Jia Zhu and Ye Wang
Department of Computer Science
National University of Singapore
3 Science Drive 2, Singapore 117543
{zhujia, wangye}@comp.nus.edu.sg

Complexity-Scalable Beat Detection with MP3 Audio Bitstreams

With the growing popularity of the MPEG-1 Audio Layer 3 (MP3) format, handheld devices such as personal digital assistants (PDAs) and mobile phones have become important entertainment platforms. Unlike conventional audio equipment, mobile devices are characterized by limited processing power, battery life, and memory, as well as other constraints. Therefore, music-processing tasks like beat detection must be implemented using low-complexity algorithms to cope with the constraints of these mobile devices.

This article presents a scheme of complexity-scalable beat detection of popular ("pop") music recordings that can run on different platforms, particularly battery-powered handheld devices. We show a user-friendly and platform-adaptive scheme such that the detector complexity can be adjusted to match the constraints of the device and user requirements. The proposed algorithm provides both theoretical and practical contributions, because we use the number of Huffman bits from the compressed bitstream without requiring any decoding as the sole feature for onset detection. Furthermore, we provide an efficient and robust graph-based beat-induction algorithm. By applying the beat detector to compressed rather than uncompressed audio, the system execution time can be reduced by almost three orders of magnitude.

We have implemented and tested the algorithm on a PDA platform. Experimental results show that our beat detector offers significant advantages over other existing methods in execution time while maintaining satisfactory detection accuracy.

Motivation

After a decade of explosive growth, mobile devices today have become important entertainment platforms alongside desktop computers and servers. Many applications such as games have been moved to handheld devices, where soundtrack tempo plays a key role in controlling relevant game parameters, such as the speed of the game (Holm et al. 2005). For content-based audio/video synchronization (Denman et al. 2005), the musical beat is the primary information source used as the anchor for timing. The *beat* of a piece of pop music is defined as the sequence of almost equally spaced phenomenal impulses. The beat is the simplest yet fundamental semantic information we perceive when listening to pop music. Groupings and strong/weak relationships form the rhythm and meter of the music (Scheirer 1998).

The beat-tracking process typically organizes musical audio signals into a hierarchical beat structure of three levels: quarter note, half note, and measure (Goto 2001), as shown in Figure 1. Beats at the quarter-note level correspond to periodic "beats" or "pulses" at a simple level, and those at the half-note level and the measure level correspond to the overall "rhythm," which is associated with grouping, hierarchy, and a strong/weak dichotomy. Pop-music beat detection is a subset of the beat-detection problem, which has been solved with detection accuracy as the primary if not the sole objective. In this article, we focus on beat detection in recorded audio rather than real-time beat tracking.

Currently, most beat-detection methods are implemented on a personal computer or server. Based on our experiments, we find that it is difficult to scale down the complexity of existing methods to run on portable platforms such as PDAs and mobile phones, where processing power, memory, and battery life become critical constraints. Although some recent results show that beat tracking can be implemented in a mobile phone after major optimizations (Seppanen et al. 2006), running such a complex algorithm taxes battery life, which is not desirable. Because software applications running on battery-powered portable platforms are gaining popularity, algorithms for content processing such as beat detection must be designed to match both the constraints of the device resources and the users' expectations.

Computer Music Journal, 32:1, pp. 71–87, Spring 2008

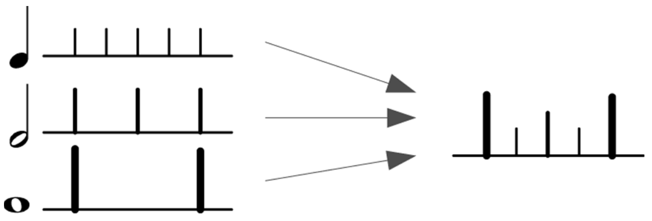


Figure 1. Hierarchical beat structure. (The 4/4 time signature prevalent in popular music is assumed.)

To identify users' requirements, we conducted surveys of students from schools and universities; these students constitute an important segment of the mobile-entertainment market. Our initial survey results indicate that system-execution time, detection accuracy, and battery life are critical performance criteria for mobile-device users. This implies that existing methods, which generally focus on detection accuracy at the cost of computational complexity, are apparently unable to meet users' expectations of mobile platforms. In addition, our survey showed that execution time, defined as the interval between program start and the reception of beat information, should not be more than a few seconds, preferably less than 2 sec. Furthermore, many users complained about having to process music on a desktop platform before beat information could be used on portable devices. Our techniques have been designed with considerations of the tradeoff between users' requirements (e.g., detection accuracy and execution speed) and device resource constraints. We show in this article that the compressed and transform domains are both excellent alternatives to the domain of uncompressed, pulse-code-modulated (PCM) audio, because they allow low complexity and high detection accuracy in beat detection on a mobile platform.

Related Work

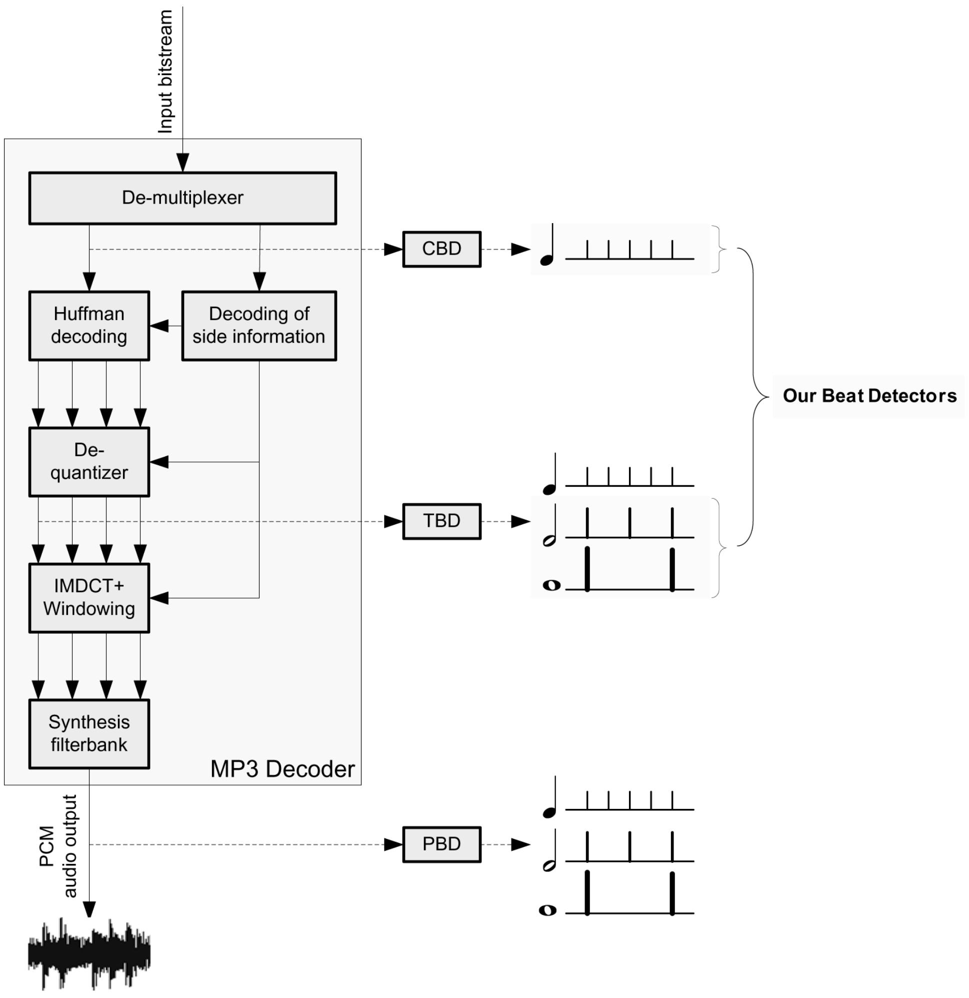
Automatic beat detection has a history of almost two decades; a fairly comprehensive review is given in Gouyon and Dixon (2005). Early beat-detection systems such as those of Povel and Essens (1985), Rosenthal (1992), and Large and Kolen (1994) do not operate on real-world acoustic signals, but rather on symbolic data such as MIDI. Their reliance on MIDI greatly limits their application, because it is not easy to obtain complete MIDI representations of real-world acoustic signals. Such models suffer from the *scaling-up problem* (Kitano 1993). Recent systems work with PCM audio bitstreams. Representative works in this category include Scheirer (1998), Goto (2001), Dixon (2001; 2003), and Shenoy et al. (2004). Some of these algorithms are able to track the whole hierarchical beat structure in pop music. Owing to the popularity of compressed audio formats such as MP3, a beat-detection technique using features from partially decoded compressed audio has been proposed in Wang and Vilermo (2001). Other related works on compressed-domain audio/video processing can be found in Tzanetakis and Cook (2000) and Pfeiffer and Vincent (2001). The work presented in Tzanetakis and Cook (2000) uses *sub-band samples* extracted prior to the synthesize filterbank in an MPEG-2 Audio Layer 3 decoder to calculate features such as spectral centroid, spectral rolloff, and so on, which are used in audio classification and segmentation. To the best of our knowledge, our work is the first to achieve beat detection without decoding; that is, the beat detection is based on features computed directly from the compressed bitstream without even performing entropy decoding. [Editor's note: The initial manuscript was submitted on 12 November 2006.]

