POP MUSIC BEAT DETECTION IN THE HUFFMAN CODED DOMAIN

Jia Zhu and Ye Wang

Department of Computer Science, School of Computing
National University of Singapore, Singapore 177543
{zhujia, wangye}@comp.nus.edu.sg

ABSTRACT

This paper presents a novel beat detector that operates in the Huffman coded domain of a MP3 audio bitstream. We seek to answer two main questions. First, whether it is possible to extract beats without even partial decoding of a MP3 audio. Second, how to construct a low complexity algorithm to detect beats for multimedia applications in small devices such as mobile phones, if the answer to the first question is positive. Our investigation shows that beat detection in the Huffman coded domain can achieve fairly good results with a significantly reduced demand for computation. We also propose a graph-based algorithm to tackle the beat detection problem from an algorithmic perspective.

Keywords: Compressed domain beat detection, computation time

1. INTRODUCTION

The *beat* of a piece of pop music is the sequence of almost equally spaced phenomenal impulses which define the tempo of the music [7]. Automatic beat detection has a history of almost two decades. A fairly comprehensive review is given in [3]. Early beat tracking systems operate on MIDI; recent ones such as [2][4][6][7] operate on PCM samples. Since more and more music is now stored in compressed formats, such as MP3, it is natural to argue for the possibility and applicability of beat detection in the compressed domain. In [1], a beat detector is proposed using the MDCT coefficients, partially decoded from the MP3 bitstream. According to the definition of *compressed domain* in [9], the system proposed in [1] belongs to a *transform domain* beat detector rather than a real *compressed domain* beat detector. Besides, the performance and computational complexity of that system hinder its applications in battery-powered small devices. The beat detector described in this paper is designed to maintain the detection performance while reducing the computational complexity significantly.

The proposed beat detector has many practical applications. It is suitable for fast processing of a pop music database stored in the MP3 format, as it does not require even partial decoding. Another possible application scenario is that we need beat information for visualization or lighting during the playback of MP3 music files using a small device such as a mobile phone. This scenario is shown in Figure 1. To reduce computational complexity and memory consumption, and enable audio and visual synchronization without an unnecessarily large playback buffer, it is clearly advantageous to detect beats in the compressed domain, parallel to the decoding process. Furthermore, the proposed beat detector can be used by the user to extract beat of his/her favorite songs to control the speed of a mobile game [11].

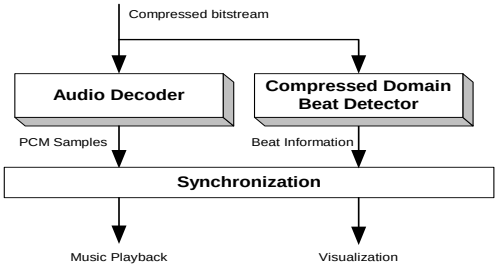

Figure 1. *A possible application scenario of the proposed beat detector.*

To our knowledge, our work is the first to design beat detection without decoding, i.e., the beat detection is based on features directly from the compressed bitstream without even performing entropy decoding. As with most beat detectors dealing with pop music, we assume that the tempo is almost constant across the entire piece of music and roughly between 70 - 160 beats per minute. The paper is organized as follows: Section 2 gives a detailed presentation of the detector. Section 3 presents the evaluation. Section 4 concludes the paper.

2. SYSTEM DESCRIPTION

As with most beat detectors, our compressed domain beat detector (which will be referred to as CBD in the sequel) consists of two main computation blocks: onset detection and beat induction. In CBD, they are carried out in cascade. A key contribution of this paper is to find good features in the Huffman coded domain as input to onset detection to maximize system performance.

2.1. Feature Selection

In an MP3 bitstream, some data is readily available without decoding, including: *window type*, *part2_3_length*, *global gain*, etc. [10]. To meet our design objective of pop music beat detection, we select features based on the following criteria: 1) The feature has good correlation with energy. 2) The feature exhibits good self-similarities. 3) The feature depends mainly on the music or the acoustic signals that are compressed, and not on the implementation of the encoder that has produced the data. This criterion renders data such as *window type* data unsuitable for beat tracking.

In practice, we use the following quantitative measure for feature selection: For each data type in the compressed domain, we form a sequence *s* by extracting the value from each granule (In MP3 bitstream, the temporal unit is a frame which is further divided evenly into two granules [10]). Then we form another sequence *b* by:

$b_{i\pm k} = 1$ *if there is an annotated beat at granule* i, $k = 0,1,2$;
$b_i = 0$ *if there is no annotated beat at granule* $i \pm k$, $k = 0,1,2$.

