ARGOS: Automatically extracting Repeating Objects from multimedia Streams

Cormac Herley
Microsoft Research
c.herley@ieee.org
January 2004
Technical Report
MSR-TR-2004-02

Many media streams consist of distinct objects that repeat. For example broadcast television and radio signals contain advertisements, call sign jingles, songs and even whole programs that repeat. The problem we address is to explicitly identify the underlying structure in repetitive streams and de-construct them into their component objects. Our architecture assumes no \textit{a priori} knowledge of the streams, and does not require that the repeating objects be known. Everything the system needs, including the position and duration of the repeating objects, is learned on the fly. We demonstrate that it is perfectly feasible to identify in realtime repeating objects that occur days or even weeks apart in audio or video streams. Both the compute and buffering requirements are comfortably within reach for a basic desktop computer. We outline the algorithms, enumerate several applications and present results from real broadcast streams.

Microsoft Research
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052

http://www.research.microsoft.com
Index Terms
Multimedia, repeats, low dimension representation, segmentation, audio fingerprint

I. INTRODUCTION

Rather than having infinite innovation many media streams are quite repetitive. They contain objects that recur with essentially no change. Advertisements on broadcast radio and television stations, and songs or video on music channels are some of the obvious examples. Other examples are station call-signs and jingles, signature tunes of particular television programs, news footage of noteworthy events (e.g. a clip from a State of the Union speech in the US will typically air many times in the days following the speech) and even entire radio or television programs (e.g. many NPR member stations broadcast syndicated programs more than once). These objects can vary from a few seconds in length to several hours. The gaps between successive copies of the same object can be as little as a few minutes or as great as weeks or even months. The objects may repeat in a very regular fashion (e.g. the signature tune of a news program occurring at defined times throughout the day) or with no apparent regularity (e.g. particular songs occurring in a radio broadcast at the discretion of the disk jockeys or playlist generators). Some objects may be repeated very frequently for a short period of time and then are seldom seen again (e.g. a clip of a current newsworthy event) others may be repeated over the spans of years (e.g. signature tunes of serial programs). Since there is such diversity in the length of repeating objects (ROs), and the frequency and regularity of their recurrence few useful generalizations about them can be drawn.

Much work has been reported on identifying, segmenting and classifying particular classes of objects. For example in applications such as television news summarization progress has been made in distinguishing between news-anchor scenes, live report scenes, weather reports and so on [13], [8]. The Scene Transition Graph developed by Yeung et al [16] assists with content-based browsing. Distinct scenes are identified by detecting shot boundaries and building a graph based on scene similarities. In [16] related scenes are clustered (e.g. closeups of the same person in a dialog scene) while our work is interested only in identifying segments of streams that are identical (other than channel deformations). Many of the approaches to stream analysis make use of the growing body of work on audio analysis, querying and retrieval. See for example [15], [12].

An interesting approach by Lienhart et al [10] is to detect a known set of commercials. That is, given a library of, say, $K$ commercials (i.e. short sequences of frames that can be expected to recur in a stream) they calculate a fingerprint, which is a function of this sequence of frames. This fingerprint, which they base on the Color Coherence Vector [11], can be compared to detect recurrence of the already known commercials. However, the authors of [10] point out that any fingerprint which is tolerant of channel deformations, has low dimension and discriminates well between different commercials will suffice. They also point out that the arrival of new, previously unknown commercials may be inferred, when, for example, an unknown 30s sequence of frames occurs between two known commercials. Like [10] we will examine possible candidates for fingerprints. Unlike [10] however, we will not assume that a library of known Repeating Objects is available or make any assumptions about the duration or
nature of new RO’s as they appear in the stream. Another interesting approach along these lines is [9], which
explores an efficient method to search for known objects in long streams or libraries. Both [10] and [9] have in
common with our work that they are explicitly interested in very long sequences (i.e. days of weeks); the point of
difference is that they take the set of sought objects to be known.

By contrast we propose explicit detection of repeating segments in the stream without making assumptions of
the nature of the objects. For example, we will not need to assume that any of the objects we seek have different
audio or video characteristics from the non-repeating portions of the stream, and we will only assume that RO’s
have some minimum length (e.g. they are at least 30s in length). We will not need to assume the existence of a
library of already labelled or recognized objects, but rather will learn the repeat patterns on the fly. An interesting
example of searching for unknown RO’s in a music stream is [5] though the authors there work on MIDI data
rather than a raw audio format. The literature on text string searching has numerous approaches [2], [14] that can
serve as analogies for our work

In the next section we show how repeats may be identified in a stream using a scheme that is conceptually
simple, but requires considerable compute and memory resources. In Section III we show that using low dimension
representations of the stream we can hugely reduce the computational complexity. Further by segmenting the objects
we can form a library of objects as they are found and remove the need to buffer large amounts of the stream.
Combined these improvements greatly reduce the computation and buffering required. In Section IV we give an
architecture of the system and show the results of object detection and extraction performed on real broadcast
streams. Numerous applications are enabled by the ability to decompose a stream into its component objects, most
obviously the ability to customize a stream and make a library of objects for later viewing. We call the system
ARGOS: Automatic RepeatinG Object Segmentation. That being also the name of the dog who recognized Ulysses
on his return, even though more than twenty years had passed.

II. SEARCHING FOR UNKNOWN REPEATING OBJECTS

Suppose that our media stream is \( s(t) \), and this contains embedded repeating objects. We assume that the objects
undergo relatively minor copy-to-copy variation; i.e. successive repeats of an object in a stream separated by time
will obviously experience different channel deformations, but will otherwise be almost identical. We further assume
that objects are generally non-overlapping. We do not assume that objects are always precisely the same length
at each occurrence. For example, in the case of a newsclip: a 10s extract of a speech may be aired on some new
bulletins, but a larger 20s extract on others. Equally, a song may not always be played from the beginning to the
end; it is likely to be shortened in a different way each time it plays, and is likely to be corrupted at the beginning
and/or end by DJ voice-over.