System Overview

A diagram of our system is shown in Figure 2. Depending on the decoding level, we have implemented the proposed beat detectors in three domains: the compressed-domain beat detector (CBD), which is the main focus of this article; the transform-domain beat detector (TBD); and the PCM-domain beat detector (PBD). In comparison to existing work, our system allows an automatic selection of beat detector (CBD, TBD, or PBD) based on the availability of computing resources, as well as manual selection by the user. We have implemented our scheme to operate on the MP3 audio format because of its popularity.

Extracting features from PCM audio or transform-domain data has been proposed in previous work

Figure 2. A systematic overview of complexity-scalable beat detectors in three different domains: compressed-domain beat detector (CBD), transform-domain beat detector (TBD), and PCM-domain beat detector (PBD).



(Scheirer 1998; Dixon 2001; Goto 2001). A system presented in Wang and Vilermo (2001) tracks beats at the quarter-note level in the transform domain. However, it has remained unknown whether it is possible to directly detect beats from a compressed bitstream without partial decoding. In this article, we investigate the possibility of detecting the whole hierarchical beat structure.

As with most beat detectors dealing with pop music, we assume that the time signature is 4/4 and the tempo is almost constant across the entire piece of music and roughly between 70 and 160 beats per minute (BPM). Our test data is music from commercial compact discs with a sampling rate of 44.1 kHz.

Compressed-Domain Beat Detection

In an MP3 bitstream, some parameters are readily available without decoding, including *window type, part2_3_length* (Huffman code length), *global gain,* etc. (Wang et al. 2003). Figure 3 shows different features extracted from a compressed bitstream and the corresponding waveform.

Because our objective was to design beat detection for pop music, we selected certain of these parameters on the basis of the following criteria: (1) the feature is well correlated to signal energy; (2) the feature exhibits good self-similarities; (3) the feature depends mainly on the music or the acoustic signals

Figure 3. Extracted compressed-domain data from a pop-music excerpt sampled from a commercial CD: (a) original waveform; (b) window types; (c) part2_3_length*; (d) scale factor bits; (e)* global gain*; and (f) annotated beat times.*

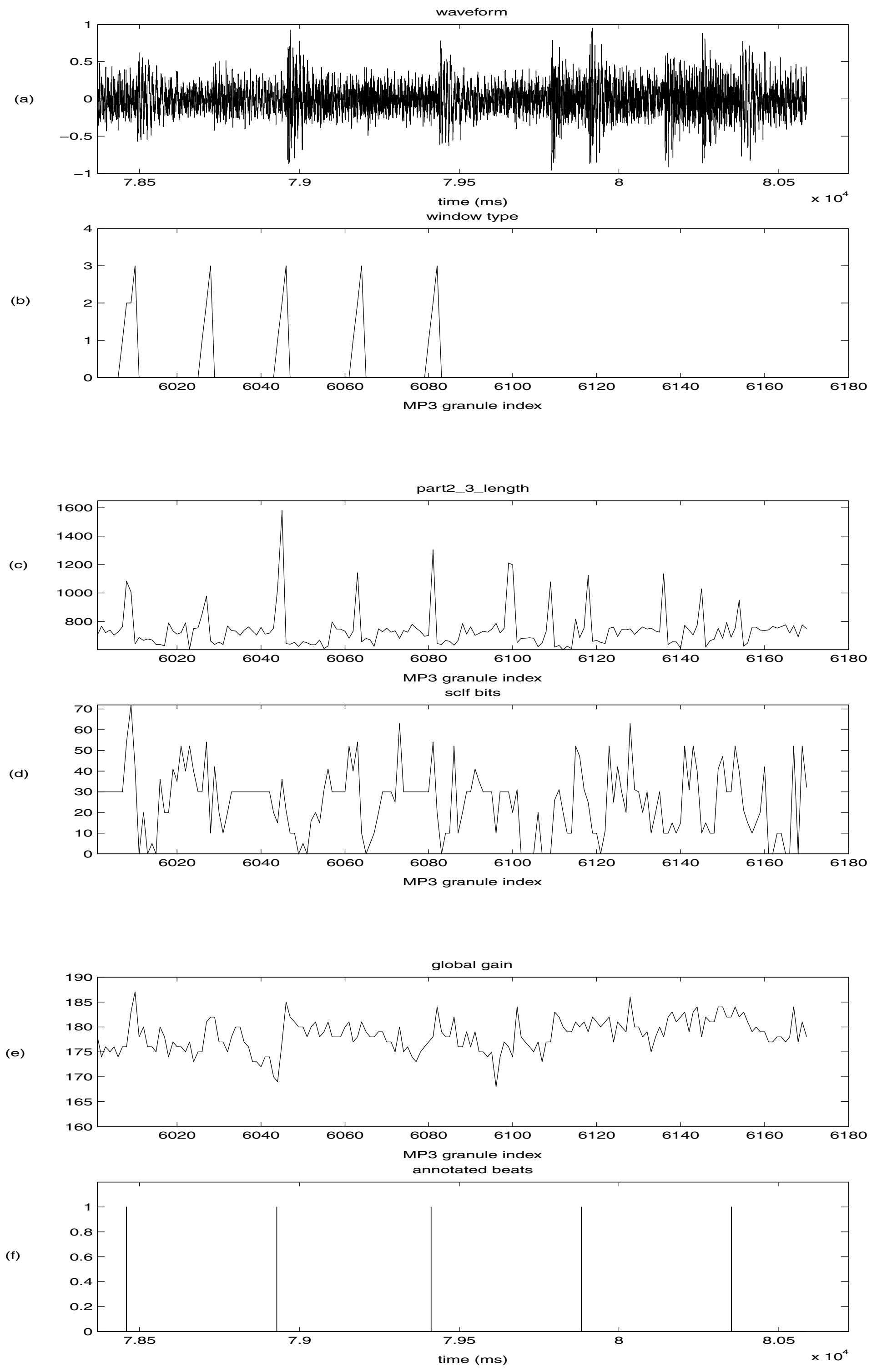
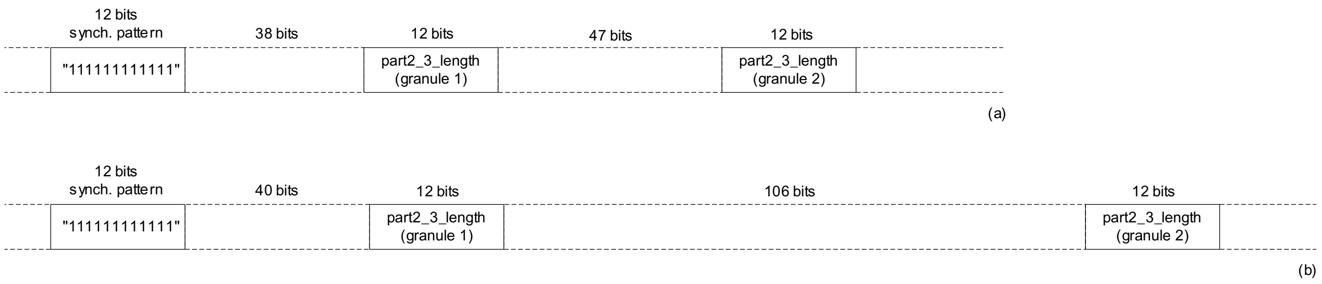


Figure 4. Locations of part2_3_length *in a compressed bitstream for (a) single-channel and (b) dual-channel audio. For dual-channel audio, we extract* part2_3_length *from only the left channel.*



that are compressed, and not on the encoder that has produced the data, which renders *window type* data unsuitable for beat detection, for example; and (4) the feature's MP3 data field has separate values for each granule. (In an MP3 bitstream, the primary temporal unit is a frame, which is further divided into two granules. Some data fields are shared by both granules in an MP3 frame, whereas others have separate values for each granule. We prefer the latter type because it gives a better time resolution.)