We calculate the cross-correlations $r_{b,s}$ between *b* and *s* at delay 0. We have applied the method on five songs, and the results are given in Table 1 (due to space constraints, we only list three features for comparison). Our investigations show that *part2_3_length* is the best compressed domain feature for beat detection because it is significantly more correlated with *b* than other compressed domain features such as *global gain*, and it is almost as correlated with *b* as *full-band energy*, calculated using transform domain data as in [1]. Despite this fact, transform domain features generally yield better accuracy for beat detection than compressed domain features do because transform domain features consist of multi-band data; the cost is computation time and memory.

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
Avg	-0.002	0.2082	0.2430

Table 1. *Results of the cross-correlation method.*

2.2. Onset Detection

Onset detection in CBD employs a very simple routine. As we have stated in Section 2.1, the input to onset detection is *part2_3_length*s, which are 12-bit integers [10]. These integers form a single dimension vector *v*. Granule *i* is supposed to contain an onset if the *i*-th element in *v* satisfies the following condition:

$$\begin{cases} v_i > thr_i \\ v_i > v_{i\pm k} \end{cases} \quad (1)$$

for k = ±1, ±2, …, ±7, where thr_i is calculated using (2):

$$thr_i = (\sum_{k=i-3}^{i+3} v_k - v_i)/4 \quad (2)$$

Using above rules, any two onsets are at least seven MP3 granules apart from each other.

2.3. Beat Induction

The beat induction process determines beat times based on the onsets detected at the previous step. Features extracted from multiple bands from a PCM bitstream are generally more reliable for onset detection, while a single parameter from the compressed domain is much noisier resulting in lower accuracy in onset detection. Therefore, we develop our beat induction algorithm to cater for onsets of low accuracy.

To facilitate illustration, we introduce a data structure called *ordered event set*, which is an *ordered set* $(S, \leq_R)$ of distinct events, to store onsets or beats. The relation $\leq_R$ is defined by: $i \leq_R j$ iff event *i* occurs earlier in time than or at the same time as event *j*. The *ordered event set* can be implemented as a list.

We formulate the beat induction problem as follows:
Input: An *ordered event set O*.
Output: A pair (*d*, *B*) which satisfies the following 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 *pair* (*i, j*), where $i, j \in B$, and *i* and *j* are adjacent in order in *B*, the time difference between the occurrence of *i* and that of *j*, denoted by *diff*(*i*, *j*), is in the range $[d - \epsilon, d + \epsilon]$. We call such a pair a *consecutive pair* of *B*.
- Condition 3: For any pair (*d*', *B*') that satisfies conditions 1 and 2 and is not identical to (*d*, *B*), $|O \cap B'| < |O \cap B|$.

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 length an algorithm is to consider respectively. In our current implementation, Q_{MIN} = 375ms and Q_{MAX} = 850ms (corresponding to tempo ranging from 70 to 160 beat per minute), and deviation ϵ is set to 25ms.

Next, we introduce another data structure that is used in the algorithm, *pattern*， which is an *ordered event set* with an *associated pair* (*s*, *d*). A *pattern P* meets the following conditions:

- Condition 1: $P \subseteq O$, where *O* is the *ordered event set* containing all the onsets.
- Condition 2: $|P| \geq 1$ and the first element in *P*, denoted by *head*(*P*), is *s*.
- Condition 3: For every *consecutive pair* (*i, j*) of *P*, if there is any, $diff(i, j) \in [d - \epsilon, d + \epsilon]$.
- Condition 4: There does not exist another *ordered event set S* such that $P \subset S$, and *S* also meets the above three conditions.

The *associated pair* (*s*, *d*) of a *pattern* uniquely identifies the *pattern* alone. This can be proven using contradiction, and using the fact that any two onsets are at least seven MP3 granules apart from each other. Thus, subsequently in this paper, we use *identification pair* and *associated pair* synonymously. If a *pattern P* has an *identification pair* (*s*, *d*), we denote *d* as the *lapse* of *P*, i.e., *lapse*(*P*) = *d*. The algorithm for extracting the pattern given the *identification pair* is straightforward – a simple iterative procedure suffices.

The beat induction algorithm starts with determining the anticipated quarter note length, using an inter-onset interval (IOI) histogram-based method which is similar to the one proposed in [2][4]. The histogram accounts for all IOIs in the range of $[Q_{MIN}, Q_{MAX}]$.

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

Definition: *Pattern A* is **compatible** with *Pattern B* **with lapse *d*** ($d > \epsilon$) iff the following condition holds:

$$tail(B) \le_R head(A), \quad lapse(A) = lapse(B) = d, \quad and \tag{3}$$

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

(ROUND is an operation that rounds its parameter to the nearest integer.) If A is *compatible* with B *with lapse* d, we denote $A \xrightarrow{d}_c B$.

