Accurate Repeat Finding and Object Skipping Using Fingerprints

Cormac Herley
Microsoft Research, One Microsoft Way, Redmond, WA

Abstract

This paper introduces a novel and very accurate segmentation algorithm. It is very efficient and consumes less than 10% of CPU on a simple desktop PC to segment a stream in real-time. It operates on an audio stream, or on the audio portion of a audio-visual stream. It is very accurate: it accurately detects the positions and durations of objects on an over-the-air broadcast television signal, and songs on both FM and internet radio stations (as checked against labeled ground truth streams). The algorithm does not require any prior information or training. We detail the system design and present results of segmenting broadcast streams.

Keywords

Repeat finding, Segmentation

1. INTRODUCTION

Segmentation is far from being a new problem. Many applications in the multimedia research literature depend on it, and it has been well studied. However, we believe our algorithm to be a considerable departure from previous approaches. Rather than seeking semantic structure, visual, acoustic or other features we seek to explicitly identify repeating segments of the stream. We demonstrate how this may be done very efficiently using recent advances in audio fingerprinting, even on high rate streams. Object skipping is one of the obvious applications enabled by this algorithm, but it has implications in many other areas.

Previous approaches to segmentation of course depend on the type of content and on the application. A system that attempts to automatically identify periods of interesting activity in a soccer game will be very different from one that attempts to identify news sequences on a broadcast television stream. A commercial skipping system might have little in common with a method to build an ontology based on scene similarities. Nonetheless the somewhat loosely defined task of segmenting streams occurs in a wide variety of applications.

In addition to applications where segmentation is the end result, there are numerous applications where it is a component of a larger system. Content indexing and retrieval systems for example must identify segments that correspond to user expectations of coherent scenes or units. A system that attempts to find clips similar to a given clip, or allows browsing based on some measure of similarity falls into this category. Variations also arise in automatic summarization, content indexing and retrieval. A large fraction of the segmentation systems in wide use today have dedicated algorithms that are particular to the type of segment being sought, and the application at hand.

In the next section we explore related and previous work. In Section 3 we present the algorithm and show how repeating objects may be located and segmented, and explore the complexity. In Section 4 we present the results and statistics of segmenting several real broadcast streams.

2. RELATED WORK

2.1 Video Segmentation Systems

Since stream segmentation is such a large area we can give no more than a sampling of recent related work. Breaking broadcast video streams into different classes of objects, has been addressed by numerous authors. Jiang *et al.*, for example, demonstrate segmenting and classifying scenes on a news channel into categories such as news-anchor scenes, live report scenes, weather reports and so on [16]. Theirs is one successful example of an approach that seeks particular features that are trained to classify certain types of content. Sundaram and Chang [25] employ separate audio and video segmentation algorithms that model scenes as semantically consistent chunks, and then integrate the results. Rui *et al.* [24], describe a system that automatically extracts segments of greatest activity from sports events such as baseball games.

Considerable effort has addressed the question of identifying objects based on a set of known features. Hampapur and Bolle [11], for example, describe a feature based indexing system that allows real-time similarity operations on streams. This allows tracking and search to be performed on broadcast video. Many of the approaches to *video* stream analysis make use of the growing body of work on *audio* analysis, querying and retrieval. Wold *et al.* [26], for example, describe a system of audio content classification and search by reducing audio to a set of perceptual and acoustic features. Pfeiffer *et al.*, [22] describe content-based segmentation of an audio stream, music analysis and a method to detect violent scenes based on the audio channel. Each of these approaches can be used in systems that attempt to segment based on audio alone, or in combination with video cues. Muramoto and Sugiyama [20], for example, follow such an approach using a system of audio and visual cues. Pass *et al.* use coherence vectors defined over the color information in a frame to derive a vector to compare frames based solely on the visual information [21]. One advantage of the algorithms that can segment based on audio data alone [22, 26] is that segmentation can be done on a compressed format video stream without having to decode the whole stream; *i.e.* the segment boundaries can be calculated without a full decode operation.

The Scene Transition Graph developed by Yeung *et al* [27] identifies related scenes in a stream. Distinct scenes are identified by detecting shot boundaries and building a graph based on scene similarities. In [27] related scenes are clustered (*e.g.* closeups of the same person in a dialog scene). This differs from our work in that we are interested only in identifying segments of streams that are identical (other than channel deformations).

2.2 Commercial Skipping Systems

The application of commercial skipping has attracted considerable attention, both in the multimedia literature and in consumer electronic products. The goal is to identify the location and boundaries of commercials and allow the viewer to skip them at will. Among commercial offerings TiVo [1] is possibly the best known, which offers its users the ability to skip commercials when they are watching a recorded show. The technology appears to make use of the fact that commercials are preceded by two blank frames in most television and cable broadcasts. These blank frames are inserted as a marker by the broadcasters, and of course make identifying and skipping the commercials simple. Another consumer electronics offering is that of ReplayTV [2] which has a 30 seconds advance button. This relies on the even simpler observation that commercial advertisement slots on television are typically 15, 30 or 60 seconds in length. While both of these systems work, the solutions are somewhat heuristic and fragile: it is clear that they will cease to work altogether if broadcasters alter their commercial format.

Thus there continues to be research in this area. Bimbo *et al.* [4] report considerable success in distinguishing commercials from regular programming content by using an efficient vector based on the evolving color information. Alternatively, Lienhart *et al.* [19] show how a library of known objects such as commercials may be detected and segmented efficiently. Our work differs from [19] in that we do not require a known collection before beginning; in fact our system bootstraps to learn the collection of objects that are repeating in the stream. Kashino *et al.* describe another interesting approach along these lines [18], which explores a histogram pruning method to search for known objects in long streams or libraries. Both [19] and [18] have in common with our work that they are explicitly interested in finding objects in very long sequences (*i.e.* days or weeks); the point of difference is that they take the set of sought objects to be known.