We’ll model the stream as being made up of \( K \) repeating objects, which are played with relative frequencies
determined by the probabilities \( p_i \). That is, the stream is constructed by choosing an object from the library, and
the \( i - th \) object has independent probability \( p_i \) of being chosen when a decision is made. For generality we will
also assume that only a fraction \( r \) of the stream consists of repeating objects; for example \( r = 0.9 \) would imply
that one tenth of the stream was non-repeating content that separated repeating objects from each other. While a little simplistic, this model is sufficiently general to capture the behavior of real streams reasonably accurately, as we shall see from the experiments on real streams in Section IV.

A. Determining Repeats

First, observe that it is simple to verify that an object at \( t_i \) also occurs at \( t_j \). This can be done, for example, by taking the cross-correlation of windowed sections of the stream centered at \( t_i \) and \( t_j \). If the cross-correlation:

\[
[w(t) \cdot s(t - t_i)] \ast [w(t) \cdot s(t - t_j)]
\]

has a sufficiently large peak (compared with the autocorrelation of either section) we would determine that they are approximately the same. Figure 1 gives an example of two signal segments that are approximately equal, for at least a portion of the segment, and the cross-correlation. The peak of the cross-correlation is clearly strong enough to allow accurate identification of similarity. Thus a very simple routine for determining equality of segments is:

\[
\text{[match]} = \text{ApproxEqual}(\text{iblock}, \text{jblock})\{
\text{acs} = \text{crossCorr}(\text{iblock}, \text{iblock});
\text{xcs} = \text{crossCorr}(\text{iblock}, \text{jblock});
\text{match} = (\text{max}(\text{xcs}) > 0.5*\text{max}(\text{acs})) \ ? \text{true} : \text{false};
\}
\]

This is only the simplest example: the Color Coherence Vector used (for video) in [10], [11] the fingerprint algorithms used in [1], [6], the audio measures of [15], [12] and the similarity measures of [9] are among other functions that could be used.

B. Searching for Unknown Repeating Objects

If we sought only a single known object it would be an easy matter to spot recurrences: we would just constantly compare the known object with every incoming block. Seeking \( K \) known objects (much as is done in [10] and [9]) would require comparing each incoming block against the \( K \) known objects. This would require \( K/2 \) comparisons per incoming block on average. Thus the complexity of determining whether known objects are present in a stream is directly related to the size of the database of sought objects. Also observe that when the objects are known, a buffer of the known objects is required but no buffer of the stream is needed.

The case where the objects are unknown is more complex, since we must learn first what the objects are, and then find all of their recurrences. A starting point to this problem is the extensive literature on finding patterns in text strings and files [2] and finding distinct entries in a database [3], [4]. A first approach, is to break a finite stream into \( N \) blocks and compare the current block with every other:

\[
\text{[found, jposn]} = \text{searchBuffer(curPosn)}(\text{curPosn};
\text{while (jposn} > \text{curPosn} - \text{NL})} \]
if (ApproxEqual(stream(iposn), stream(jposn))
    AddToRepeatsList(iposn, jposn);
    found = true;
    break;
end
jposn -= L;
end }

The blocks are spaced $L$ apart. We assume that our ApproxEqual() routine determines whenever two blocks contain
the same object with almost no false positives or negatives (we’ll examine these assumptions and how long $L$ needs
to be in Sections III-A and III-B). We assume that AddToRepeatsList() performs whatever actions are desired when
the fact that an object occurs at both iposn and jposn has been verified.

The complexity of the algorithm is determined by the $N - 1$ calls to ApproxEqual(). So clearly, a buffer of
duration $NL$ can be searched in realtime only if $N - 1$ calls to ApproxEqual() can be computed in time $L$. To
consider a concrete example, suppose we have a finite amount of computing power available which is capable of
performing 10 calls to ApproxEqual() per second, and our blocks are $L = 90s$ apart. The longest buffer we could
search would then be approximately 22 hours in duration.

1) Analysis of probability of objects repeating in the buffer: For an object to be found by searchBuffer() it must
appear twice in the buffer. Assume objects have average duration $E$. Then the average time between repeat copies
of object $i$ will be $E/(rpi)$. Suppose the length of our buffer is $B$. So all objects for which $E/(rpi) < B$ have a
good chance of occurring twice in the buffer. We now show that the probability that two copies of an object appear
in the buffer increases significantly as the buffer evolves.

At any given time there will be $M = \lfloor B \cdot r/E \rfloor$ repeating objects in the buffer. Call $Pr(i, t, found)$ the probability
that two copies of object $i$ have been in the buffer simultaneously at least once by time $t$ (and thus the object has
been found by routine searchBuffer()). Clearly, at $t = B$, when the buffer first fills,

$$Pr(i, B, found) = \sum_{j=2}^{M} \binom{M}{j} p_i^j \cdot (1 - p_i)^{M-j}.$$ 

We now wish to know how $Pr(i, t, found)$ evolves for increasing $t$. Since this is a cumulative probability it can
only increase. Recall that in time $E/r$ one new object enters the buffer, and another drops out. The increase brought
about is:

$$Pr(i, t + E/r, found) - Pr(i, t, found) = [1 - Pr(i, t, found)] \cdot Prob\{1 \text{ copy of i in buffer}|Not 2 \text{ copies of i}\} \cdot \frac{M! \cdot p_i(1 - p_i)^{M-1}}{(M-1)!p_i(1 - p_i)^{M-2} + (1 - p_i)^M} \cdot p_i \cdot \frac{M-1}{M}.$$ 