In practice, we have used the following quantitative measures for feature selection. For each data type in the compressed domain, we created a sequence s by extracting the value from each granule. Then another sequence b was generated as follows:

$$b_{i \pm k} = 1 \text{ if there is an annotated beat at granule } i, k = \{0,1,2\}$$

$$b_i = 0 \text{ if there is no annotated beat at granule } i \pm k, k = \{0,1,2\}$$

(An annotated beat is one that has been previously specified by a human listener, as explained later.) We calculated the cross-correlations $r_{b,s}$ between b and s at delay 0. Table 1 lists the results of this method for five songs. After checking all the possible parameters in the compressed MP3 bitstream, we found that the *part2_3_length* is well correlated with the onsets and is therefore a good proxy for onset, because it is a high-level indication of the "innovation" or "uniqueness" in each data unit (i.e., granule). The CBD uses *part2_3_length* (see Figure 4) as input data. All beat detectors have two main blocks: onset detection and beat induction, which are presented next.

Transform-domain features are generally more reliable for beat detection than are compressed-domain features, because transform-domain features consist of multi-band data, whereas compressed-domain data seem to reveal only full-band characteristics. In other words, we can achieve better detection accuracy by using multi-band processing with increased complexity. However, if instant results are needed, a single-band approach can offer significantly reduced complexity with reduced detection accuracy.

Table 1. Results of the Cross-Correlation Method

Song No.	Global Gain	Part2_3_ Length	Full-Band Energy
1	0.002	0.228	0.326
2	0.036	0.194	0.253
3	−0.043	0.184	0.184
4	0.004	0.217	0.188
5	−0.009	0.218	0.264
Average	**−0.002**	**0.208**	**0.243**

Onset Detection

The CBD calculates the input data length from *part2_3_length.* Onset candidates are selected using a simple threshold *thr:*

$$thr_i = a \times mean$$

where i is the granule index, and a is an empirically determined constant value. During the system evaluation, we noted that the beat-detection accuracy is not particularly sensitive to the choice of a, because the proposed beat-induction algorithm is robust to the inaccuracy of onset detector. The window for calculation is $\{i - 34, i + 34\}$. Thus, the window size is 69 granules, which corresponds to approximately 900 msec. The selected window size is the same as the one used in Wang et al. (2003) for onset detection. Granule i is considered to contain an onset if the following conditions are met:

$$\begin{cases} f_i > thr_i \text{ (condition 1)} \\ f_i > f_{i \pm k} \text{ (condition 2)} \end{cases}$$

where f_i is the ith feature obtained from half-wave rectification, and $k \in \{1 \ldots 17\}$. Condition 2 ensures that any two onsets are at least two granules (approximately 26 msec) apart from each other. This implies at most one onset can be detected within any period of 50 msec. We denote this property as *onset property* and use it in beat induction.

It should be noted that the onset detector is selected mainly for its simplicity and for the characteristics of the feature. Many of the methods in Bello et al. (2005) are simply not applicable to compressed-domain features.

Beat Induction

The beat-induction process determines beat times based on onset times from the previous step. Our beat-induction algorithm is designed to be robust enough to work with input onsets that have low accuracy. Unlike the onsets detected from a PCM bitstream, features extracted from a compressed bitstream are generally much noisier.

We use a data structure called an *Ordered Event Set,* which is composed of an *ordered set* of distinct events, denoted $(S, \leq_R)$, to store onsets or beats. Two events are distinct if and only if they do not occur simultaneously. The relation $\leq_R$ is defined as follows: $i \leq_R j$ if and only if event i occurs earlier than or at the same time as event j. It is obvious that relation $\leq_R$ is anti-symmetric and transitive. An *ordered pair* (i, j) of an ordered event set ES satisfies $i, j \in ES \wedge i \leq_R j \wedge i \neq j$. A pair (i, j) of ES is a *consecutive pair* if (i, j) is an ordered pair and there is no element e such that (i, e) and (e, j) are ordered pairs of ES. The *difference* of an ordered pair (i, j), denoted by $diff(i, j)$, is the absolute value of the time difference between the occurrence of event i and that of event j.

Because elements in ES are distinct and ordered, we can get the rank of an element e with the operation $rank(ES, e)$; this function returns the rank of e if $e \in ES$, and -1 otherwise. If e is the *head* of ES, that is, $e = head(ES)$, then $rank(ES, e)$ returns 1; if e is the *tail of ES*, that is, $e = tail(ES)$, then $rank(ES, e)$ returns the size of ES. A reverse operation *get* returns the element given a rank, namely, $get(ES, rank(ES, e)) = e$ if $e \in ES$. $Succ(ES, e)$ returns the successive element of e in ES. We formulate the beat-induction problem as shown in Table 2.

Table 2. Formulation of the Beat-Induction Problem

Input:	An ordered event set O
Output:	A pair (d, B) that satisfies three conditions
Condition 1:	d is a real number and $Q_{MIN} \leq d \leq Q_{MAX}$, where Q_{MIN} and Q_{MAX} are constants; B is an ordered event set
Condition 2:	For every consecutive pair (i, j) of B, $diff(i, j) \in [d - \varepsilon, d + \varepsilon]$
Condition 3:	For any pair (d', B') that satisfies conditions 1 and 2 and is not identical to (d, B), $\lvert O \cap B' \rvert < \lvert O \cap B \rvert$

Intuitively, the input set O contains all the detected onsets of a piece of music, the output value d is the anticipated quarter-note length, and the output set B contains all the beats. Q_{MIN} and Q_{MAX} are the smallest and largest possible quarter-note lengths allowed by the algorithm, respectively. In our current implementation, $Q_{MIN} =$ 375 msec, and $Q_{MAX} = 923$ msec, which correspond to tempi ranging from 65 to 160 BPM. The deviation, ε, is set to 25 msec. Because we work with MP3 granules instead of units of msec in the compressed domain, the corresponding parameters in the compressed domain (for a sampling rate of 44.1 KHz) are $Q_{MIN} = 28$ granules, $Q_{MAX} = 72$ granules, and $\varepsilon = 2$ granules.

Next, we introduce another data structure called a *pattern.* A pattern is defined to be an ordered event set with an associated pair (s, d). A pattern P meets the following conditions: (1) $P \subseteq O$, where O is the *ordered* event set containing all the onsets; (2) $\lvert P \rvert \geq 1$ and $head(P) = s$; (3) for every consecutive pair (i, j) of P, if there is any, $diff(i, j) \in \{d - \varepsilon, d + \varepsilon\}$; and (4) there does not exist another ordered event set S

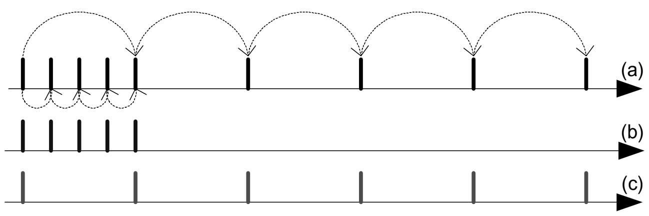


Figure 5. Two patterns can be identified from the onsets on axis (a) and are denoted on axis (b) and axis (c).

such that $P \subset S$, and S also meets conditions 1, 2, and 3.

Figure 5 provides an intuitive illustration of a *pattern.* We claim that the associated pair (s, d) of a pattern uniquely identifies the specific pattern. This can be proved as follows. Suppose there are two patterns $P1$ and $P2$ with the same associated pair (s, d). Then $head(P1) = head(P2) = s$, according to condition 2. Because there is at most one onset within the interval $[t - \varepsilon, t + \varepsilon]$, where t is arbitrary, according to the onset property, we have $diff(s, x) \in [d - \varepsilon, d + \varepsilon] \wedge diff(s, y) \in [d - \varepsilon, d + \varepsilon] \rightarrow x = y$, which implies that the second element of $P1$ is identical to that of $P2$ according to condition 3.