2.3 Repeat Finding Systems

The subject of repeat finding in *noise-free streams* has a rich history. Using dedicated algorithms and efficient data structures such as suffix trees or tries it is possible to find all repeating subsequences of a finite alphabet stream. These algorithms are the basis of the renowned Lempel-Ziv compression, for example. For a length m string over an alphabet of size Z all patterns of length L can be found in $O(L)$ time, but in general the tree requires $O(mZ)$ space [9]. The text by Gusfeld [9] is a good reference on the numerous variations on the theme, but string matching algorithms become impractical as the alphabet size increases. In addition to the memory problems, most multimedia streams are distorted by compression and channel noise, so that exact matches do not occur. There do not appear to be successful applications of string matching ideas to repeat finding in multimedia streams.

An interesting example of searching for unknown repeating objects in a *music* stream is by Hsu *et al.* [14]. Here, however, the authors work on MIDI data rather than a raw audio format. So, in this sense the stream is noise-free, and there is no clear generalization to video streams. An approach by Johnson and Woodland [17] describe a direct audio search method. In common with [19, 13] and our approach the authors seek exact repeats. They describe an efficient algorithm for finding repeats of given pieces of cue audio (such as station call signs). They do not address the problem posed here; *i.e.* automatically extracting the unknown repeating objects in a stream. Cooper and Foote [8] describe an elegant approach to video summarization by establishing similarity between scenes. They establish a matrix of relative similarities between scenes and choose as summary the scene with the highest average similarity to its companions. Our approach by contrast is to extract *exactly repeating objects.* Rather than seek clusters of objects that are similar, we seek repetitions of objects that are precisely the same. The only errors we deal with are erasures, compression and channel errors, so repetitions are unambiguous.

Herley [13, 12] directly tackles the question of identifying repeats in multimedia streams. There, cross-correlations are used to identify repeats, and dimension reduction techniques make the computation manageable. While we address essentially the same problem as [13], our algorithm is far more computationally efficient (as analyzed in Section 3.5). The segmentation is also a great deal more accurate; since we rely on fingerprints which are known to have very low error rates, while [13] relies on a somewhat heuristically derived Bark-Band method.

2.4 Audio Fingerprinting Algorithms

A set of highly performant algorithms based on linking unlabeled audio to a database entry have been reported recently by Haitsma and Kalker and [15] and by Burges *et al.* [6]. While we in no way overlap with these contributions we actually use a fingerprint algorithm as a component of our algorithm, and hence briefly review them here. Depending on the application area these are variously known as audio fingerprinting, Content Based Identification, or audio hashing algorithms. There is even a successful web service that allows users to identify music played over a cellphone [3]. The idea is that a perceptual signature of a segment of audio is calculated and this signature can be compared with the signatures in a labeled database. In many ways the signature is analagous to a software hash such as CRC. Like their software counterparts audio hashes are generally much smaller than the audio segments they represent, are efficiently calculated, have very low instances of collision (*i.e.* two different audio segments having the same signature), and (ideally) can be compared against a large database in better than linear time. In addition audio hashes have a requirement that their software cousins do not: they should be robust to noise and deformations. Cano *et al.* give an excellent review of audio fingerprinting algorithms in [7].

It is beyond the scope of this paper to detail the inner workings of fingerprint algorithms; interested readers are referred to [15, 7, 6] and to experiment with the web service [3]. Since the fingerprint forms a vital block of our algorithm it is important that readers understand how performant these algorithms are. In our implementation we have used the fingerprint described in [6]. By way of benchmarks on this algorithm Burges *et al.* document in [6] that performing searches every 0.186 seconds in a database of 240000 fingerprints consumed less than 5% of CPU on a 1.2GHz PC. This system was demonstrated at ACM MM 2003 [5]. In addition they document robustness to various distortions such as FM transmission, amplitude distortion, considerable noise addition and time alignment. The fact that the service offered by [3] allows identification of music played over a cell phone also can be taken as demonstration of the considerable noise robustness of current fingerprinting schemes. In the rest of the paper we will refer to $h[\mathbf{s}(n)]$ as the *audio fingerprint* of a stream at time n. Following [6] this will be a 256 byte record similar to a hash. We will use the notation $h[\mathbf{s}(n)] \approx h[\mathbf{s}(n-k)]$ to denote that the fingerprints at times n and $n-k$ match, and this the content at those locations is the same.

3. FINDING AND SEGMENTING OBJECTS

3.1 Problem Setup

Commercials are perhaps the most obvious example of objects that repeat in a multimedia stream. Since the system we describe is not limited to commercials we first clarify our definition and give examples of typical repeats in various multimedia streams.

First, when an object repeats we mean that a segment repeats *exactly* except for noise caused by channel distortion or compression. For example: commercials that repeat without variation, taped news items that are replayed every hour, and the signature sequence at the beginning of a broadcast television show would all be repeats under this definition. We should also clarify certain objects that *are not* repeats. On a video stream a commercial that appears once with English audio and once with Spanish audio would not be a repeat; these are distinct objects under our definition. In an audio stream two different recordings of the same song would not be considered repeats of the same object: generally the difference between two recordings of a song, even when recorded by the same artist, is far greater than is typically introduced by channel or compression distortions.

We should make clear that we expect our system to be robust to the various channel errors that occur. Just as important as the channel distortions introduced by the physical transport mechanism are the intentional distortions introduced by broadcasters in editing their content. The fingerprinting systems [15, 6] that we employ were carefully designed to be robust to these distortions. In fact detailed analyses of their ability to withstand, for example, time axis compression and companding are among the distortions measured in [10, 6] and [3].

3.2 Production model for multimedia streams

Our model of a multimedia stream is that it contains a mixture of repeating objects and non-repeating content. Without loss of generality we say it is synthesized by choosing objects from a library of K objects $\mathbf{O}_0(n), \mathbf{O}_1(n), \cdots, \mathbf{O}_{K-1}(n)$. Denote by L_i the length of $\mathbf{O}_i$ in seconds, so that the object is $L_i R$ samples long, where R is the sampling rate. Observe that $\mathbf{O}_i$ can be one-dimensional or multi-dimensional, in the case of video. We stress that L_i can vary from a few seconds to several minutes, depending on the stream, and that K might be as small as a few hundred or range into the tens of thousands (we give examples in Section 3.2.1 and measurements in Section 4). Since a stream may not consist entirely of repeating content we will also say 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. This model is not at all restrictive, and is sufficiently general to capture most broadcast streams.