The graph-based approach starts with the collection of all *patterns* with *lapse qnl* from the onsets, where *qnl* is the anticipated quarter note length. With some sophistication, a single iterative pass through the sequence of onsets shall exhaust all *patterns* with the prescribed *lapse*. We use another *ordered event set* $(L, \le_{R'})$ of the same properties and operations as $(S, \le_R)$ as the data structure to store all the *patterns*. The relation $\le_{R'}$ is defined by: $L_i \le_{R'} L_j$ iff $head(L_i) \le_R head(L_j)$.

After collecting all the *patterns*, we create a *compatibility* matrix *CM* whose size is $|L| \times |L|$ and:

$$CM[i][j] = \begin{cases} 1 & if\ get(L,i) \xrightarrow{qnl}_c get(L,j); \\ 0 & otherwise, \end{cases} \tag{4}$$

for $1 \le i, j \le |L|$. (*get* is an operation which, given a rank as input, returns the element with that rank in an *ordered event set*. *rank* is its reverse operation; namely, given an element, *rank* returns the rank of that element in an *ordered event set*.) *CM* can be viewed as the adjacent matrix of a graph $G = (V, E)$, where $V[G] = \{x \mid x \in Z \wedge x \ge 0 \wedge \exists p, x = rank(L, p)\}$, $E[G] = \{(j, k) \mid j, k \in V[G] \wedge CM[j, k] = 1\}$. This graph is directed and acyclic. The problem is transformed to finding a path $p = \langle v_0, v_1 \ldots, v_k \rangle$, where $v_0, v_1 \ldots, v_k \in V[G]$, such that $\sum_{i=0}^{k} |get(L, v_i)|$ is maximized. In order to solve the problem, we first convert graph G into another directed acyclic but weighted graph $G' = (V, E)$. G' is formed as: $V[G'] = V[G] \cup \{dummy\}$, and $E[G'] = E[G] \cup \{(dummy, k) \mid k \in V[G]\}$. The weight of an edge (j, k) in G' is assigned $-|get(L, k)|$. The negation allows us to compute minimum instead of maximum of total weights so that the *Bellman-Ford* algorithm can fit in. After we apply the *Bellman-Ford* algorithm on G' to compute the path with minimal total weights, we collect *patterns* represented by the vertices on the path and store 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 $diff(x, y) \notin [qnl - \epsilon, qnl + \epsilon]$, then x and y do not appear in the same *pattern*; x is the *tail* of one *pattern* $P1$, and y is the *head* of another *pattern* $P2$. We can also infer $P2$ is *compatible* with $P1$ *with lapse qnl*. Based on the definition of *compatibility*, we have:

$$\frac{diff(x, y)}{ROUND(diff(x, y)/qnl)} \in [qnl - \varepsilon, qnl + \varepsilon] \tag{5}$$

So 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 $diff(x, b_1) = diff(b_1, b_2) = \cdots = diff(b_k, y) = d = diff(x, y)/k$, we can infer from equation (5) that $d \in [qnl - \epsilon, qnl + \epsilon]$, which implies the tempo is validly maintained across the interpolated beats.

Figure 2 shows the compressed domain feature, detected onsets and beats of a pop music clip.

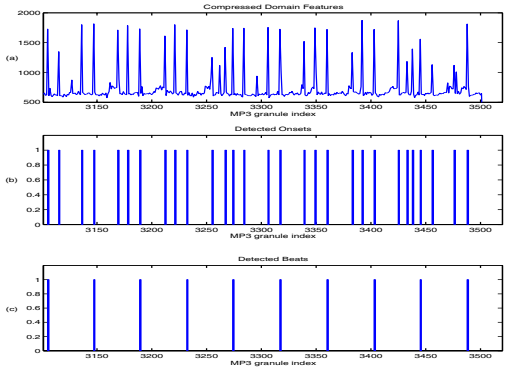

Figure 2. *(a) Compressed domain feature – part_2_3_length; (b) Detected onsets; (c) Detected beats.*

3. EVALUATION

We evaluate CBD from two aspects: accuracy and time complexity. We present our evaluation results next.

3.1. Accuracy Evaluation

We used the evaluation method presented in [5], and tested our system using 25 real-world pop songs sampled from commercial CDs. The result is shown in Table 2. CBD successfully tracked 21 of the 25 test songs, and for the remaining four songs, CBD also tracked partially the *beat* of three of them. Therefore, we conclude that CBD achieves quite satisfying accuracy for the set of test music.

No.	Song Title	Result
1	Back to you	C
2	Breathless	C
3	Burn	C
4	Crush	C
5	Drops of Jupiter	C
6	Gangsta's paradise	C
7	Heal the world	C
8	I can't tell you why	C
9	It must have been love	C

10	I want to know what love is	P
11	Losing my religion	C
12	Mmmbop	C
13	One	P
14	One of us	P
15	Road to hell	C
16	Seasons in the sun	C
17	Someday	C
18	Stayin' alive	C
19	The way it is	C
20	Torn	N
21	Truly, madly, deeply	C
22	Viva forever	C
23	Walking away	C
24	Whenever, wherever	C
25	You love making fun	C
Remark: C: completely tracked; P: partially tracked; N: completely not tracked.		