If $|P1| = |P2|$, then using the same argument inductively for the rest of the elements in $P1$ and $P2$, we can infer that all of them are identical, that is, $get(P1, k)$ is identical to $get(P2, k)$ for $k \in \{1, 2, \ldots, |P1|\}$, and thus $P1$ and $P2$ have the same pattern. If $|P1| \neq |P2|$, we can assume $|P1| < |P2|$ without loss of generality. Then $get(P1, k)$ is identical to $get(P2, k)$ for $k \in \{1, 2, \ldots, |P1|\}$. This implies that $P1 \subset P2$, which contradicts condition 4. Hence, a pattern can be uniquely identified by its associated pair. If a pattern P has an associated pair (s, d), we denote d as the *lapse* of P, that is, $lapse(P) = d.$ The procedure for extracting the pattern given the associated pair (s, d) is straightforward. The initial status of the *pattern P* is $\{s\}$. For each onset o, if $diff(tail(P), o) \in [d - \varepsilon, d + \varepsilon]$, we add o into P, that is, $P \leftarrow P \cup \{o\}$.

The beat-induction algorithm begins by detecting the anticipated quarter-note length (QNL). The procedure is an inter-onset interval, histogram-based method, commonly used in beat detectors like those described by Gouyon et al. (2006). We improve these methods with emphasis on speed and tolerance of inaccurate onsets. To achieve prompt detection of the anticipated QNL, we carry out the histogram method in two stages. The first stage detects a coarse QNL, and the second stage detects a fine QNL. In the first stage, we use nine bins that cover the interval $[Q_{MIN}, Q_{MAX}]$, each of which spans five granules. After the normal histogram procedure, the center of the bin with the maximum number of elements is taken as the coarse QNL, *cqnl.* In the second stage, we only consider inter-onset intervals in the range $[cqnl - 2, cqnl + 2]$. We use five bins, each of which spans one granule, and then perform the histogram procedure again. The granule index represented by the bin with the maximum number of elements is taken as the fine QNL. An example of the histogram method is shown in Figure 6.

To further speed up this procedure, we can use just a small segment, for example, the first half-minute, of the whole song as an input to the histogram. However, we did not use this method in our experiment, because it might fail if there are large gaps between successive onsets over the whole song. Furthermore, experimental results have shown that our two-stage histogram method is fast enough.

After the quarter-note length is detected, the next step is to compute beat times based on the quarter note length *qnl.* Our objective is to create an ordered event set B such that for every consecutive pair (i, j) of B, $diff(i, j) \in [qnl - \varepsilon, qnl + \varepsilon]$, and $|B \cap O|$ is maximum. To solve this problem, we propose a graph-based approach. We first introduce the concept of *compatibility.*

A pattern A is defined to be *compatible* with pattern B with lapse d $(d > \varepsilon)$ if and only if the following condition holds:

$$tail(B) \leq_R head(A),\ \ lapse(A) = lapse(B) = d,\ \ and$$

$$\frac{diff(tail(B), head(A))}{ROUND(diff(tail(B), head(A)) / d)} \in [d - \varepsilon, d + \varepsilon].$$

Here, $ROUND$ is an operation that rounds its parameter to the nearest integer. If A is *compatible* with B *with lapse d*, we denote $A \underset{\rightarrow c}{\overset{d}{}} B.$ The compatibility relation satisfies the following property:

$$A \underset{\rightarrow c}{\overset{d}{}} B \wedge B \underset{\rightarrow c}{\overset{d}{}} A \text{ never holds}$$

This property can be proved using contradiction; the proof is straightforward and is hence omitted here. Figure 7 gives an example of compatibility.

The graph-based approach starts with the collection

Figure 6. The two-stage histogram method is carried out in the compressed domain and in the PCM domain, respectively, with the same input song. In the PCM domain, the first histogram has 10 bins, with a resolution of 50 msec, and the second histogram has 50 bins, with a resolution of 1 msec. The quarter-note length detected in the compressed domain is 54 granules (707.4 msec), whereas that in the PCM domain is 709 msec.

Figure 7. Pattern II is compatible with pattern I. Neither pattern I nor pattern II is compatible with pattern III.