Call the overall stream $\mathbf{s}(n)$. An example of such stream might be written

$$\mathbf{s}(n) = \{\cdots|\mathbf{O}_{495}(n)|\mathbf{i}_d(n)|\mathbf{O}_4(n)|\mathbf{O}_{343}(n)|\mathbf{O}_{17}(n)|\cdots\},$$

where the symbol $|$ denotes concatenation of objects, and $\mathbf{i}_d(n)$ denotes a non-repeating segment of duration d. Clearly, if an object $\mathbf{O}_i(n)$ is repeated at times n_0 and n_1 in the stream we will have

$$\mathbf{s}(n_0 + k) = \mathbf{s}(n_1 + k), \text{ for } 0 \leq k \leq L_i R. \qquad (1)$$

The synthesized stream is generally carried over a channel before being consumed by the user. This may consist of a terrestrial broadcast, the internet, cable or other distribution network, and the transport mechanism may be either analog or digital. In any of these cases we make the assumption that what is received is corrupted by noise:

$$\mathbf{r}(n) = \mathbf{s}(n) + \mathbf{n}(n).$$

The noise can have any of the characteristics typical of real communication systems (we enumerate some examples in Section 3.2.1), we merely assume that the signal strength is much greater than the noise: $\| \mathbf{s} \| \gg \| \mathbf{n} \|$. Thus, following (1), when a repeat occurs, we will have for the received signal:

$$\mathbf{r}(n_0 + k) = \mathbf{r}(n_1 + k) + \mathbf{n}(n_1 + k), \text{ for } 0 < k < L_i R.$$

This in turn implies that the fingerprints at these locations match:

$$h[\mathbf{r}(n_0 + k)] \approx h[\mathbf{r}(n_1 + k)], \text{ for } 0 < k < L_i R. \qquad (2)$$

Finally we stress that this model is valid both for audio and video streams. For fixed n, thus $\mathbf{s}(n)$ would be a pair of 16-bit numbers in the case of stereo audio, and R might be 44100 samples per second. In the case of NTSC video $\mathbf{s}(n)$ might be a 640×480 image, and R might be 50 (though an accompanying audio stream would be sampled at a higher rate).

3.2.1 *Example streams*

Terrestrial FM music broadcast: the $\mathbf{O}_i$ consist of songs, generally with lengths between 150 and 240 seconds, commercials that are either 15, 30 or 60 seconds in length, and the jingles and signature tunes (of variable length) that separate program elements. A commercial pop music station generally plays on the order of 200-500 songs, and a collection of 200 or so commercials. Alternative, or college radio stations can play from libraries of thousands of songs. Of course K will be the sum of all of the song, commercial, jingle libraries on a station. Noise is determined by the terrestrial broadcast, so there can be fading components and additive components. The sampling rate R is determined by the acquisition system rather than the transmitter. $\mathbf{r}(n)$ is one dimensional.

Internet radio: the $\mathbf{O}_i$ consist of songs, generally with lengths between 150 and 240 seconds, there may or may not be commercials, and the jingles and signature tunes that separate program elements (of variable length). The dominant source of noise is erasures that occur in sending a UDP stream, and compression errors due to mp3 or similar coding. The sampling rate R is generally determined by the transmitter. $\mathbf{r}(n)$ is one dimensional.

Analog Cable Network News: the $\mathbf{O}_i$ consist of news stories with lengths between 30 seconds and 10 minutes, commercials that are either 15, 30 or 60 seconds in length, and the jingles and signature tunes that separate program elements (of variable length). The dominant source of noise is channel noise. The sampling rate R is generally determined by the transmitter. $\mathbf{r}(n)$ is three dimensional.

3.3 Finding Unknown Repeating Objects

3.3.1 *Brute Force Algorithm*

As pointed out in [13] searching for repeating patterns in streams is trivially simple if memory and computation are not constrained. We can simply compare a chunk of the current stream with previous chunks up to some maximum time D_{max} seconds in the past. That is, check if

$$\text{Chunk}[\mathbf{r}(n)] \approx \text{Chunk}[\mathbf{r}(n-k)] \text{ for } k = L_{min}R, \cdots, D_{max}R.$$

When equality occurs for some k_x we have found a repeat at locations n and $n - k_x$.

Rather than compare stream segments directly we can compare their fingerprints, which is considerably more efficient and also more noise robust. Relying on the fact that false positives are extremely rare, we can compare the fingerprint at some time instant with that at all other instants. Thus let $h[\mathbf{r}(n)]$ be the fingerprint at instant n. To determine if the current stretch is a repeat, for example, we might compare a fingerprint of the most recent sample with a fingerprint at every sample in a finite buffer of length D_{max} by exhaustive search. That is check

$$h[\mathbf{r}(n)] \approx h[\mathbf{r}(n-k)] \text{ for } k = L_{min}R, \cdots, D_{max}R.$$

As before: when a match occurs for some k_x we have found a repeat at locations n and $n - k_x$. For large enough D_{max} every repeating object can be found in this way.

The problem with this approach, of course, is that it is computationally infeasible: we have to store $D_{max}R$ fingerprints of the stream, and have to search that store R times per second. Let's assume a buffer of length a week, so $D_{max} = 60 \cdot 60 \cdot 24 \cdot 7$, and (following the implementation in [6]) assume that each fingerprint consumes 64 floats or 256 bytes. Thus the buffer would require $D_{max}R \times 256 = 6.8$ terrabytes, and a total of $D_{max}R^2 = 1.2e15$ fingerprint comparisons per second would be required. This is several orders of magnitude beyond the capacity of even the fastest computers.