Table 2. *Experiment Results*.

3.2. Time Complexity Evaluation

As expected, CBD is much less computationally intensive than a typical PCM domain beat detector such as [2]. The reasons are: 1) CBD deals with compressed domain data, which is much smaller in quantity than its PCM domain counterpart; 2) CBD avoids operations of high computational complexity, such as transform. In this section, we give a comparison in time complexity between CBD and the *PCM domain beat detector* (PBD) presented in [2]. We plot in Figure 3 the time consumed by CBD and PBD for tracking 21 songs (those songs are completely tracked by CBD). The average duration of the 21 songs is 4.2 min. The two beat detectors ran on an IBM R40 laptop with Intel Celeron CPU of 1.7GHz and 256M of RAM.

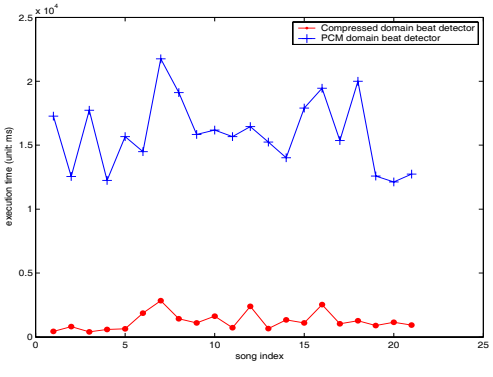

Figure 3. *Time complexity comparison*

It is clear that CBD saves a lot of computation time – the average time taken by CBD for a song with an average duration of 4.2 min is 1.23 sec while that taken by PBD is 15.92 sec. Note that for PBD, we do not include the time required for decoding; otherwise, the savings will be even higher.

We have also implemented the proposed algorithm on a HP iPAQ hx4700 running Microsoft Windows Mobile 2003 SE. In summary, 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 and 13 minutes for PCM domain beat detector (PBD). These results show that the compressed domain processing provides a significant advantage for mobile platforms, while PBD is more suitable for PC or server platforms.

4. CONCLUSION

In this paper, we have investigated the feasibility of a real compressed domain beat detection and have implemented a novel low-complexity beat detector both in laptop and PDA platforms. Although our current algorithm processes data in an all-at-once manner, it still has clear value in real-life applications such as personalized mobile gaming. We plan to investigate the feasibility of a real time algorithm for compressed domain beat detection.

5. REFERENCES

[1] Wang, Y., Vilermo, M., "*A Compressed Domain Beat Detector Using MP3 Audio Bitstreams*," ACM Multimedia 2001, Ottawa, Canada, pp. 194-202, Sept. 30-Oct. 5, 2001.

[2] Shenoy, A., Wang, Y., "*Key, Chord and Rhythm Tracking of Popular Music Recordings*," Computer Music Journal, pp. 75-86, Fall 2005

[3] Guoyon, F., Dixon, S., "A Review of Automatic Rhythm Description Systems," Computer Music Journal, 29(1):34-54, 2005

[4] Dixon, S., "*Automatic extraction of tempo and beat from expressive performances*," Journal of New Music Research, 30(1):39-58, 2001.

[5] Goto, M., and Muraoka, Y., "*Issues in Evaluating Beat Tracking Systems*," Working Notes of the IJCAI-97 Workshop on Issues in AI and Music – Evaluation and Assessment, pp.9-16, August 1997.

[6] Goto, M., "*An Audio-based Real-time Beat Tracking System for Music With or Without Drum-sounds*," Journal of New Music Research, Vol.30, No.2, pp.159-171, June 2001.

[7] Scheirer, E., "*Tempo and Beat Analysis of Acoustic Musical Signals*," Journal of the Acoustical Society of America, 103:1 (Jan 1998), pp. 588-601.

[8] Povel, D.-J., and Essens, P. (1985). "*Perception of temporal patterns*," Music Perception 2, p.p. 411-440.

[9] Wang, Y., Huang, W., Korhonen, J., "*A Framework for Robust and Scalable Audio Streaming*," ACM MM 2004, New York, USA, Oct 10-16, 2004.

[10] ISO/IEC JTC/SC29, "*Information Technology – Coding of Moving Pictures and Associated Audio for Digital Storage Media at up to about 1,5 Mbit/s-IS 11172 (Part 3, Audio)*," 1992.

[11] Holm, J., Havukainen, K., Arrasvuori, J., "*Novel Ways to Use Audio in Games,*" Game Developer Conference, 2005