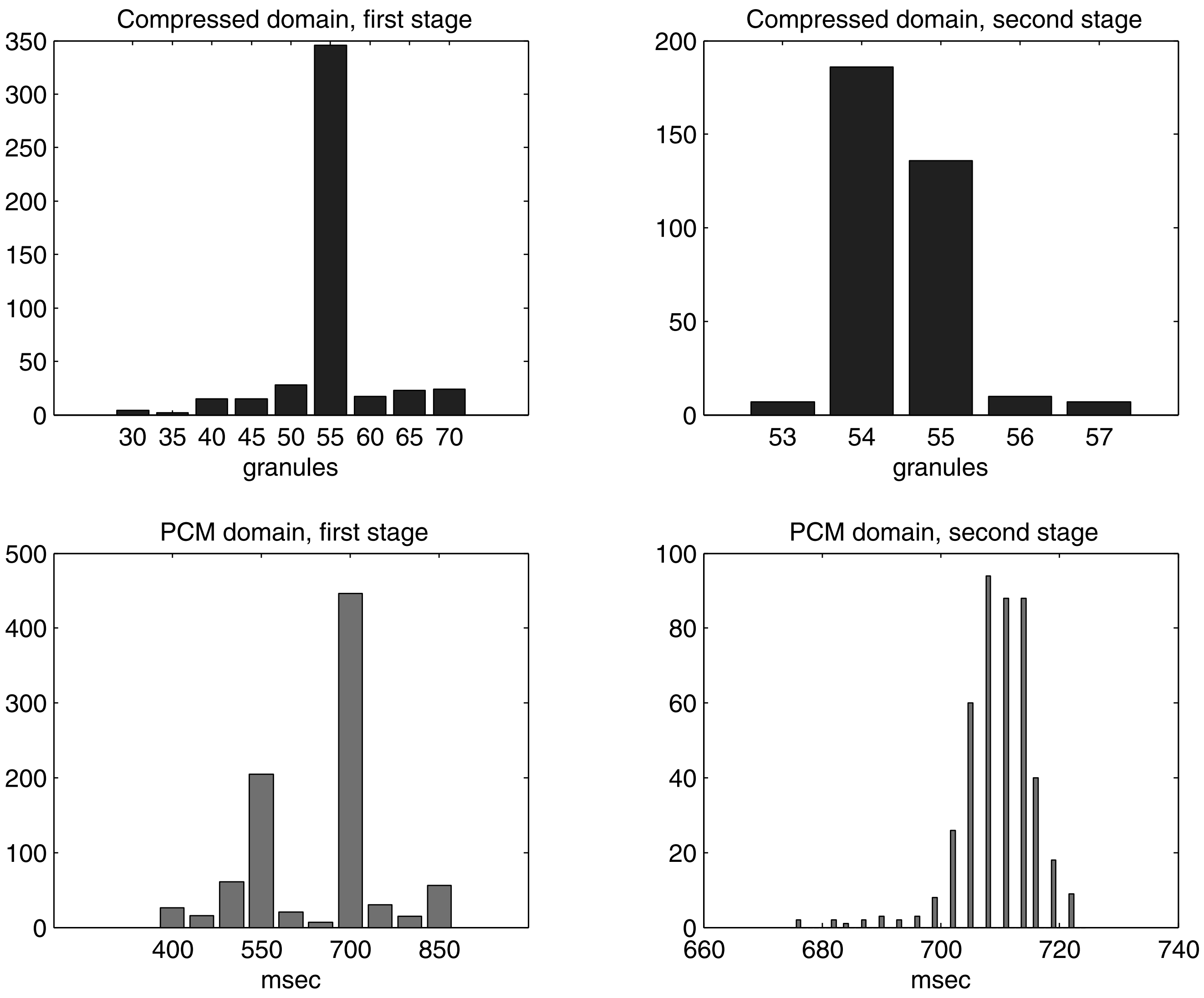


Figure 6

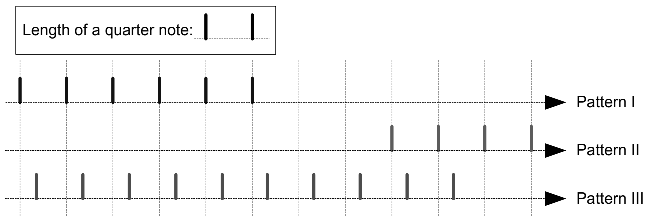


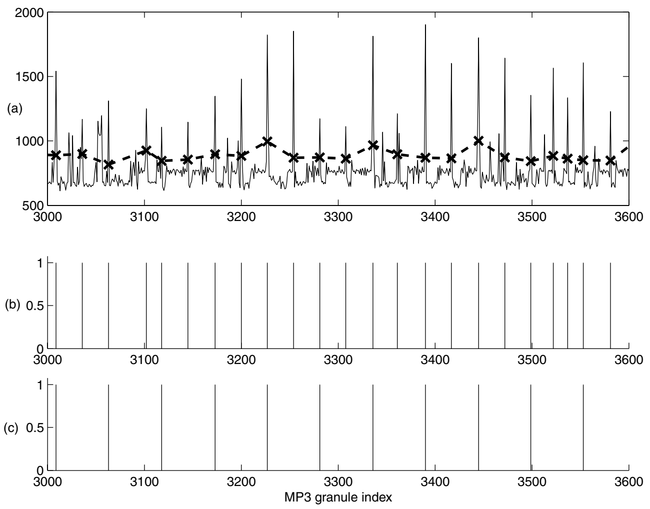
Figure 7

of all patterns with lapse *qnl* from the onsets, where *qnl* is the quarter-note length. The procedure shown below extracts all patterns with a prescribed lapse by a single iteration through the ordered set of all onsets. In that procedure, we use another ordered event set $(L, \leq_{R'})$, which has the same properties and operations as $(S, \leq_R)$ as the data structure to store all the patterns. The relation $\leq_{R'}$ is defined by $L_i \leq_{R'} L_j$ if and only if $head(L_i) \leq_R head(L_j)$. The algorithm is shown in Figure 8.

Figure 8. Procedure for collecting patterns.

```
Procedure: CollectAllPatterns(O, qnl)
Input: The ordered event set O containing all the
onsets, and the detected quarter-note length qnl.
Output: An ordered event set L containing all the
patterns with lapse qnl.
L ← ∅
Initialize a flag array F of the same size as O, with
all elements being 0
for each element e’ in O
     e ← e’
     if F[rank(O, e)] = 0
         then initialize a new empty pattern P
               P ← P ∪ {e}
               F[rank(O, e)] ← 1
               es ← succ(O, e)
               while diff(es, tail(O)) > 0
                    do if diff(es, e) ∈ [qnl – ε, qnl + ε]
                      then P ← P  ∪ {es}
                         F[rank(O, es)] ← 1
                         e ← es
                    if diff(es, e) > qnl + ε
                         then break
                    es ← succ(O, es)
               L ← L ∪ {P}
```

Figure 9. (a) Part2_3_length *(solid line) and threshold (dashed line); (b) detected onsets; (c) detected beats after beat induction.*



After collecting the patterns, we create a compatibility matrix *CM* with dimension |L| × |L| as follows:

$$CM[i][j] = \begin{cases} 1 & if\ get(L,i) \xrightarrow[c]{qnl} get(L,j); \\ 0 & otherwise, \end{cases} \text{, for any } 1 \le i,j \le |L|$$

CM can be viewed as the adjacent matrix of a graph $G = (V, E)$, where $V[G] = \{x \mid x \in Z \wedge x \ge 0 \wedge \exists p, x = rank(L, p)\}$, $E[G] = \{(j, k) \mid j, k \in V[G] \wedge CM[j, k] = 1\}$. By the compatibility property, the graph is directed and acyclic: $(i, j) \in E[G]$ if and only if $get(L, i) \xrightarrow[c]{qnl} get(L, j)$.

The problem is transformed to finding a path $p = \{v_0, v_1, \ldots, v_k\}$, where $v_0, v_1, \ldots, v_k \in V[G]$, such that $\Sigma_{i=0}^{k}\ pattern\_count(get(L, v_i))$ is maximized. To solve the problem, we first convert graph *G* into another directed acyclic but weighted graph $G' = (V, E)$, on which we can apply the Bellman-Ford algorithm. The new graph *G′* is obtained by adding a dummy vertex $dummy = |V[G]| + 1$ to the vertex set of *G*, and creating edges from the dummy vertex to every other vertex in *G′*. Thus, $V[G'] = V[G] \cup \{dummy\}$, and $E[G'] = E[G] \cup \{(dummy, k) \mid k \in V[G]\}$. The weight of an edge (j, k) in *G′*, denoted by $w(j, k)$, is assigned by $pattern\_count(get(L, k))$. The negation allows us to apply the Bellman-Ford algorithm, which finds the path that originates from the dummy vertex with minimal total weights instead of maximum total weights. Based on the output path of the Bellman-Ford algorithm, we collect the patterns represented by the vertices on the path and store the elements of those patterns in an ordered event set *B*. Then *B* contains partial beats.

The next step is to obtain the complete beats. The rest of the beats are interpolated based on the partial beats in *B*. Interpolation is done as follows. For every consecutive pair (x, y) in *B*, if $diff(x, y) \notin [qnl - \varepsilon, qnl + \varepsilon]$, then *x* and *y* do not appear in the same pattern; *x* is the tail of one pattern *P*1, and *y* is the head of another pattern *P*2. We can also infer *P*2 is compatible with *P*1 with lapse *qnl*. Based on the definition of *compatibility,* we have

$$\frac{diff(x,y)}{ROUND(diff(x,y)\,/\,qnl)} \in [qnl - \varepsilon, qnl + \varepsilon]$$

Therefore, if we insert $k = (ROUND(diff(x, y) / qnl) - 1)$ number of beats $b_1, b_2, \ldots, b_k$ between *x* and *y* such that $diff(x, b_1) = diff(b_1, b_2) = \ldots = diff(b_k, y) = d$, we can infer that $d \in [qnl - \varepsilon, qnl + \varepsilon]$. This will ensure that the tempo is maintained across the interpolated beats.

The worst-case running time of our beat-induction algorithm is $O(n_1^3)$, where n_1 is the total number of detected onsets. However, in practice, the algorithm usually performs much faster than $O(n_1^3)$. The actual running time is $\max(O(n_1^2), O(n_2^3))$, where n_2 is the total number of patterns, because the Bellman-Ford algorithm has a cubic time complexity. Because $n_1 >> n_2$ in almost all cases, and $n_1^3 >> n_1^2$ when n_1 is large, it follows that $\max(O(n_1^2), O(n_2^3)) << O(n_1^3)$. Hence, the actual running time is much less than $O(n_1^3)$. The memory consumption of our beat-induction algorithm is $\max(O(n_1), O(n_2^2))$. We use a bit array to implement the compatibility matrix. A 16-bit integer is used to represent each onset. (Note that in the compressed domain, we work with MP3 granule indices, which can be represented as 16-bit integers.) Thus, the hidden constant in the Big-O notation of memory consumption is small.

Our onset detection and beat induction are illustrated in Figure 9.

Transform/PCM-Domain Beat Detection

Both TBD and PBD have three general steps: onset detection, beat induction, and bar detection. The first two of these steps are analogous to the corresponding steps of CBD, which does not include bar detection. The onset detector is different in each of these three domains, although the onset detectors for TBD and PBD are similar. In comparison with the onset detector for TBD, the onset detector for PBD requires an additional fast Fourier transform (FFT) operation for frequency analysis, which is detailed in Shenoy et al. (2004). We use the same beat-induction algorithm for beat detectors in all three domains. The onset detection and bar detection for TBD are discussed in this section.

Onset Detection

The onset detector for TBD uses the threshold-by-band method. It first divides the modified discrete cosine transform (MDCT) frequency lines into four sub-bands. The division for long windows is 1–3, 4–25, 26–85, and 86–576. (These numbers indicate the indices of MDCT frequency lines.) The corresponding frequency intervals are thus 0–115 Hz, 116–957 Hz, 958–3,254 Hz, and 3,255–22,050 Hz. For short windows, we try to match the frequency intervals with those for long windows as closely as possible. The division for short windows is 1, 2–9, 10–29, and 30–192, corresponding to frequency intervals of 0–114 Hz, 115–1,033 Hz, 1,034–3,330 Hz, and 3,331–22,050 Hz. This approach is similar to that described by Wang and Vilermo (2001); however, unlike that approach, we employ all sub-band information.

Next, energy from each band is calculated for each granule. The energy $E_b[n]$ of band b (b = 1, 2, 3, or 4) in granule n is calculated as

$$E_b[n] = \begin{cases} \sum_{j=N_1}^{N_2} \left(X_j[n]\right)^2 \\ \sum_{a=1}^{3} \left(\sum_{j=N_1}^{N_2} \left(X_{a,j}[n]\right)^2 \right) \end{cases}$$

where the first relation applies to granules that contain a long window, and the second relation applies to granules that contain short windows. Also, $X_j[n]$ is the jth MDCT coefficient decoded at granule n (when granule n contains a long window), $X_{a,j}[n]$ is the jth MDCT coefficient decoded in the ath short window of granule n (when granule n contains three short windows), N_1 is the lower bound index, and N_2 is the upper bound index of band b. Full-band energy is calculated by adding all the sub-band energies for each granule.