In words, in the interval $E/r$ it takes a new object to enter the buffer, the probability that object $i$ has appeared at
least twice increases by the probability that: the buffer previously contained only one copy, and the incoming object
is the $i\text{-th}$, and the outgoing object is not the $i\text{-th}$. In Figure 2 we graph how $P_r(i, t, \text{found})$ evolves for various scenarios. Using $B = 1440$ minutes (i.e. one day) and $r = 0.9$ and $E = 3.5$ minutes we show $P_r(i, t, \text{found})$ for $p_i = 0.0081, 0.0027$ and $0.0009$. Objects with these probabilities would have average repeat intervals $E/(rp_i)$ of 8 hours, 1 day an 3 days respectively. As can be seen from Figure 2, even objects that occur on average with intervals larger than the searched buffer have high likelihood of being detected if we wait long enough.

C. Summary

We saw in Section II-A that determining that two segments of a media stream are the same is simple. Applying this approach by brute force to every segment allows us to find all the objects that appear more than once in a buffer. For very common objects it is clear that they are likely to appear twice or more in a sufficiently long buffer. In Section II-B.1 we saw that for less common objects time is on our side also: they have increasing probability of appearing at least twice as the buffer evolves.

However this approach is impractical for a number of reasons:

- Calculating cross-correlations of audio or video segments is expensive
- The longest buffer it is feasible to search in realtime may not be long enough to contain many repeating objects
- Long buffers of media streams consume much memory (one day of uncompressed 44.1kHz stereo audio consuming 15.2 Gbytes).

The main approaches to improve the efficiency of this algorithm are:

1) Reduce the complexity of each call to ApproxEqual()
2) Reduce the number of calls to ApproxEqual().

The sorting and searching literature [2] will be of some assistance with 2., and we’ll examine this in Section III-D. Some techniques that are particular to media can help with 1., and we’ll examine them in Sections III-B and III-A. We will also show that we can eliminate the need to buffer long stretches of the full rate stream.

III. REDUCING THE COMPLEXITY OF DETERMINING REPEATS

Clearly, as we’ve just seen, determining that two segments of a stream are the same can be done by computing their cross-correlation. However, calculating cross-correlations on video data, or even audio at 44.1k samples per second is computationally expensive. Both video and audio contain much redundancy that does not help decide similarity. Here we can borrow an insight from the database literature: to determine that two objects are the same we do not need to compare them directly, it suffices to compare their hashes [3], [4]. The advantage of comparing the hashes is that the hash will typically be a very short string (128 bits or so) and so there is considerable computational saving. The probability of different records having the same hash is negligible. To follow this lead, we now seek a media equivalent of such a hash: it should have far lower dimension than the media segment it represents, it should be robust to deformations that are likely on the channel but there should be negligible probability that two different segments will produce the same hash.
A. Low Dimension Representations of Audio

Since media streams primarily consist of audio alone or video accompanied by audio, we restrict attention to low dimension representations for audio objects. For an RO to occur in a video stream it is necessary, though not sufficient that an RO occur in the associated audio component of the stream. Hence we’ll concentrate on audio streams; this spares involving the video in search process.

A commonly used tool for audio analysis is to split the audio into critical bands [7] (sometimes known as Bark bands). For example a commonly used set for audio splits the signal into 25 bands with band centers at \{100, 200, 300, 400, 510, 630, 770, 920, 1080, 1270, 1480, 1720, 2000, 2320, 2700, 3150, 3700, 4400, 5300, 6400, 7700, 9500, 12000, 15500, 22200\} Hz. These are narrow frequency selective channels, and can of course be sampled at a much lower frequency than the overall audio signal. In fact, we take the energy of the signal in the bark band centered at 770 Hz, lowpass filter it, and then sample at 11 samples/sec. We call this waveform BB7. This of course is no longer sufficient to give even a crude representation of the signal; however it suffices for recognition.

By way of example in Figure 4 we show two sections of the energy of BB7. Each segment represents slightly greater than 10 minutes of audio. They are different copies of the same object recorded from an FM radio station at different times. As can be seen in the center portion both copies are approximately equal, while at the beginning and end they differ.

1) Experimental: To test the hypothesis that BB7 forms a good low dimension representation of audio we performed the following experiment. We recorded an FM radio station for a period of seven days, and calculated BB7 for the entire stream. We randomly selected 500 6 minute segments; some of the segments were voice, some music, some a mixture of voice and music. We then formed the cross-correlation of each segment with every 6 minute segment from the entire stream. When the cross-correlation indicated a match we examined the two segments manually to determine whether or not they actually matched. Some of the objects had no matches (other than the segment of their own occurrence in the stream) and others occurred as many as 20 times. In no case was a match indicated where data did not correspond to the same object. This suffices to indicate that the false positive rate of using cross-correlations of BB7 is very low.

Determining the false negative rate is harder. To accomplish this we hand parsed a 48 hour section of the stream; i.e. hand labeled every repeating object greater than 2 minutes in length. We randomly selected 50 objects from the first 24 hours of labeled stream, and for each of them: cross-correlated a 6 minute segment of BB7 centered on the object with the entire second 24 hours of labeled stream. No false negatives were found in the labeled stream.