Clearly, this brute force algorithm is overkill: there is no need to check the fingerprint against the entire buffer at every single time sample. To address the storage and computation requirements we seek an approach that operates only on samples of the stream.

3.3.2 Sampled Algorithm

Consider a new algorithm that consists of two threads that run infinitely.

- Thread 1: at time pM_w calculate $h[\mathbf{r}(pM_w)]$, and deposit in a database.
- Thread 2: at time qM_r calculate $h[\mathbf{r}(qM_r)]$, and check whether it matches any fingerprint in the database.

Since we wish to search only a distance D_{max} into the past we will say the database is full when there are a total of $D_{max}R/M_w$ fingerprints entered; and at this point the database acts as a FIFO. This database is checked R/M_r times a second. So there are a total of $D_{max}R^2/(M_wM_r)$ fingerprints comparisons per second. Hence, in writing fingerprints only at intervals M_w, we reduce the database size and the computation by a factor of M_w. In checking the database only at intervals M_r we reduce the computations by a further factor M_r.

Recall now one of the important properties of the fingerprint algorithms mentioned earlier was alignment robustness. That is suppose that $\mathbf{O}_i$ repeats at n_0 and n_1. We would thus expect that

$$h[\mathbf{r}(n_0 + k)] \approx h[\mathbf{r}(n_1 + k)] \text{ for any } 0 \leq k \leq L_iR.$$

That is, fingerprints in two copies of $\mathbf{O}_i$ at n_0 and n_1 match so long as the relative offset, k, from the beginning is the same. Actually, at least using the fingerprints in [15, 6], something stronger is true. The relative offsets don't have to be *exactly* equal. So long as they are within δ_T of each other the fingerprints still match:

$$h[\mathbf{r}(n_0 + k)] \approx h[\mathbf{r}(n_1 + k + \delta_k)]$$

for $0 \leq k + \delta_k \leq L_iR$ and $|\delta_k| < \delta_TR$. Under experimental conditions [6] the alignment tolerance of the fingerprint algorithm we use was determined to be $\delta_T = 0.186$ seconds (or $\delta_TR = 8202$ samples). This leads the way to a huge computational and buffering simplification over the brute force algorithm.

3.3.3 Choosing the write and check rates M_w and M_r

It remains to find the conditions on the write rate, M_w, and the check rate, M_r, such that all repeating objects are found. Suppose $\mathbf{O}_i$ occurs in the stream at n_0, and that Thread 1 writes a fingerprint to the database at some time $pM_w \geq n_0$. Next suppose $\mathbf{O}_i$ recurs at time n_1. To match the written fingerprint Thread 2 must check the database at some time in the range

$$n_1 + pM_w - n_0 - \delta_TR \leq qM_r \leq n_1 + pM_w - n_0 + \delta_TR.$$

This requires $M_r \leq 2\delta_TR$. Observe, however, that we do not have to match a *particular* fingerprint. If several fingerprints were written by Thread 1 for $\mathbf{O}_i$ we merely require that *one of* the fingerprint checks from Thread 2 matches *one of* the fingerprints written by Thread 1.

Some solution pairs M_w, M_r are easily found. For example, suppose we choose $M_w = L_{min}R$. This guarantees that even the shortest repeating object encountered will contain at least a single fingerprint entry in the database. Then if we choose $M_r = 2\delta_TR$, there will always be a fingerprint check within $2\delta_TR$ of it on any subsequent repeat copies of this object. This strategy involves making M_w as large as it can be, but still guarantee at least one fingerprint per object, and then check the buffer frequently enough to ensure this fingerprint is never missed. This is shown in Figure 1 (a).

At the opposite extreme we can choose $M_w = 2\delta_TR$ and $M_r = L_{min}R$ as shown in Figure 1 (b). This strategy involves checking the buffer only once per object on average, but we place enough fingerprints in the database to ensure that each object has many. Observe that the computation in both of these cases is the same, $D_{max}R^2/(M_wM_r)$ fingerprints comparisons per second.

These are by no means the only choices however. We will see in Section 3.4 why small values of M_w are advantageous for the segmentation. If fingerprints are written with spacing M_w, then there will be L_iR/M_w entered for object $\mathbf{O}_i$. Thus, there is a subset of total length $\delta_TR \cdot L_i/M_w$ of the length L_i, and if one of the fingerprint checks falls within this subset a match will be found. Actually, with some restrictions, we can choose:

$$M_r \leq \frac{L_{min}R}{M_w} \cdot 2\delta_TR. \tag{3}$$

The main restriction is that M_w and M_r be incommen-

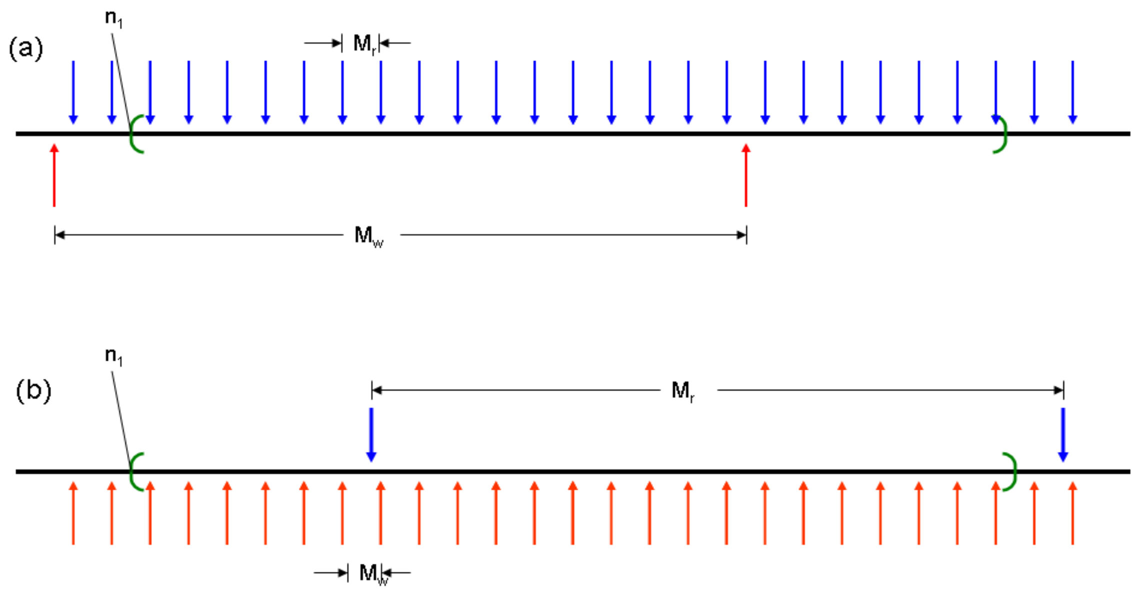


Figure 1: Examples of choices of the write and check rates for fingerprints M_w and M_r. (a) A very coarse write rate $M_w = L_{min}R$ requires a very fine check rate $M_r \leq 2\delta_T R$. (b) A very fine write rate $M_w = 2\delta_T R$ allows a very coarse check rate $M_r \leq L_{min}R$. Our ability to estimate the start position n_1 is determined by how tightly it is bracketed by fingerprints in the database. Hence, the smaller M_w the more accurate the segmentation can be.

surate, *i.e.* $\gcd(M_w, M_r) = 1$. The proof, which is somewhat technical is given elsewhere.

This shows however, that in addition to the solution we have already seen, $M_w = L_{min}R$ and $M_r = 2\delta_T R$ there is a whole range of possible solutions. Observe that the computation will remain unchanged at $D_{max}R^2/(M_w M_r)$ fingerprints comparisons per second. However, the database contains $D_{max}R/M_w$ fingerprints. Clearly smaller values of M_w will ensure that the database has more entries. We will see next why this can be advantageous.

3.4 Segmentation

Using the sampled algorithm we find a repeating object when Thread 2 finds a match in the database:

$$h[\mathbf{r}(pM_w)] \approx h[\mathbf{r}(qM_r)],\ pM_w \neq qM_r$$

for some integers p and q. That is, a fingerprint at one section of the stream matches one from a previous section. Once Thread 2 finds a match we know that a repeat has occurred. However we we do not yet know the starting position of each of the copies. We know merely that there's a copy of an object before pM_w; *i.e.* at some location n_0 where $pM_w - L_{max}R \leq n_0 \leq pM_w$. And we know that there's another copy before qM_r; *i.e.* at some location n_1 where $qM_r - L_{max}R \leq n_1 \leq qM_r$. Equally we do not know the length of the object.

However, since this match represents two copies of an object $\mathbf{O}_i(n)$ of length L_i the current and previous streams will match not just at the locations of the fingerprints found (*i.e.*, pM_w and qM_r) but also at locations on either side of these instants for a total length of L_iR. Thus we ought to be able to determine, the beginning position of the two copies of $\mathbf{O}_i$, and its length by tracing fingerprints backwards and forwards. That is we know that (2) holds, not just at the particular positions just found by Thread 2, but over a range of values of length L_iR.

We can't evaluate (2) at every value in this range however, since we do not have access to fingerprints at arbitrary locations in the past. We merely have the fingerprints placed in the database by Thread 1, which are spaced M_w apart. Thus, we can compare:

$$h[\mathbf{r}(pM_w + kM_w)] \approx h[\mathbf{r}(qM_r + kM_w)]. \qquad (4)$$

Clearly the boundaries of the object can be approximately found by choosing

$$n_0 = pM_w + k_{left}M_w,$$

$$L_iR = pM_w + k_{right}M_w - n_0,$$

where k_{left} is the smallest k and k_{right} is the largest k for which (4) holds.

Observe that in order to evaluate k_{left} and k_{right} we need only fingerprints spaced M_w apart in the vicinity of n_0 so everything needed is available in the database of fingerprints. Observe also however, that we can determine the endpoints only to within a tolerance of M_w. That is, if

$$h[\mathbf{r}(pM_w + kM_w)] \approx h[\mathbf{r}(qM_r + kM_w)],$$

but

$$h[\mathbf{r}(pM_w + (k-1)M_w)] \not\approx h[\mathbf{r}(qM_r + (k-1)M_w)]$$

we then know that

$$pM_w + (k-1)M_w \leq n_0 \leq pM_w + kM_w,$$

but cannot determine n_0 and n_1 more accurately than that. So M_w gives the expected error in the segmentation. This then gives us a good reason to choose M_w to be small. If we choose $M_w = R$ for example we will have a maximum error of one second in determining the object boundaries.

So the boundaries of an object are easily calculated once a match is found. Since we compared the two copies of the stream in increments spaced M_w apart, the worst case error in determining both the left and right boundaries of $\mathbf{O}_i(n)$ will be M_w samples.

3.5 Complexity and comparison with [13]

Let's compare the complexity of the sampled algorithm given above with the approach described in [13].

First, our approach requires searching a database of $D_{max}R/M_w$ fingerprints every M_r/R seconds. We have not thus far addressed the structure of the database, and how it may be searched efficiently. Assume the worst case: we do a linear search by comparing the current fingerprint with every fingerprint in the database every time. Using (3) this gives a total of

$$\frac{D_{max}R^2}{M_w M_r} = \frac{D_{max}}{L_{min}2\delta_T}$$

fingerprint comparisons per second. Each comparison merely involves comparing the difference between two 64 element vectors with a threshold.

Now consider the complexity of the scheme in [13], which involves comparing correlations of blocks of data to find repeats. The schemes are not directly comparable, but [13] also uses block sizes determined by L_{min}. It divides the desired search window (of length D_{max}) into $2D_{max}/L_{min}$ windows of length L_{min} with overlap of 50 %. A search for correlations is done every $L_{min}/2$ seconds, and it involves comparing the current block with each of the blocks in the buffer. Thus there are $4D_{max}/L^2_{min}$ correlation comparisons per second. However the comparisons involve taking the FFT of the current block, multiplying against one of the buffered blocks, taking the inverse FFT, and then searching for a peak. Thus each correlation comparison can be expected to take computation proportional to $2L_{min}(\log L_{min} + 4)$. Hence the overall, best case computation is

$$\frac{8D_{max} \cdot L_{min} \log L_{min}}{L^2_{min}} = \frac{8D_{max} \log L_{min}}{L_{min}}$$

operations per second.