Energy values of the four sub-bands and the full-band form five vectors of features. We carry out a procedure similar to that described in Wang et al. (2003) on the five vectors of features to detect onsets. The procedure chooses onset candidates from each feature vector using a threshold-based method, and the onset candidates from the five feature vectors are converged using a weighted-average method.

Note that the onsets detected by this method, like those detected by CBD, have the onset property, which renders them valid as input to the beat-induction algorithm presented earlier.

Bar Detection

Our bar-detection algorithm uses the idea of detecting chord changes, similar to the algorithm described in Goto (2001), which detects bar information in the PCM domain. We have modified that algorithm to work in the transform domain. Our TBD calculates chord-change probabilities at each quarter-note boundary. The calculation of chord-change probabilities at each eighth-note boundary is omitted in our implementation. A histogram is formed by

$$H(n,f) = \sum_{i=q(n)+gap(n)}^{q(n+1)-gap(n)} (X_f[i])^2$$

where $X_f[i]$ is the fth MDCT coefficient decoded at granule i, $q(n)$ is the granule index mapped from the nth beat time, $q(n + 1)$ is the granule index mapped from beat time $(n + 1)$, and $gap(n) = (q(n + 1) - q(n))/5$.

We consider only the frequency range of 1–1,000 Hz, which is supposed to contain the frequencies of dominant tones (Goto 2001). Thus, only the first 27 MDCT frequency lines for long windows and the first nine MDCT frequency lines for short windows are used to create the histogram.

To solve the mismatch of different frequency resolutions between long and short windows, a compromise method is applied, as follows. Because there are three windows in a granule of short window type, we pick the first nine MDCT frequency lines in each of the three windows, and we order them as follows:

$$X[3 \cdot (n-1) + a] = w_a[n], a \in \{1,2,3\} \text{ and } 1 \le n \le 9$$

where $w_a[n]$ is the nth MDCT frequency line in short window a in one granule. The ordered frequency lines constitute 27 lines, which are used in our histogram calculation in the same way as the first 27 frequency lines are in a long window.

After calculating the histogram, we follow the same procedure described in Goto (2001) to calculate the chord-change probabilities at each beat time. The chord-change probabilities are used to infer bar boundaries. In particular, we calculate four values, S_1, S_2, S_3, and S_4:

$$S_i = \sum_{k=0}^{bn/4-1} T(4 \cdot k + i), i \in \{1,2,3,4\}$$

In the above equation, bn is the total number of beats, and function T is defined recursively as

$$T(n) = \begin{cases} W1 \cdot T(n-4) + W2 \cdot C(n) & n > 4 \\ 0 & otherwise \end{cases}$$

where the $C(n)$ are the chord-change probabilities calculated at beat n, and $W1$ and $W2$ are two constants. Suppose ix is an integer such that $ix = \arg_{1 \le i \le 4} \max(S_i)$; then beat $4k + ix$ marks the start of bar $(k + 1)$, where $k \in \{0, 1, 2, \ldots, bn/4-1\}$.

Evaluation

We use libmad, a highly optimized, open-source MP3 decoder, for our system implementation and evaluation. We carefully selected 25 pop songs to provide sufficient sampling variety, and we encoded each song at a bit rate of 128 kbps. Pop-music beat detection in the PCM domain is a relatively straightforward task; we investigated the performance degradation of the TBD and CBT relative to our PBD baseline (Shenoy et al. 2004), which can detect beats in the selected 25 songs correctly.

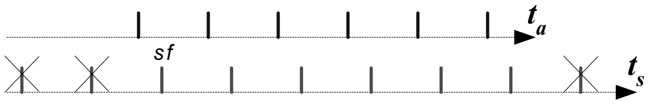


Figure 10. In this example, the first two beat times and the last beat time in t_s *are removed so that* t_s *is matched with* t_a.

Evaluation Method

The test music for the all three detectors—CBD, TBD, and PBD—is identical and is all sampled from commercial CDs. Three music students from our university manually annotated beat times. They first worked individually on all the test samples, and then the individual annotations were averaged to get the final annotations. The annotated beat times and system-generated beat times were sent to an evaluator program. The evaluator program used a variation of the evaluation method proposed in Goto and Muraoka (1997), which we briefly summarize as follows.

A system-generated beat time sequence is denoted as t_s, and an annotated beat-time sequence is denoted as t_a. Before we calculate the normalized deviation at each detected beat, we carry out the following procedure to match t_s with t_a. First, we find in t_s the element sf that is closest to the first element of t_a. Suppose the index of sf in t_s is τ, the length of t_a is l_a, and that of t_s is l_s. We remove the first $\tau - 1$ elements and the last $l_s - l_a - \tau + 1$ elements from t_s. Figure 10 gives a simple example of this procedure.

The normalized deviation at detected beat n, $d(n)$, is calculated as

$$d[n] = \begin{cases} \dfrac{2 \cdot (Ts[n] - Ta[n])}{Ta[n+1] - Ta[n]} & if \;\; Ts[n] \ge Ta[n] \\ \dfrac{2 \cdot (Ta[n] - Ts[n])}{Ta[n] - Ta[n-1]} & if \;\; Ts[n] < Ta[n] \end{cases}$$

The mean α and standard deviation β of the sequence formed by $d[2], \ldots, d[size - 1]$, where $size$ is the size of sequence Ta, are then calculated. We also calculate

$$\gamma = \max_{1<i<size}(d[i])$$

We accept Ts as a correct beat sequence if $\alpha < 0.1$, $\beta < 0.15$, and $\gamma < 0.5$.

For TBD, the correctness of detected bars is also examined. If the detected quarter-note information

Table 3. Experimental Results

Song Title	*Artist*	*CBD*	*TBD*		
		♩	♩	𝅗𝅥	𝅝
Back to You	Bryan Adams	×	√	√	×
Breathless	The Corrs	√	√	√	√
Burn	Tina Arena	√	√	√	√
Crush	Jennifer Paige	√	√	√	√
Drops of Jupiter	Train	√	√	√	√
Heal the World	Michael Jackson	√	√	√	√
I Can't Tell You Why	Eagles	×	√	√	√
It Must Have Been Love	Roxette	√	√	√	√
I Want to Know What Love Is	Foreigner	√	√	√	√
Losing My Religion	R.E.M.	√	√	√	√
Mmmbop	Hanson	√	√	×	×
One	U2	√	×	×	×
One of Us	Joan Osborne	√	√	×	×
Road to Hell	Chris Rea	√	√	√	√
Seasons in the Sun	Westlife	√	√	√	√
Smooth	Santana	√	√	√	√
Someday	Michael Learns To Rock	√	√	√	√
Stayin' Alive	Bee Gees	√	√	√	√
The Way It Is	Bruce Hornsby	√	√	√	×
Time of Your Life	Green Day	×	×	×	×
I Knew I Loved You	Savage Garden	√	√	√	×
Viva Forever	Spice Girls	√	√	×	×
Walking Away	Craig David	√	√	√	√
Whenever, Wherever	Shakira	×	√	√	√
You Make Loving Fun	Fleetwood Mac	√	√	×	×
Number of songs tracked, from a total of 25		21	23	19	16

fails in the evaluation, then the detected half notes and bars are all rejected; otherwise, we find in sequence *Ta* a beat *b*1 that marks the start of a bar and in sequence *Ts*, as well as a beat *b*2 that also marks the start of a bar. Suppose the index of *b*1 in *Ta* is i_1, and the index of *b*2 in *Ts* is i_2. If $(i_1 - i_2)$ modulo 4 is 0, we accept the detected half notes and bars; otherwise, if $(i_1 - i_2)$ modulo 4 is 2, we accept the detected half notes and reject the detected bars; if not, both the detected half notes and bars are rejected.

Detection Accuracy

The evaluation results are listed in Table 3. Figure 11 shows the average performance with respect to detection accuracy and the corresponding execution time.