Thus, we find that BB7 is a suitable low dimension representation of the signal. In going from 44.1 k samples/sec to 11 samples/sec we have achieved a 4000-fold reduction in the data rate, with no meaningful loss in recognition ability. We should emphasize that we no claims of optimality for BB7. In fact we point out many other low dimension representations that have been explored [9], [15], [12], [10] and several of these might serve as well or better. We choose BB7 merely as an example that works well.

2) Choosing a block length: If we break the buffer into blocks, the length of the blocks should be determined by length of the shortest RO we are interested in. For example, if an RO is 10s long: two instances each of which
sit in a block of length 200s are unlikely to generate discernible peak in the cross-correlation function, while they probably would if we chose 20s. A basic analysis of how the peak of the cross-correlation of two blocks falls off (relative to the auto-correlation of either one) reveals that we should choose the block length to be on the order of twice the length of the shortest RO of interest.

A related question is how much overlap, \( L \) there should be between blocks. Again the shortest RO of interest determines the answer. An overlap of \( L \) for blocks of length \( 2L \) guarantees that for at least one of the calls to \texttt{approxEqual()} any object of length \( L \) will be entirely contained in both blocks (and thus will be detected).

B. Audio fingerprints

Others have examined the question of producing audio fingerprints, which efficiently determine whether two sections of audio are the same, and yet are robust to commonly encountered deformations. In particular the fingerprints described by Burges \textit{et al} in [1], and by Haitsma and Kalker in [6] are extremely powerful. Both of these schemes seem capable of scaling to large databases. For example Burges \textit{et al} report searching a database of 240000 objects in real-time every 0.186s using only 5% of CPU on a 1.2 GHz Pentium III PC.

Fingerprints can be thought of as snapshots of a particular segment of audio, much of the blocks of BB7 that we use are. An approach to detecting RO’s in a stream using fingerprints, that is similar to \texttt{searchBuffer()} is shown in Figure 3. One thread calculates fingerprints and enters them in a database, a second thread calculates fingerprints and checks the database. When a newly calculated fingerprint matches an old one an RO has been found. The fingerprint algorithms of [1] and [6] are at least as powerful as our approach. However, having a continuous representation of the stream (such as BB7) rather than a series of fingerprints is important as we show next.

C. Identifying the boundaries of found objects

We have seen that repeating objects can be identified in a number of ways. Cross-correlations (such as in Figure 1), Audio fingerprints [1], [6] or Color Coherence Vectors [11], [10] (in the case of video) are among the approaches that work. Any of these methods will determine that the data in the stream at locations \( t_i \) and \( t_j \) are approximately similar. Of course to carry out any action on an RO, \textit{e.g.} delete it from the stream, copy it to disk and so on, we need to know its precise endpoints. We need to know, for example, whether it is 10 seconds or 3 minutes long before any action can be carried out. Here we reach an important distinction between systems where the RO’s are drawn from a library of known objects and those where the objects are unknown, which is the case we are interested in:

- RO’s from finite known library: endpoints determined from metadata (\textit{e.g.} the library contains length information for all RO’s) or inferred (\textit{e.g.} commercials are known to be 30s or 15s long)
- RO’s not known: endpoints must be determined from the stream.

Determining the endpoints becomes simple however if we have access to the stream. Conceptually, if we align the two portions of the waveform we can trace backward toward the beginning and forward toward the end to determine where the two copies diverge, giving the boundaries of the object. Recall, in Section III-A, we found
that using the full rate stream was unnecessary to determine when matches occurred. In fact, in Figure 4 we saw that BB7 showed considerable visual similarity between two different occurrences of the same object. In Figure 5 we align, normalize and overlay these two segments. As can be seen there is substantial (though not precise) overlap between the two occurrences. This allows us to state with reasonable accuracy that the segments coincide approximately between samples 5800 and 8300 and thus (since BB7 is sampled at 11 samples/sec) the object is approximately 227 seconds in duration.

The alignment and normalization are crucial enough that is worth specifying how they are accomplished. Assuming that we have determined that the streams at \( t_0 \) and \( t_1 \) are approximately the same, we examine a windowed section of the stream centered at each of these locations:

\[
z_0(t) = [w(t) \cdot s(t - t_0)]
\]

\[
z_1(t) = [w(t) \cdot s(t - t_1)].
\]

The window \( w(t) \) has length sufficient to ensure that the entire RO is included. Hence it should have length greater than twice the longest RO of interest. We can align the two sections of stream by calculating

\[
\tau = \tau_0 - \tau_1
\]

where \( \tau_0 \) and \( \tau_1 \) are the locations at which \( R_{y0y0}(t) \) and \( R_{y0y1}(t) \) are maximum, i.e. the peaks of the auto and cross-correlations respectively. We set

\[
y_0(t) = [w(t) \cdot s(t - t_0 + \tau/2)]
\]

and

\[
y_1(t) = [w(t) \cdot s(t - t_1 - \tau/2)].
\]

Denote the low dimension representation of the RO by \( x(t) \). Both \( y_0(t) \) and \( y_1(t) \) should now have the form

\[
y_0(t) = [l_0(t) \ f_0(x(t)) \ r_0(t)]
\]

\[
y_1(t) = [l_1(t) \ f_1(x(t)) \ r_1(t)],
\]

where \( l_i(t) \) and \( r_i(t) \) are the parts of the stream that precede and follow \( x(t) \) at the two locations in the stream where we have determined it repeats, and \( f_0() \) and \( f_1() \) are functions that denote any deformation suffered by \( x(t) \) in the channel. By definition \( l_0(t) \) and \( l_1(t) \) have the same length, as do \( r_0(t) \) and \( r_1(t) \) (since we have aligned the two segments). Since the two repeats are recorded from the same stream, but potentially separated by a large amount of time, it is not safe to assume that \( f_0 = f_1 \). While many factors possible contribute to the channel deformations, in our experience differences due to reception conditions for terrestrial broadcast seem adequately captured by a simple affine model:

\[
f_i(x(t)) = \alpha_i \cdot x(t) + \beta_i.
\]