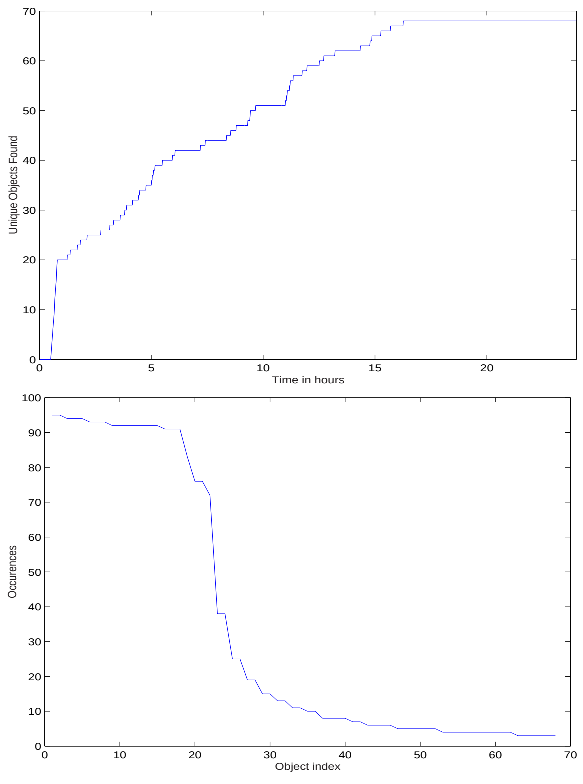


Figure 2: Commercials and other repeats found on CNN headline news. (a) Number of unique repeats found as a function of time. (b) Number of times each object was found.

Hence the *worst case* complexity for our scheme is almost an order of magnitude better than the best case complexity for [13] when L_{min} is small; and the complexity advantage increases as L_{min} increases. Further, we assumed that the database in our scheme was searched linearly, as this was the case in our current implementation. Clearly, using any of the efficient strategies for searching databases will improve the complexity advantage even further.

4. ARCHITECTURE AND RESULTS

4.1 Architecture

We have used an implementation of the fingerprinting algorithm of [6] as the basis of our algorithm. A video acquisition card on the PC receives the video input, and audio is input to "line-in" on the soundcard. All of the writing and checking of fingerprints is done only on the audio portion of the signal. Whether the signal is video or audio alone we always capture the audio portion at 44100 samples per second, and this (along with L_{min}) determines the read and check rates of the algorithm. The database created by Thread 1 stores fingerprints every M_w samples and contains $D_{max}R/M_w$ entries. A separate thread does all of the checking as described in Section 3.3.2.

In order to skip objects obviously the system must be buffered. The length of the buffer must be at least as long as the sum of all the objects that will be skipped. Since our implementation is for proof of concept rather than actual use we buffer only 20 minutes of the raw stream. Thus, when the input is an audio signal, the user listens to the audio with a 20 minute delay. If the user skips objects the read and write edges of the buffer get closer together; when they are too close together skipping is no longer possible. When an object is found the boundaries are determined using the method in Section 3.4. If the object has not been previously found a single copy is kept for verification purposes. This is done by cutting the segment from the 20 minute buffer of the raw stream. On all subsequent occurrences of that object we merely record the occurrence in a log. Thus after listening to a stream for a period of time we will have a library of all of the repeats found, and a log of the times when they occurred.

Parameter settings: we always assume CD quality audio acquisition (*i.e.*, $R = 44100$) if we seek songs of length at least $L_{min} = 150$ seconds and will tolerate a segmentation error of one second we can choose $M_w = 44099$, and $M_r = 1476501 < L_{min}\delta_T/M_w$. Note that $\gcd(M_w, M_r) = 1$. Thus the database of $60 \times 60 \times 24 \times 7 = 6e5$ entries must be searched every $M_r/R \approx 33.4$ seconds. This actually consumes less than 2% of CPU on a 2.8GHz PC. If instead we seek commercials and choose $L_{min} = 15$ the numbers change somewhat. Tolerating again a maximum segmentation error of one second we find M_w is unchanged, while $M_r = 123041$; so that now the database must be checked every $M_r/R = 2.79$ seconds. This consumes approximately 3.5% of the same CPU.

4.2 Results

For consistency, in all of our experiments we use database that contains a week worth of fingerprints; *i.e.* $D_{max} = 60 \times 60 \times 24 \times 7$. We will present results on various kinds of streams, such as listed in Section 3.2.1. We will estimate the fraction of repeating content in each of the experiments as

$$r = \frac{1}{L} \sum_i L_i f_i, \qquad (5)$$

where L is the duration of the experiment and f_i is the number of times $\mathbf{O}_i$ was found.

4.2.1 CNN Cable Network News

We set our system to listen to the CNN Headline News cable channel. The channel plays headline news with brief news stories repeated every 15 minutes. Typical stories can be from 45 to 120 seconds in length. To be safe we set $L_{min} = 20$ seconds for this experiment. The results are summarized in Figure 2. Figure 2 (a) shows the number of unique objects found as a function of time over a 24 hour period. Clearly, objects are found almost at the outset, as the first repeats begin after 15 minutes. Figure 2 (b) shows the number of times f_i that object $\mathbf{O}_i$ was found during the interval. Using (5) we estimate $r_{CNN} = 0.91$ meaning that the channel plays almost entirely repeating content. The remaining fraction can be accounted for by objects that played only once (and hence were not found) or were shorter than L_{min}.