Execution Time

The three beat detectors were implemented on an HP iPAQ hx4700 PDA running Microsoft Windows Mobile 2003 SE. (The iPAQ hx4700 uses the Intel PXA270 processor with a clock speed of 624 MHz and has 64 MB of SDRAM and 128 MB of ROM.) Owing to the low quality of compressed-domain features, the proposed beat detector must be performed offline in the compressed domain. The average execution times in the three domains are presented in Figure 12. We normalize the execution

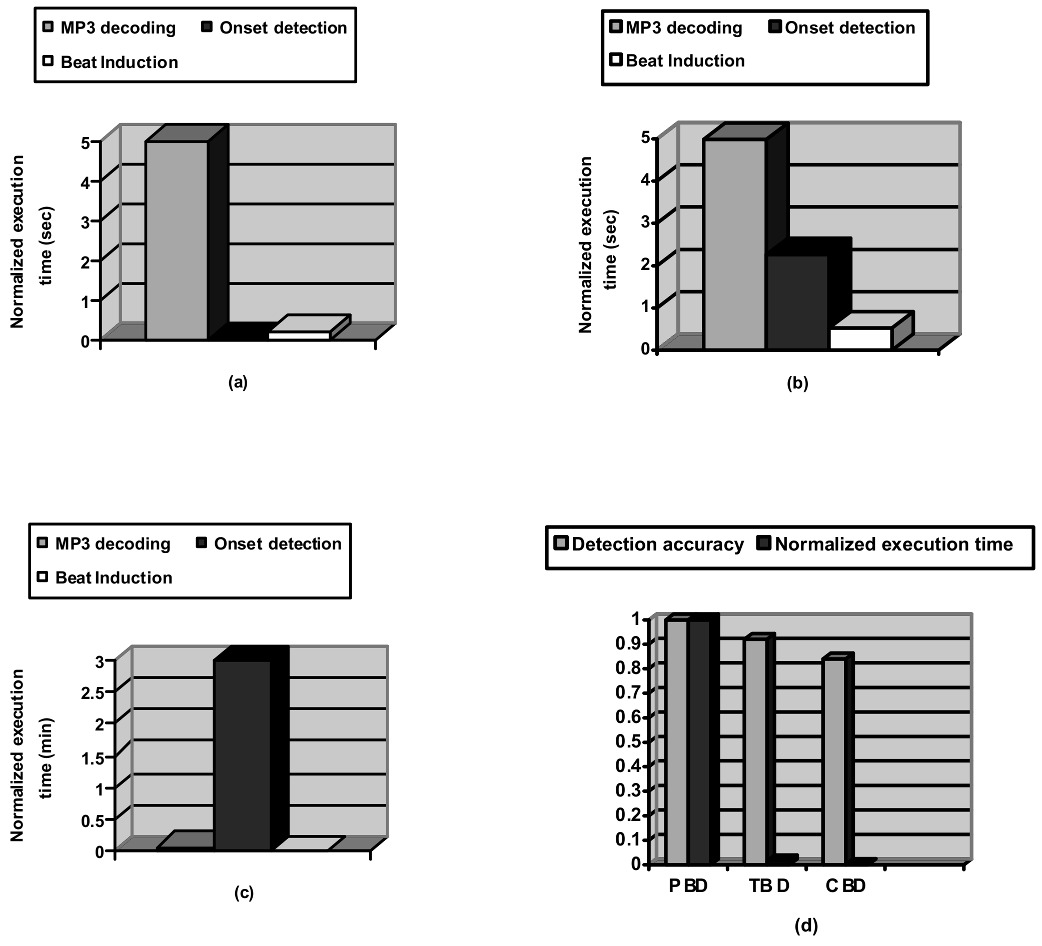


Figure 11. Performance comparison: execution time of (a) CBD, (b) TBD, and (c) PBD as compared to MP3 decoding time; (d) detection accuracy versus execution time in the three domains.

time by dividing the actual execution time by the duration of the input song (in minutes).

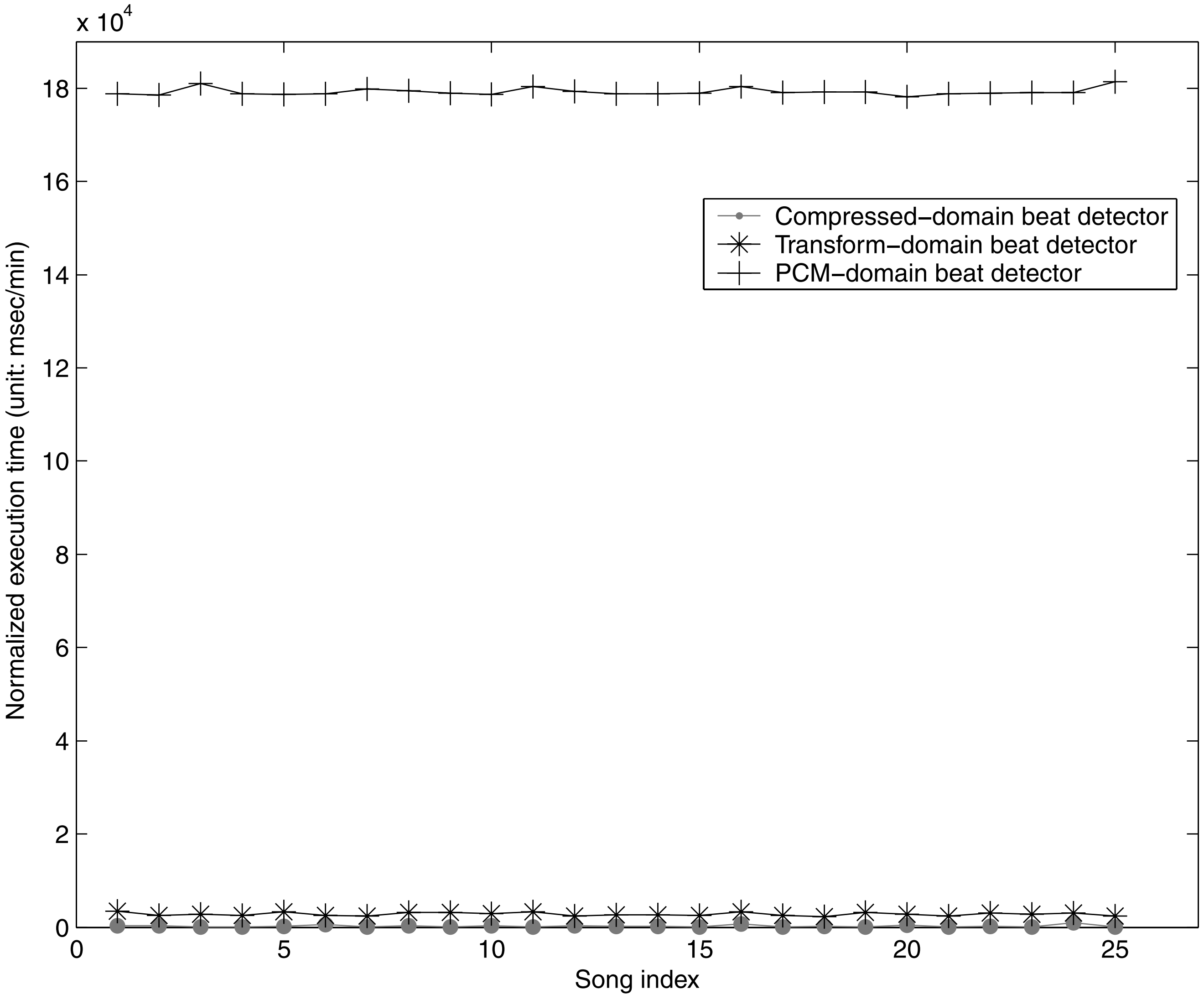
The experimental results show that beat induction takes roughly the same amount of time in the three operation domains. The main difference lies in the onset detection, which is the dominant factor that causes the vast difference between CBD and PDB in terms of execution time. The execution time of CBD is negligible in comparison to MP3 decoding. The execution time of TBD is comparable to MP3 decoding. PBD requires a significantly longer execution time compared to MP3 decoding, mainly due to an extra time-frequency transform.

In summary, the average duration of the 25 test songs is about 4 minutes. The average decoding time per song from MP3 to PCM is about 21 seconds. The average beat detection time is about 1 second for CBD, 12 seconds for TBD, and 13 minutes for PBD. These results show that the compressed- or transform-domain processing provides a significant advantage for mobile platforms, whereas PBD is more suitable for desktop or server platforms.

Applicability to Other Formats

To evaluate dependency on the input audio format, we also implemented the proposed algorithm with the Advanced Audio Coding (AAC) decoder at a constant bit rate of 128 kbps. The detection performance is significantly lower than that with MP3.

Figure 12. Normalized execution time for each song by the three detectors.