This might be too simple to capture the channel differences experienced by two full rate copies, however in the low dimensional representation it seems to adequately capture the differences.
Define an expectation operator

\[ E\{x(t)\} = \frac{1}{b-a} \int_a^b x(t)dt. \]

We can estimate \( \beta_i \approx E\{y_i(t)\} \) and \( \alpha_i \approx E\{|y_i(t) - \beta_i|\} \), so long as the interval \((a, b)\) is a subset of the interval on which \( f_i(x(t)) \) is supported. That is, so long as \( a \) is to the right of the boundary between \( l_i(t) \) and \( f_i(x(t)) \) and \( b \) is to the left of the boundary between \( r_i(t) \) and \( f_i(x(t)) \) our estimates of \( \alpha_i \) and \( \beta_i \) will be accurate. In other words, in normalizing, we wish to estimate the gain \( \alpha_i \) and offset \( \beta_i \) over intervals where our two stretches of stream coincide, without including portions of the preceding and following stretches of the stream. This may seem like a small matter, but it introduces a difficulty when no \textit{a priori} expectation of the object length is available. In practice, if we are interested, for example, only in objects of length 3 minutes or greater we can take \( b-a \) to be one minute or so; this is long enough to make the expectation operator accurate, while being short enough to avoid the risk of including data from neighboring objects.

Once the gain and offsets have been estimated we form

\[ y_0''(t) = (y_0'(t) - \beta_0) = [l_0(t) - \beta_0 \quad \alpha_0 x(t) \quad r_0(t) - \beta_0] \]

and

\[ y_1''(t) = (y_1'(t) - \beta_1) \cdot \alpha_0 / \alpha_1 = \left[ \frac{\alpha_0}{\alpha_1} l_1(t) - \beta_1 \quad \alpha_0 x(t) \quad \frac{\alpha_0}{\alpha_1} r_1(t) - \beta_1 \right]. \]

These two stream segments can now be compared directly. This has been done in Figure 5. The boundaries of the object \( x(t) \) can be estimated by, for example, thresholding the accumulated difference between the waveforms as one works out from the center. Once the normalization has been properly performed many schemes work equally well, so we omit the details. In keeping with our pseudo-code approach we will define a routine

\[
[i0,i1,j0,j1] = \text{getBoundaries}(iposn,jposn)\{
[y0pp, y1pp] = \text{normalizeAndAllign}(iposn,jposn);
i = \text{length}(y0pp)/2;
\text{while} \ (\text{diff}(y0pp(i),y1pp(i)) < \text{threshold}) \{
\quad i-- ;
\}
i0 = iposn + i;
j0 = jposn + i;
i = \text{length}(y0pp)/2;
\text{while} \ (\text{diff}(y0pp(i),y1pp(i)) < \text{threshold}) \{
\quad i++;
\}
i1 = iposn + i;
j1 = jposn + i;
\}
\]

This performs the alignment and normalization of the segments at iposn and jposn and returns the boundaries of both copies.
1) Experimental: To test the accuracy of our segmentation we randomly selected 100 of the repeating objects from the hand-labelled 48 hour stream referred to in Section III-A.1. We exhaustively searched for all matches in the following 48 hours. Of the 100 objects, 88 had matches, and some of them multiple matches. We first manually examined the stream and decided where the endpoints were\(^1\) and then calculated the end points using our tracing algorithm. In most cases the calculated endpoints corresponded accurately with those determined by hand. In only 6 of the 88 cases was the difference greater than 2s.

Thus, the endpoint determination procedure based on examining BB7 data alone allows reasonably accurate segmentation of objects. This implies that archiving large stretches of the original media stream is not necessary to accurately find and extract objects. We saw, in Section III-A, that whether two segments of a stream matched could be determined by examining a low dimension version such as BB7. Now we see that segmenting objects can also be accomplished using a low dimension version.

Simple though it is, the endpoint determination is key to our ability to determine the underlying structure of a media stream. We already saw, in Sections III-A and III-B, that finding repeating objects was possible in real time. Determining the endpoints makes it possible to reliably determine the duration without needing a pre-existing database, or needing to make assumptions of the nature of the objects, or the semantics of the stream. Thus, for the first time, we can decompose a repetitive stream into it’s component objects. Further, the fact that the endpoint detection can be done reliably on a low dimension version of the signal means that we do not have to buffer large amounts of full rate data. For example, an object that recurs only twice in a week long video stream can be identified and accurately segmented based on a buffer of BB7, which consumes only 26 MBytes. This hugely simplifies the requirements of the system we will use to extract the underlying structure in our media streams; the architecture of such a system will be presented in Section IV.

D. Increasing the efficiency of the search for Repeating Objects

The search strategy we introduced in Section II-B was essentially brute force: break a buffer of the stream into blocks and compare the current block with all past blocks in the buffer. We’ve seen (in Sections III-A and III-B) how to improve the efficiency of the comparisons, and (in Section III-C) how to extract the objects once found. Recall, from Section II-B, that for fixed computational resources our algorithm could search a finite distance into the past. The complexity clearly grows linearly with the length of the buffer we wish to search. We now show that by exploiting the repetitive nature of the stream we can improve the search.