4.2.2 NBC Cable Affiliate

We set our system to listen to the NBC cable affiliate for a period of five days. The channel plays mixed content and commercials, which are typically 30 seconds long. We set $L_{min} = 20$ for this experiment. The system found a total of 218 commercials. Common commercials were found mostly at the outset. Using (5) we estimate $r_{NBC} = 0.21$. Clearly this is far lower than the news channel; other than commercials, most of the content appears to be non-repeating.

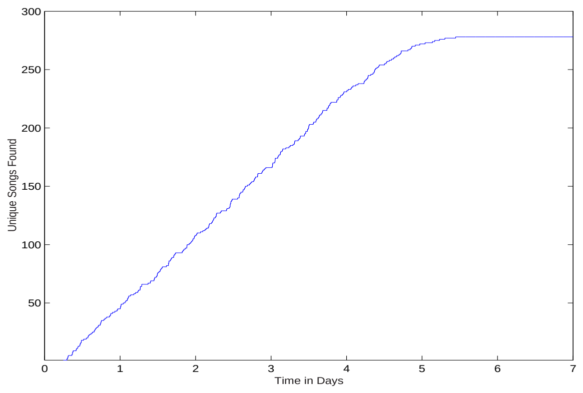
4.2.3 Terrestrial FM music broadcast

We set our system to segment the stream from the Seattle FM pop music station KISW. An FM tuner was connected to the soundcard of the PC. the quality of the FM reception was good. The station plays a mix described as “Today’s hottest music,” along with commercials. We classified all repeating objects found as songs if they were greater than 150 seconds in length, and commercials or other if they were less. In a listening period of seven days our system segmented a total of 273 unique songs, 87 commercials and 11 other objects. The other objects were call signs, jingles and one instance of a repeat of the FCC mandated emergency broadcast system signal. The data is summarized in Figure 3 (a) which shows the number of unique songs found by the system as a function of time. Observe that no songs are found at first, since the fingerprint database is empty, but repeats begin to arrive rapidly after one day or so. The entire collection of songs appears to be captured in less than seven days. We estimate $r_{pop} = 0.87$, indicating again that repeating content dominates the channel.

4.2.4 Internet Radio Station

We set our system to listen to `www.kexp.com` for a period of three weeks. It is a very varied college station. It does not play commercials. To be safe we set $L_{min} = 100$ for this experiment. Our system found a total of 546 objects of 100s or longer that repeated. We estimate $r_{internet} = 0.182$ indicating a lower repeat rate. The results are summarized in Figure 4. In all likelihood a longer listening period would have yielded more repeats, since new repeats continued to be found even after three weeks.

4.3 Evaluation of Results



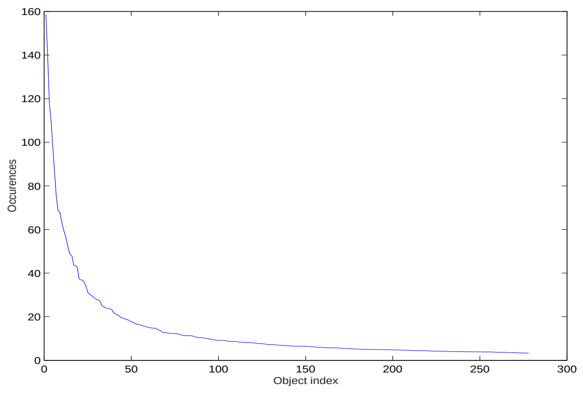


Figure 3: Songs found on an FM pop radio station. (a) Number of unique songs found as a function of time. (b) Number of times each object was found.

4.3.1 *False Positive Rate*

The false positive rate is the percent of erroneous segments identified by the system. For each of the experiments performed this can be estimated by examining a sampling of the actually segmented objects. A false positive of our segmentation algorithm requires a series of consecutive false positives from the fingerprint algorithm (*i.e.* (4) must hold over a series of values). In sampling our segmented objects we failed to find false positives, which is unsurprising.

4.3.2 *False Negative Rate*

The false negative rate is the percent of actual repeats that were not found by our algorithm. To estimate this we examined a 24 hour stretch of the FM station broadcast and labeled by hand the songs played. A total of 137 songs played during that time. We then checked whether these were present in the list compiled by our algorithm. There were two false negatives present in the list. We also examined several commercial breaks on the NBC affiliate channel. Here there were 12 false negatives among 97 hand checked commercials. In all cases the false negatives were 15 seconds in length, and thus missed since we chose $L_{min} = 20$.

4.3.3 *Accuracy of Segmentation*

By listening to a sampling of the songs, and watching

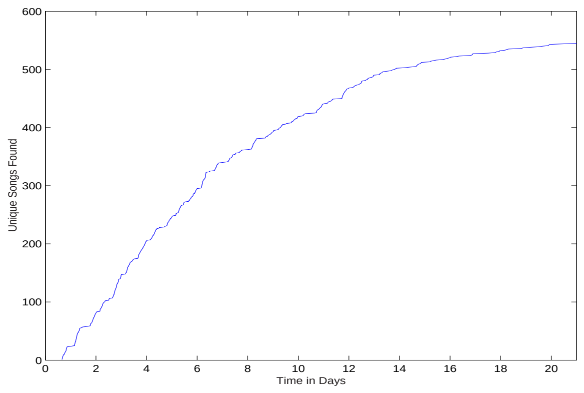


Figure 4: Number of unique songs found on internet radio station `www.kexp.org` as a function of time.

a sampling of the commercials segmented in the various experiments we verify *manually* that the segmentations are quite accurate. Recall, our choice of $M_w = R - 1$ guaranteed an error of about one second. For a more systematic verification we plot in Figure 5 the frequency of segment lengths found in the NBC affiliate broadcast. The cluster 30 seconds represent the most common commercial lengths played on this channel. Assuming that all of these were indeed 30 second commercials, we can see that the segmentations are indeed quite accurate, with average error of 2.3 seconds.

