduce another somewhat more abstract level of description that deals with *activities*.

activity

- activity name – id
- statements – *role = v*

An activity has a name, and consists further simply of a set of statements describing the *roles* that take part in the activity.

example

{ giver : Person, receiver : Person, item : Object }
giver = Jane, receiver = Dennis, object = briefcase

For example, an *exchange* activity may be characterized by identifying the *giver*, *receiver* and *object* roles. So, instead of looking for persons and objects in a video fragment, you'd better look for activities that may have taken place, by finding a matching set of objects for the particular roles of an activity. Consult [MMDBMS] if you are interested in a further formalization of these notions.

video libraries

Assuming a knowledge representation scheme as the one treated above, how can we support search over a collection of videos or video fragments in a video library.

What we are interested in may roughly be summarized as

video libraries

which videos are in the library
what constitutes the content of each video
what is the location of a particular video

Take note that all the information about the videos or video fragments must be provided as meta-information by a (human) librarian. Just imagine for a moment how laborious and painstaking this must be, and whata relief video feature extraction would be for an operation like *Amsterdam Drugport*.

To query the collection of video fragments, we need a query language with access to our knowledge representation. It must support a variety of retrieval operations, including the retrieval of segments, objects and activities, and also property-based retrievals as indicated below.

query language for video libraries

- *segment retrievals* – *exchange of briefcase*
- *object retrievals* – all people in v:[s,e]
- *activity retrieval* – all activities in v:[s,e]
- *property-based* – find all videos with object oid

[MMDBMS] lists a collection of video functions that may be used to extend SQL into what we may call VideoSQL. Abstractly, VideoSQL may be characterized by the following schema:

VideoSQL

SELECT – v:[s,e]
FROM – video:<source><V>
WHERE – term IN funcall

where *v:[s,e]* denotes the fragment of video *v*, starting at frame *s* and ending at frame *e*, and *term IN funcall* one of the video functions giving access to the information about that particular video. As an example, look at the following VideoSQL snippet:

example

```
SELECT  vid:[s,e]
FROM video:VidLib
WHERE (vid,s,e) IN VideoWithObject(Dennis) AND
      object IN ObjectsInVideo(vid,s,e) AND
      object != Dennis AND
      typeof(object) = Person
```

Notice that apart from calling video functions also constraints can be added with respect to the identity and type of the objects involved.

research directions– *presentation and context*

Let's consider an example. Suppose you have a database with (video) fragments of news and documentary items. How would you give access to that database? And, how would you present its contents? Naturally, to answer the first question, you need to provide search facilities. Now, with regard to the second question, for a small database, of say 100 items, you could present a list of videos thatb matches the query. But with a database of over 10.000 items this will become problematic, not to speak about databases with over a million of video fragments. For large databases, obviously, you need some way of visualizing the results, so that the user can quickly browse through the candidate set(s) of items.

[Video] provide an interesting account on how *interactive maps* may be used to improve search and discovery in a (digital) video library. As they explain in the abstract:

> *To improve library access, the Informedia Digital Video Library uses automatic processing to derive descriptors for video. A new extension to the video processing extracts geographic references from these descriptors.*
>
> *The operational library interface shows the geographic entities addressed in a story, highlighting the regions discussed in the video through a map display synchronized with the video display.*

So, the idea is to use geographical information (that is somehow available in the video fragments themselves) as an additional descriptor, and to use that information to enhance the presentation of a particular video. For presenting the results of a query, candidate items may be displayed as icons in a particular region on a map, so that the user can make a choice.

Obviously, having such geographical information:

> *The map can also serve as a query mechanism, allowing users to search the terabyte library for stories taking place in a selected area of interest.*

The approach to extracting descriptors for video fragments is interesting in itself. The two primary sources of information are, respectively, the spoken text and graphic text overlays (which are common in news items to emphasize particular aspects of the news, such as the area where an accident occurs). Both speech recognition and image processing are needed to extract information terms, and in addition natural language processing, to do the actual 'geocoding', that is translating this information to geographical locations related to the story in the video.

Leaving technical details aside, it will be evident that this approach works since news items may relevantly be grouped and accessed from a geographical perspective. For this type of information we may search, in other words, with three kinds of questions:

- *what* – content-related
- *when* – position on time-continuum

- *where* – geographic location

and we may, evidently, use the geographic location both as a search criterium and to enhance the presentation of query results.

mapping information spaces Now, can we generalize this approach to other type of items as well. More specifically, can we use maps or some spatial layout to display the results of a query in a meaningful way and so give better access to large databases of multimedia objects. According to [Atlas], we are very likely able to do so:

> *More recently, it has been recognized that the process of spatialization – where a spatial map-like structure is applied to data where no inherent or obvious one does exist – can provide an interpretable structure to other types of data.*

Actually, we are taking up the theme of *visualization*, again. In [Atlas] visualizations are presented that (together) may be regarded as an *atlas of cyberspace*.

atlas of cyberspace

> *We present a wide range of spatializations that have employed a variety of graphical techniques and visual metaphors so as to provide striking and powerful images that extend from two dimension 'maps' to three-dimensional immersive landscapes.*

As you may gather from chapter 7 and the *afterthoughts*, I take a personal interest in the (research) theme of *virtual reality interfaces for multimedia information systems*. But I am well aware of the difficulties involved. It is an area that is just beginning to be explored!