The key observation is that once an object is found, and it’s endpoints identified, that segment of the buffer does not have to be searched again, and the object can be added to a list of known objects. Every time we find a repeating object, we can shorten the length of the buffer that remains to be searched or, extend the distance into the past that we can search. In fact, once we find a repeating object, we add it to a library of objects. Subsequently,

\(^1\)Note: the endpoint is regarded as the point where the two copies of the object diverge. For example, a 10s news clip which is itself part of a larger 20s clip will count as a repeating object. The repeating object will be considered to be 10s long however.
we search this library first, and search the remainder of the buffer only if we find no match in the library. The advantage is that after its second appearance each repeating object will be in the library and will be identified from there without having to search the buffer. So long as the library is smaller than the buffer this improves the search, and for repetitive streams this is the case. In addition the library of found repeating objects can be ordered by frequency of repetition, so that most common object are checked first; this further improves the efficiency.

This improves the efficiency of the search by an amount related to how repetitive the stream is. The most common objects are found first, and also reduce the remaining length most. This gives rise to the following simple variation on our first algorithm.

```plaintext
[found] = searchLibraryAndBuffer(curPosn)
for k = 0 to objectsFound{
    if (approxEqual(block(curPosn), Library(k)){
        [i0,i1,j0,j1] = getBoundaries(block(curPosn), Library(k));
        shuffleUp(i0,i1,curPosn-NL);
        foundInLibrary = true;
        break;
    }
} if (not(foundInLibrary)){
    [foundInBuffer, jposn] = searchBuffer(curPosn);
    if (foundInBuffer){
        [i0,i1,j0,j1] = getBoundaries(block(curPosn), block(jposn));
        objectsFound++;
        addToLibrary(curPosn, i0, i1);
        shuffleUp(i0,i1,j0);
        shuffleUp(j0-(i1-i0),j1-(i1-i0),curPosn-NL);
    }
}
```

The major differences are that once an object is found we add it to the library, remove both copies from the buffer and "shuffle up" the rest of the buffer by an the appropriate amount. The shuffleUp() routine merely deletes the found object and moves the buffered data to fill the gap:

```plaintext
shuffleUp(i0,i1,edge){ for ix = i1 to edge
    LD(ix) = LD(ix - (i1-i0));
}
```
Here we assume that LD is a stream that contains the low dimension data. Obviously it is possible to delete objects from the buffer without shuffling the data. A system of pointers allows easy “deletion” without moving any data. We present shuffleUp() in this fashion only for clarity.

We now analyze the improvement of this algorithm over the brute force approach in Section II-B. If the current block contains the $i^{th}$ object the number of calls to approxEqual() will on average be $K/2$ if that object is already in the library and it will be less than $K + (N - 1)$ otherwise. Actually, if we sort the library of objects by frequency of occurrence and search it in that order, the number of calls will on average be less. The average number of objects in the library is $\sum_{i=0}^{K-1} Pr(i, t, \text{found})$, so it should take $\sum_{i=0}^{K-1} Pr(i, t, \text{found}) \cdot i$ calls to approxEqual() on average to find the $i^{th}$ object in the library (assuming that the $p_i$ are sorted in descending order). Thus, the average number of calls to approxEqual() is less than

$$Pr(i, t, \text{found}) \cdot \left( \frac{\sum_{i=0}^{K-1} Pr(i, t, \text{found})}{K} \cdot i \right) + \left[ 1 - Pr(i, t, \text{found}) \right] \cdot \left( \sum_{i=0}^{K-1} Pr(i, t, \text{found}) + N - 1 \right).$$

Thus the average number of comparisons will be upper bounded by

$$\sum_{i=0}^{K-1} p_i \cdot \left( Pr(i, t, \text{found}) \cdot \left( \frac{\sum_{i=0}^{K-1} Pr(i, t, \text{found})}{K} \cdot i \right) + \left[ 1 - Pr(i, t, \text{found}) \right] \cdot \left( \sum_{i=0}^{K-1} Pr(i, t, \text{found}) + N - 1 \right) \right) .$$

As shown on Figure 2 $Pr(i, t, \text{found})$ tends to one for increasing time; hence, the average number of comparisons tends to decrease to the number required to search the library. That is, as the RO library fills most objects are found from the library and the necessity of searching the buffer becomes rarer and rarer. Recall from Section II that the complexity of searching for known objects was linear in the size of the library $K$, while searching for unknown objects was linear in the size of the buffer $N$. Now we have shown that, using searchLibraryAndBuffer(), the complexity is initially proportional to $N$, but quickly converges to the cost of searching the library. Thus, for repetitive streams, the complexity of identifying unknown repeating objects converges to the complexity of identifying a collection of known repeating objects.

We take the example of stream with composed of $K = 100$ objects drawn from a uniform distribution and $K = 500$ from a Zipf distribution. This means that the n-th most common object occurs with a frequency inversely proportional to n (a distribution that occurs naturally in a wide variety of contexts) [2]. We choose $r = 0.9$ and the average object is of length $E = 210 s$, and a buffer of length $B = 24 \times 60 \times 60 s$. In Figure 6 (a) we show how the average number of calls to approxEqual() as predicted by Equation (1) evolves over time. As can be seen the number is high as the entire buffer has to be searched. It drops as the RO library fills however, and approaches the size of the library when almost all objects are found from the library and the buffer seldom needs to be searched. In the case of the uniform distribution the number of calls approaches $K/2$ since this is the average required to search the library once all objects have been found. In Figure 6 (b) we show the fraction of the $K$ objects that comprise the stream that have been found as a function of time. In this example, using a buffer of length one day, most objects have been found within two days.
E. Summary

In Section II we saw that searching for unknown repeating objects in a stream was possible but demanding because of

- Complexity of each call to approxEqual()
- Number of calls to approxEqual() required to search a buffer for unknown objects
- Memory required to buffer long stretches of a media stream.