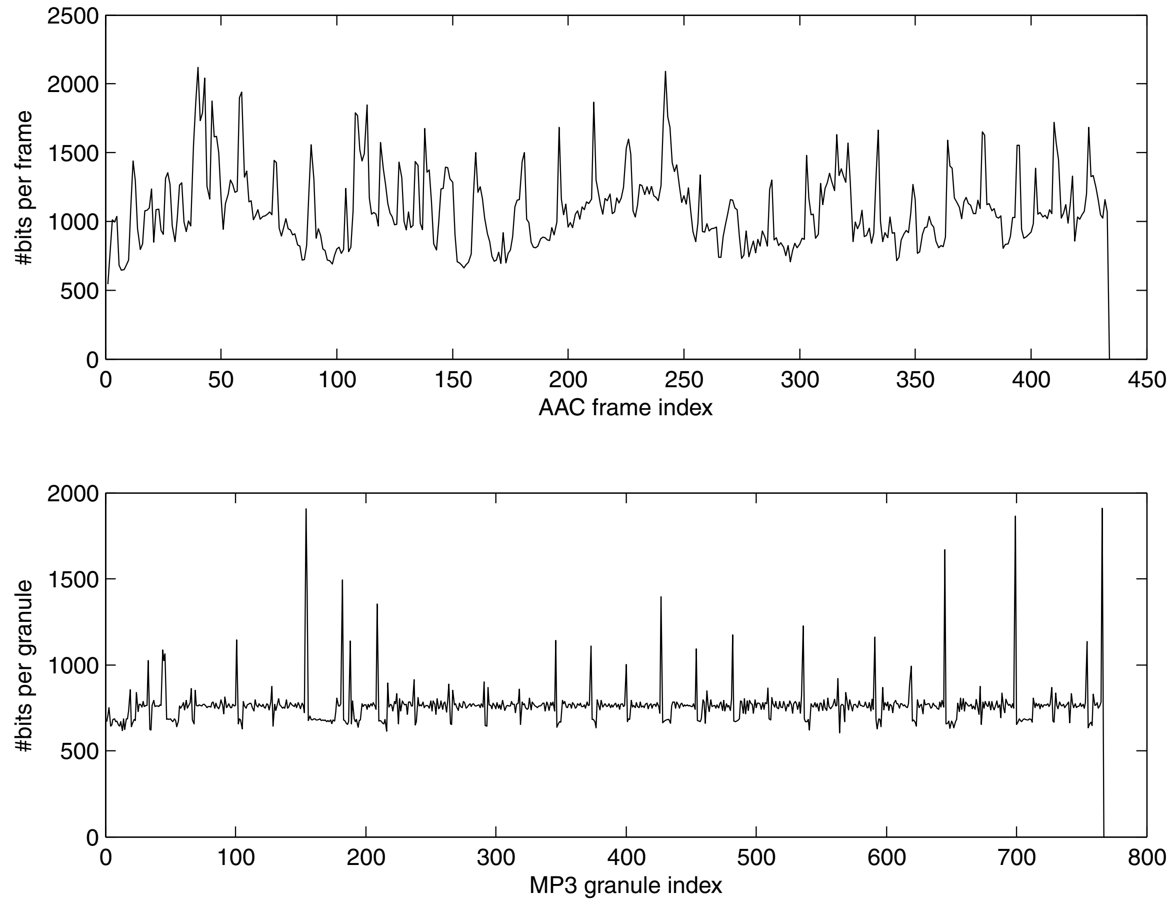


Most of the errors with AAC bitstreams are π-errors (Goto and Muraoka 1997). We believe that the main reason for the difference is that the time resolution of AAC is much lower, which results in a lower feature quality. The difference is illustrated in Figure 13. This implies that the proposed method may not be directly applicable to other audio formats. Given the popularity of MP3, this is not overly restrictive. It will be interesting to investigate how sensitive the algorithm is to the bitrate of MP3 files.

Concluding Remarks

We have presented a complexity-scalable beat-detection method that considers user expectations and the resource constraints of mobile devices. The algorithm was implemented and tested on a targeted PDA platform. Experimental results show that the compressed- and transform-domain processing are particularly suitable for mobile applications, providing a satisfactory tradeoff between detection accuracy and execution speed.

Figure 13. Compressed-domain feature comparison between MP3 and AAC.



Because the TBD can provide a good tradeoff between detection accuracy (comparable to the PBD) and execution speed (comparable to the CBD), we are working on optimizing the TBD to make it more suitable for mobile devices. In the future, we plan to transport our beat detectors to different hardware (e.g., mobile phones) and software platforms (e.g., Symbian). Another avenue of future work is to design algorithms by taking into account the constraints of power consumption of mobile platforms.

References

Bello, J. P., et al. 2005. "A Tutorial on Onset Detection in Music Signals." *IEEE Transactions on Speech and Audio Processing* 13(5):1035–1047.

Denman, H., et al. 2005. "Exploiting Temporal Discontinuities for Event Detection and Manipulation in Video Streams." *Proceedings of the 2005 International Workshop on Multimedia Information Retrieval.* New York: Association for Computing Machinery, pp. 183–192.

Dixon, S. 2001. "Automatic Extraction of Tempo and Beat from Expressive Performances." *Journal of New Music Research* 30(1):39–58.

Dixon, S. 2003. "On the Analysis of Musical Expressions in Audio Signals." *International Society for Optical Engineering* 5021(2):122–132.

Goto, M. 2001. "An Audio-Based Real-Time Beat Tracking System for Music With or Without Drum-Sounds." *Journal of New Music Research* 30(2):159–171.

Goto, M., and Y. Muraoka. 1997. "Issues in Evaluating Beat Tracking Systems." *Working Notes of the 1997 International Joint Conference on Artificial Intelligence*

Workshop on Issues in AI and Music—Evaluation and Assessment, pp. 9–16.
Gouyon, F., and S. Dixon. 2005. "A Review of Automatic Rhythm Description Systems." *Computer Music Journal* 29(1):34–54.
Gouyon, F., et al. 2006. "An Experimental Comparison of Audio Tempo Induction Algorithms." *IEEE Transactions on Audio Speech and Language Processing* 14(5):1832–1844.
Holm, J., et al. 2005. "Personalizing Game Content Using Audio-Visual Media." *Proceedings of the 2005 International Conference on Advances in Computer Entertainment Technology.* New York: Association for Computing Machinery, pp. 298–301.
Kitano, H. 1993. "Challenges of Massive Parallelism." *Proceedings of the 1993 International Joint Conference on Artificial Intelligence.* San Francisco, California: Morgan Kaufmann, pp. 813–834.
Large, E., and J. F. Kolen. 1994. "Resonance and the Perception of Musical Meter." *Connection Science* 6:177–208.
Pfeiffer, S., and T. Vincent. 2001. "Formalisation of MPEG-1 Compressed Domain Audio Features." Technical Report 01/196, CSIRO Mathematical and Information Sciences, Australia.
Povel, D. J., and P. Essens. 1985. "Perception of Temporal Patterns." *Music Perception* 2:411–440.
Rosenthal, D. F. 1992. "Machine Rhythm: Computer Emulation of Human Rhythm Perception." PhD thesis, Department of Architecture, MIT.
Scheirer, E. 1998. "Tempo and Beat Analysis of Acoustic Musical Signals." *Journal of the Acoustical Society of America* 103(1):588–601.
Seppanen, J., et al. 2006. "Joint Beat and Tatum Tracking from Music Signals." *Proceeding of the Seventh International Conference on Music Information Retrieval.* Victoria, Canada: University of Victoria, pp. 23–28.
Shenoy, A., et al. 2004. "Key Determination of Acoustic Musical Signals." *Proceedings of the 2004 International Conference on Multimedia and Expo.* New York: Institute of Electrical and Electronics Engineers, pp. 1771–1774.
Tzanetakis, G., and P. Cook. 2000. "Sound Analysis Using MPEG Compressed Audio," *Proceedings of the 2000 International Conference on Acoustic, Speech, and Signal Processing.* New York: Institute of Electrical and Electronics Engineers, pp. 761–764.
Wang, Y., and M. Vilermo. 2001. "A Compressed Domain Beat Detector Using MP3 Audio Bitstreams." *Proceedings of the 2001 ACM Multimedia Conference.* New York: Association for Computing Machinery, pp. 194–202.
Wang, Y., et al. 2003. "Parametric Vector Quantization for Coding Percussive Sounds in Music." *Proceedings of the 2003 International Conference on Acoustic, Speech, and Signal Processing.* New York: Institute of Electrical and Electronics Engineers, pp. 652–655.