5.3 feature extraction

Manual content annotation is laborious, and hence costly. As a consequence, content annotation will often not be done and search access to multimedia object willnot be optimal, if it is provided for at all. An alternative to manual content annotation is (semi) automatic feature extraction, which allows for obtaining a description of a particular media object using media specific analysis techniques.

The Multimedia Database Research group at CWI has developed a framework for feature extraction to support the *Amsterdam Catalogue of Images* (ACOI). The resulting framework for feature extraction is known as the ACOI framework, [ACOI].

The ACOI framework is intended to accomodate a broad spectrum of classification schemes, manual as well as (semi) automatic, for the indexing and retrieval of arbitrary multimedia objects. What is stored are not the actual multimedia objects themselves, but structural descriptions of these objects (including their location) that may be used for retrieval.

The ACOI model is based on the assumption that indexing an arbitrary multimedia object is equivalent to deriving a grammatical structure that provides

a namespace to reason about the object and to access its components. However there is an important difference with ordinary parsing in that the lexical and grammatical items corresponding to the components of the multimedia object must be created dynamically by inspecting the actual object. Moreover, in general, there is not a fixed sequence of lexicals as in the case of natural or formal languages. To allow for the dynamic creation of lexical and grammatical items the ACOI framework supports both *black-box* and *white-box* (feature) detectors. Black-box detectors are algorithms, usually developed by a specialist in the media domain, that extract properties from the media object by some form of analysis. White-box detectors, on the other hand, are created by defining logical or mathematical expressions over the grammar itself. Here we will focus on black-box detectors only.

The information obtained from parsing a multimedia object is stored in a database. The feature grammar and its associated detector further result in updating the data schemas stored in the database.

formal specification Formally, a feature grammar G may be defined as $G = (V, T, P, S)$, where V is a collection of variables or non-terminals, T a collection of terminals, P a collection of productions of the form $V \rightarrow (V \cup T)$ and S a start symbol. A token sequence ts belongs to the language $L(G)$ if $S \xrightarrow{*} ts$. Sentential token sequences, those belonging to $L(G)$ or its sublanguages $L(G_v) = (V_v, T_v, P_v, v)$ for $v \in (T \cup V)$, correspond to a complex object C_v, which is the object corresponding to the parse tree for v. The parse tree defines a hierarchical structure that may be used to access and manipulate the components of the multimedia object subjected to the detector. See [Features] for further details.

anatomy of a feature detector

As an example of a feature detector, we will look at a simple feature detector for (MIDI encoded) musical data. A special feature of this particular detector,that I developed while being a guest at CWI, is that it uses an intermediate representation in a logic programming language (Prolog) to facilitate reasoning about features.

The hierarchical information structure that we consider is defined in the grammar below. It contains only a limited number of basic properties and must be extended with information along the lines of some musical ontology, see [AI].

feature grammar

```
detector song; # # to get the filename
detector lyrics; # # extracts lyrics
detector melody; # # extracts melody
detector check;  # # to walk the tree

atom str name;
atom str text;
atom str note;
```

```
midi: song;

song: file lyrics melody check;

file: name;

lyrics: text*;
melody: note*;
```

The start symbol is a *song*. The detector that is associated with *song* reads in a MIDI file. The musical information contained in the MIDI file is then stored as a collection of Prolog facts. This translation is very direct. In effect the MIDI file header information is stored, and events are recorded as facts, as illustrated below for a *note_on* and *note_off* event.

```
event('twinkle',2,time=384, note_on:[chan=2,pitch=72,vol=111]).
event('twinkle',2,time=768, note_off:[chan=2,pitch=72,vol=100]).
```

After translating the MIDI file into a Prolog format, the other detectors will be invoked, that is the *composer*, *lyrics* and *melody* detector, to extract the information related to these properties.

To extract relevant fragments of the melody we use the melody detector, of which a partial listing is given below.

melody detector

```
int melodyDetector(tree *pt, list *tks ){
char buf[1024]; char* _result;
void* q = _query;
int idq = 0;

  idq = query_eval(q,"X:melody(X)");
  while ((_result = query_result(q,idq)) ) {
        putAtom(tks,"note",_result);
        }
  return SUCCESS;
}
```

The embedded logic component is given the query `X:melody(X)`, which results in the notes that constitute the (relevant fragment of the) melody. These notes are then added to the tokenstream. A similar detector is available for the lyrics.

Parsing a given MIDI file, for example *twinkle.mid*, results in updating the database.

implementation The embedded logic component is part of the *hush* framework, [OO]. It uses an object extension of Prolog that allows for the definition

of native objects to interface with the MIDI processing software written in C++. The logic component allows for the definition of arbitrary predicates to extract the musical information, such as the melody and the lyrics. It also allows for further analysis of these features to check for, for example, particular patterns in the melody.

questions

content annotation

1. (*) How can video information be made accessible? Discuss the requirements for supporting video queries.

concepts

2. What are the ingredients of an *audio data model*
3. What information must be stored to enable search for video content?
4. What is *feature extraction*? Indicate how feature extraction can be deployed for arbitrary media formats.

technology

5. What are the parameters for *signal-based (audio) content*?
6. Give an example of the representation of *frame-dependent* en *frame-independent* properties of a video fragment.
7. What are the elements of a query language for searching in video libraries?
8. Give an example (with explanation) of the use of *VideoSQL*.