We have now solved each of these problems, making the identification of repeating objects not merely possible but within reach with modest compute and storage resources.

IV. Architecture of a System to Allow Automatic Identification and Segmentation of Repeating Objects

Having abstracted the blocks that

- Determine whether two segments are the same object (Sections III-A and III-B)
- Identify the endpoints of the object (Section III-C)

we are now in a position to illustrate the architecture of a scheme for identifying and extracting repeats.

A block diagram of the architecture is shown in Figure 7. An incoming media stream is fed into a full rate buffer before playback, so that the incoming stream is delayed before viewing. At the same time a low dimension version of the stream is calculated and fed into a second (low dimension) buffer. The first buffer is full rate, meaning that it contains the full fidelity stream, e.g. 44.1 kHz stereo for an audio stream, or 50 frames per second for video. This buffer can be quite short as we will use it only for performing operations on objects when they have been identified and before they are played. The second buffer, containing the low dimensional data, can represent a far longer stretch of the stream. This buffer will be used to determine when a segment in the stream is recurring (as covered in Sections III-A and III-B) and to identify the endpoints of the objects once found (as covered in Section III-C). That is, once two copies of an RO have been identified in the low dimension buffer using searchLibraryAndBuffer() the boundaries are identified using getBoundaries(). One of the copies will have just entered the buffer, the other may be some distance in the past. Thus, the most recent copy will also be sitting in the full rate buffer. We use the boundaries to delete, copy or perform other action on it.

Again, to use the concrete example of 16 bit stereo audio, a full rate 20 minute buffer would consume $20 \cdot 60 \cdot 44.1e3 \cdot 2 \cdot 2 \approx 211$ Mbytes if the audio were uncompressed. Using BB7 for the low dimensional representation, a week long low rate buffer would consume only 26 Mbytes. Thus neither of these presents any real burden for a modestly capable desktop computer.

A. Binding Actions to Objects

The architecture of Figure 7 allows identification of objects as they repeat, and determination of their endpoints. Thus before an object is played in the high rate buffer, we will know whether it is a repeating object, and will have
identified its endpoints if it is. By providing a suitable User Interface (UI) we can allow users to specify actions that they wish to associate with objects as they recur. For example:

- Keep a copy of this object
- Delete all future instances of this object.

Keeping a copy is easily accomplished: using the endpoint information derived from the low dimension buffer we determine the boundary points of the object in the full rate buffer. Copying merely involves storing the appropriate section of the full rate buffer to a file. Deleting involves advancing the read pointer of the buffer to the rightmost boundary of the object playing. An example of deletion and insertion on the full rate buffer is shown in Figure 8.

Of course, every time an object is deleted the distance between the read and write pointers of the buffer decreases. Deleting many objects will bring the read and write edges of the buffer close together. The edges can be pushed further apart by inserting objects in the buffer. These can be taken from a library or from objects identified and copied earlier in the stream. For example a stream is made up of objects some of which the user elects to skip; as he does so the buffer starts to get smaller, and hence the ability to continue deleting objects will be reduced if no action is taken. To prevent this objects that the user has seen and not deleted are inserted to prevent the read and write edges of the buffer from getting too close.

B. Example of identifying Repeating Objects on Broadcast Streams

We implemented the algorithm above and ran on several broadcast streams. Using a Pentium III 750 MHz PC with 512 MBytes of RAM we were able to search a buffer of 3 days in real-time using only approximately 20% of CPU. An FM receiver was connected to the line-in of the PC. Our low dimension representation was used to find repeats and detect endpoints of objects once found.

Examples of the results listening to two different FM Radio stations are given in Figures 9 and 10. In Figure 9 we show the results of listening to a Seattle pop music radio station (FM 96.5 KYPT) for 7 days. We plot the rate at which unique new objects were found, and the distribution of play frequencies. We also give a pie chart showing the portions of the stream that are made up of repeating objects greater than 2 minutes, less than 2 minutes and non-repeating content. Roughly speaking these might be taken to represent the portion of the streams that are songs, advertisements and talk respectively. Clearly in Figure 9 (a) very few objects were found in the first day. This reflects the fact that objects will not be recognized until they repeat, and repeats do not happen immediately. Once repeats start to occur (after one day) objects are found rapidly at first and then tail off, with a total of 766 unique objects of length 2 minutes or greater being found. Figure 9 (b) show the play frequencies of the objects found; some objects played almost every day while others appeared only twice in the seven day period. In Figure 10 we show the results from listening to FM 107.7 KNDD for a period of 6 days. Comparing the results from the two stations is interesting; clearly KYPT plays a somewhat larger collection of objects, while KNDD plays fewer objects (a total of 219 unique objects of length 2 minutes or greater were found) but plays some objects very often. The two pie charts Figures 9 (c) and 10 (c) show the difference in makeup of the streams. On KYPT almost 75 % of the stream consists of RO’s of length 2 minutes or greater (presumably music) while only 7.2 % was contained.
no repeats (presumably talk or objects that occurred only once and hence were not detected as repeats). For KNDD these fractions were 39% and 47% respectively.

C. Applications

We have demonstrated that a system capable of extracting repeat objects from streams can operate in real-time on a consumer level PC. This enables a number of applications.

1) Customization and Adaptation: Customization of the stream can clearly be achieved by associating an action to be performed on the stream to the object. Every time a particular object appears an action is performed. This is useful in a number of scenarios:

- Parents might wish to delete objects that they deem unsuitable for their children
- Listeners and viewers might wish to store copies of favorite objects, or delete objects they do not like
- Broadcasters might wish to substitute commercials suitable for local markets, or tuned to the preferences of the viewer.