5. CONCLUSION

We have presented a new algorithm that accurately and efficiently segments repeating objects in streams. We have demonstrated that it takes a small fraction of the resources of a desktop PC to identify in real-time, for example, all commercials on a broadcast signal. We have verified the accuracy against ground truth. In addition to object skipping numerous other applications exist: libraries of commercials or songs can be compiled for any stream. Similarities between song objects can be inferred based on an adjacency graph [23]. The advantages of our algorithm are:

- It is efficient: using the worst case exhaustive search of the database it is an order of magnitude faster than the algorithm of [13] (see Section 3.5.
- Since the segmentation is performed on audio, it is possible to segment a compressed video stream without having to decode the whole stream.
- We have measured and verified the segmentations to be accurate (see Section 4.3).

While we believe this algorithm holds great promise for a number of applications where segmentation is important, we also wish to communicate its limitations: the algorithm only finds repeating objects. We believe it outperforms previously reported work for commercial detection, songs segmentation or parsing news streams *so long as we can listen long enough for repeat patterns to become evident.* If listening to a stream for long enough to spot repeats is not possible then the algorithm simply does not work. Equally, for segmenting scenes or content that do not repeat it simply is not applicable.

Acknowledgements: the author gratefully acknowledges discussions and assistance from Chris Burges and John Platt.

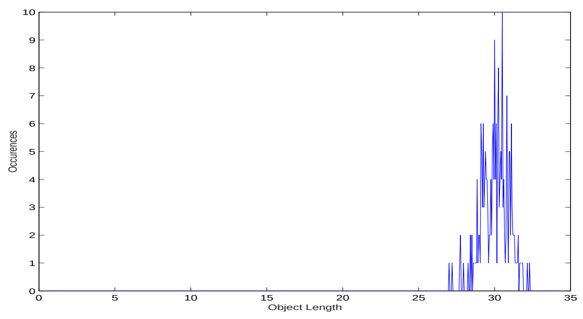


Figure 5: Lengths of commercials found on NBC cable affiliate. Observe the extremely tight clustering about 30 seconds. The variation is a measure of the segmentation error of the algorithm.

6. REFERENCES

[1] `http://www.tivo.com`.

[2] `http://www.replaytv.com`.

[3] `http://www.shazam.com`.

[4] A. Del Bimbo, P. Pala, and L. Tanganelli. Retrieval by content of commercials based on dynamics of color flows. *Proc. ICME*, pages 479–482, 2000.

[5] Chris J. C. Burges, John C. Platt, and Jonathan Goldstein. Identifying audio clips with rare. *Proc. ACM MM Demonstration Session II*, 2003.

[6] C. J. C. Burges, J. C. Platt and S. Jana. Distortion descriminant analysis for audio fingerprinting. *IEEE Trans. on Speech and Audio Processing*, 11:165–174, 2003.

[7] P. Cano, E. Batlle, T. Kalker, and J. Haitsma. A review of algorithms for audio fingerprinting. *IEEE Workshop on Multimedia Signal Processing*, 2002.

[8] M. Cooper and J. Foote. Summarizing video using non-negative similarity matrix factorization. *Proc. IEEE Multimedia Signal Processing Workshop*, 2002.

[9] D. Gusfield. *Algorithms on Strings, Trees, and Sequences.* Cambridge, 1997.

[10] J. Haitsma, T. Kalker, and J. Oostveen. An efficient database search strategy for audio fingerprinting.

[11] A. Hampapur and R. Bolle. Feature based indexing for media tracking. *Proc. ICME*, 2000.

[12] C. Herley. ARGOS: Automatically Extracting Repeating Objects from Multimedia Streams. *IEEE Trans. Multimedia.*

[13] C. Herley. Extracting repeats from streams. *Proc. ICASSP*, 2004.

[14] J.-L. Hsu, C.-C. Liu, and A. L. P. Chen. Discovering nontrivial repeating patterns in music data. *IEEE Trans. on Multimedia*, 3(3):311–325, 2001.

[15] J. Haitsma and T. Kalker. A highly robust audio fingerprinting system. *Proc. Intl Conf on Music Information Retrieval*, 2002.

[16] H. Jiang, T. Lin, and H.-J. Zhang. Video segmentation with the assistance of audio content analysis. *ICME*, 2000.

[17] S. E. Johnson and P. C. Woodland. A method for direct audio search with applications to indexing and retrieval. *ICASSP*, 2000.

[18] K. Kashino, T. Kurozumi, and H. Murase. A quick search method for audio and video signals based on histogram pruning. *IEEE Trans. on Multimedia*, 5(4):348–357, June 2003.

[19] R. Lienhart, C. Kuhmuench, and W. Effelsberg. On the detection and recognition of television commercials. *Proc. Intl. Conf. on Multimedia Computing and Systems*, pages 509–516, June 1997.

[20] T. Muramoto and M. Sugiyama. Visual and audio segmentation for video streams. *Proc. ICME*, pages 1547–1550, 2000.

[21] G. Pass, R. Zabih, and J. Miller. Comparing images using color coherence vectors. *Proc. ACM Multimedia*, pages 65–73, Nov. 1996.

[22] S. Pfeiffer, S. Fischer, and W. Effelsberg. Automatic audio content analysis. *Proc. ACM Multimedia Conf.*, pages 21–30, 1996.

[23] R. Ragno, C. J. C. Burges, and C. Herley. Inferring Similarity Between Music Objects with Application to Playlist Generation. *ACM Multimedia Information Retrieval*, 2005.

[24] Y. Rui, A. Gupta, and A. Acero. Automatically extracting highlights for tv baseball programs. *Proc. ACM MM*, 2000.

[25] H. Sundaram and S.-F. Chang. Video scene segmentation using video and audio features. *ICME*, 2000.

[26] E. Wold, T. Blum, D. Keislar, and J. Wheaton. Content-based classification, search and retrieval of audio. *IEEE Multimedia*, 3(3):27–36, 1996.

[27] M. Yeung, B.-L. Yeo, and B. Liu. Extracting story units from long programs for video browsing and navigation. *Proc. IEEE Conf. on Multimedia Computing and Systems*, pages 296–305, June 1996.