The method of associating an action with an object can vary. For example, in the parental control application a suitable UI might be as simple as a button to enable a parent while viewing an object to indicate that all future occurrences are to be deleted. Commercial substitution could be accomplished by the broadcaster associating a substitution action with an object based on the viewer’s location and/or preferences.

2) Compression and Archiving: Compression of media streams generally exploits the correlation in the stream. The correlations exploited, however, are typically among portions of the signal less than a second apart. When streams contain whole objects that repeat there are correlations on much longer times scales. The library of repeating objects found by searchLibraryAndBuffer() forms a codebook; a stream can be sent for the cost of the codebook plus the pointers to the appropriate entries.

3) Gathering Statistics: The ability to recognize repeating objects allows us to build a statistical picture of the composition of a stream. Basic examples were shown in Figures 9 and 10, but various other statistics on the streams can be gathered. We can compile histograms of relative frequencies of objects, and compile "Top 10" lists of the most popular objects.

V. CONCLUSIONS

We have shown that identifying repeats in media streams that are days or even weeks apart is perfectly feasible for a consumer PC. Key to this process was selection of a low dimension representation that reduced the complexity of the search, exploiting the repetitive nature of the stream, and use of two copies to identify the endpoints of the object. The main contributions are that we show how to efficiently search for unknown objects, that for repetitive streams the complexity converges to that of searching for known objects, we show how to determine the endpoints of objects, and that buffering large sections of the stream is not necessary to extract even objects that recur infrequently. A number of applications are advanced.
A. Acknowledgments

The author would like to thank Chris Burges, Phil Chou, Paul England, Dinei Florencio, John Platt, Erin Renshaw and Jay Stokes for numerous discussions on the subject of this work.

REFERENCES

Fig. 1. Example of two portions of a stream that are approximately equal. Above: first signal, Center: second signal. Observe that the latter half of the second signal approximately coincides with the earlier half of the first signal. Below: Cross-correlation of first and second signals. Observe that the prominent peak indicates the similarity of the two signals, and position of the peak the amount by which they are offset.

Fig. 2. Probability that two or more copies simultaneously appear in a one day buffer for objects with various probabilities $p_i$. Observe that even an which repeats on average every $E/rp_i = 3$ days has a 50% probability of being detected after 10 days.
Fig. 3. Block diagram of system to find repeats using fingerprints. One thread calculates fingerprints and enters them in a database, another calculates fingerprints and checks the database. When a new fingerprint matches an old one an RO has been found.
Fig. 4. Two different copies of the same object captured at different times from an FM radio broadcast. The representation is BB7 sampled at 11 samples/sec. The center portions of the two copies approximately coincide, while the beginning and end portions do not. The similarity of the segments is evident, even though we are using a very low dimension representation.

Fig. 5. Example indicating how the boundaries of an object can be calculated once two or more copies have been found. Overlay of the first and second instance of the object once they have been aligned. The points where the two streams diverge at the beginning and end are the endpoints of the object.
Fig. 6. Improving the efficiency of searching for unknown repeating objects. A buffer of length one days is assumed, for the two distributions shown. (a) Number of calls to ApproxEqual(). Observe that the number of calls begins as the cost of searching the buffer, but rapidly converges to the cost of searching the library. (b) Fraction of the ROs played in a stream found. Observe that after two days almost every RO being played has already been found.
Fig. 7. Block diagram of a system to automatically identify and extract repeating objects from a media stream. The media stream enters a full rate buffer and is delayed before playing. A low dimension version of the stream is sent to a low dimension buffer. This buffer is constantly searched for repeating objects using routine searchLibraryAndBuffer(). Found objects are stored in an RO library. Boundaries calculated by comparing two copies in the low dimension buffer are used to generate the boundaries in the full rate buffer. Deletion and copying then become simple. The full rate buffer needs only to be longer than the longest expected RO. The low dimension buffer needs to span an interval long enough to allow objects to repeat. This allows objects that occur days apart to be found without having to buffer the full rate stream.
Fig. 8. Evolution of the full rate buffer. The ROs are found by searching the low dimension buffer (not shown). (a) Objects A-G have been found and their boundaries marked. The user is currently viewing object A. (b) User skips object A. The read pointer advances to the next object boundary. (c) User skips object B. The read pointer advances to the next object boundary. (d) User skips object C. The read pointer advances to the next object boundary. Observe that the read and write edges of the buffer are getting closer. (e) System inserts objects W, X, and Y between F and G.
Fig. 9. Repeating Objects detected on a real broadcast stream. The Seattle pop radio station KYPT FM 96.5 analyzed for a period of 7 days. (a) Number of unique ROs found as a function of time. For the first day few objects are found, since repeats do not occur. After the first day many objects are found initially, but the rate slows as fewer objects remain to be found. A total of 766 unique ROs greater than 2 minutes in length were found. (b) Number of times that unique ROs were played in stream. A small number of popular objects are played very frequently and the stream is very repetitive. (c) Pie chart of the ROs greater and less than 2 minutes in length and non-repeating parts of the stream.
Fig. 10. Repeating Objects detected on a real broadcast stream. The Seattle pop radio station KNDD FM 107.7 analyzed for a period of 6 days. (a) Number of unique ROs found as a function of time. Many objects are found initially, but the rate slows as fewer objects remain to be found. A total of 219 unique ROs greater than 2 minutes in length were found. (b) Number of times that unique ROs were played in stream. A small number of popular objects are played very frequently and the stream is very repetitive. (c) Pie chart of the ROs greater and less than 2 minutes in length and non-repeating parts of the stream.