An Efficient JMC Algorithm for the Rhythm Query in Music Databases

Ye-In Chang*, Jun-Hong Shen†, Chen-Chang Wu*, and Han-Ping Chou*
*Dept. of Computer Science and Engineering
National Sun Yat-Sen University
Email: changyi@cse.nsysu.edu.tw
†Dept. of Information Communication
Asia University
Email: shenjh@asia.edu.tw

***Abstract*—The rhythm query is the fundamental technique in music genre classification and content-based retrieval, which are crucial to multimedia applications. Recently, Christodoulakis *et al.* has proposed the CIRS algorithm that can be used to classify music duration sequences according to rhythms. In order to classify music by rhythms, the CIRS algorithm locates $MaxCover$ which is the maximum-length substring of the music duration sequence, which can be covered (overlapping or consecutive) by the rhythm query continuously. However, this algorithm will repeatedly generate unnecessary results during the processing, resulting in the increase of the running time. To reduce the processing cost in the CIRS algorithm, we propose the JMC (Jumping-by-$MaxCover$) algorithm which provides a pruning strategy to find $MaxCover$ incrementally. From our experimental results, we have shown that the running time of our proposed algorithm could be shorter than that of the CIRS algorithm.**

***Index Terms*—music databases, music duration sequence, rhythm queries.**

I. INTRODUCTION

In recent years, music becomes more popular due to the evolution of the technology [1], [2], [3]. Various kinds of music around us become more complex and huge [4], [5]. This explosive growth in the music has generated an urgent need for new techniques and tools that can intelligently and automatically transform the music into useful information and classify the music into correct music group precisely [6]. The rhythm query for music databases is the fundamental technique in music genre classification and content-based retrieval, which are crucial to multimedia applications.

In [7], Christodoulakis *et al.* proposed a kind of problem for rhythm queries. In the CIRS algorithm, a rhythm is represented by a sequence of "Quick" (Q) and "Slow" (S) symbols, which corresponds to the (relative) duration of notes, such that $S = 2Q$. In order to classify music by rhythms, the CIRS algorithm locates the $MaxCover$, which is the maximum-length substring of the music duration sequence which can be $cover$ed (overlapping or consecutive) by the rhythm query continuously.

This algorithm uses the notated music data of durations for the rhythm query. As compared with the rhythm query using audio music data, the CIRS algorithm can save a lot of time. Although the CIRS algorithm has the above advantages, it does not apply any pruning strategy to reduce the processing cost. This is because that the CIRS algorithm cannot decide how long one rhythm of "S"(slow) is. Therefore, it needs to trace all different duration values occurring in the duration sequence, and regards each different duration value as one rhythm of "S"(slow). So that, as the number of different duration values increases, the processing time of the CIRS algorithm increases. Therefore, in this paper, we proposed the JMC (Jumping-by-$MaxCover$) algorithm to avoid tracing all different duration values, in order to speed up answering the rhythm query. From our experimental results, we have shown that the running time of our proposed algorithm could be shorter than that of the CIRS algorithm.

The rest of paper is organized as follows. In

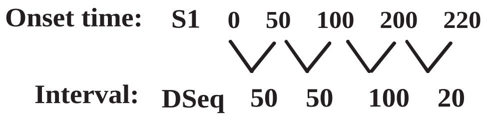

Fig. 1. Two equivalent definitions of a musical sequence

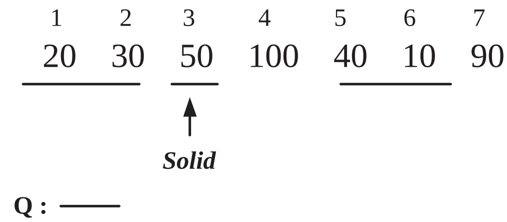

Fig. 2. The rule of q-matching and $solid$ for q = 50

Section 2, we present our proposed algorithm. The experimental results are presented in Section 3. Finally, we conclude this paper in Section 4.

II. The Jumping-By-MaxCover Algorithm

In [7], Christodoulakis *et al.* proposed a new model for song classification based on dancing rhythms. Although their CIRS algorithm can find the interesting result (the q-cover), it takes long time. Therefore, we propose an efficient algorithm named Jumping-By-$MaxCover$ (JMC), which requires shorter time to solve the same *Maximal Coverability* problem. In this section, we first describe formal definitions of duration sequences, the rhythm representation, q-match, q-cover and the $Maximal$ $Coverability$ problem [7], and then present the proposed JMC (Jumping-By-$MaxCover$) algorithm.

A. Definitions

1) Duration Sequences: A musical sequence can be thought of as a sequence of occurrences of events [7]. Consider a music signal having 5 musical events occurring at 0th, 50th, 100th, 200th and 220th milliseconds. Then, sequence $S1$ = [0, 50, 100, 200, 220] can be regarded as the corresponding sequence representing the music signal under consideration, as shown in Figure 1. Alternatively, we can represent the same music signal by stating the duration of the consecutive musical events, instead of the start time. In this algorithm, duration sequence $DSeq$ = [50, 50, 100, 20] represents the same music signal, as shown in Figure 1.

2) The Rhythm Representation: In particular, there are two types of intervals in the rhythm of a song: quick (*Q*) and slow (*S*). Quick means that the duration between two onsets is *q* milliseconds, while the slow interval is equal to 2*q*. For example, tango, the dancing rhythm, is given as sequence $SSQQS$.

Definition 1.: A rhythm Rhy is a string $Rhy = Rhy[1]Rhy[2]...Rhy[m]$, where $Rhy[j] \in Q, S$, for all $1 \leq j \leq m$.

3) q-Match:

Definition 2.: Let *Q* represent an interval of $q \in N^+$ milliseconds, and *S* represent an interval of 2*q* milliseconds. Then, *Q* is said to **q-match** with substring $DSeq[i..i']$ of duration sequence $DSeq$, if and only if

$$q = DSeq[i] + DSeq[i+1] + ... + DSeq[i'],$$

where $1 \leq i \leq i' \leq n$. If $i = i'$, then the matching is said to be *solid*. Similarly, *S* is said to **q-match** with $DSeq[i..i']$, if and only if either one of the following conditions is true

- $i = i'$ and $DSeq[i] = 2q$, or
- $i \neq i'$ and there exists $i \leq i_1 < i'$ such that $q = DSeq[i] + DSeq[i+1] + ... + DSeq[i_1] = DSeq[i_1+1] + DSeq[i_1+2] + ... + DSeq[i']$.

As with *Q*, the match of *S* is said to be $solid$, if $i = i'$. In a word, duration sequence *DSeq* can be transformed to *Q* and *S* by accumulating the consecutive ones.

For example, Figure 2 shows that duration sequences $DSeq[1..2]$, $DSeq[3]$ and $DSeq[4..5] = 50$ can be transformed to *Q*, because $DSeq[1] + DSeq[2] = 20 + 30 = 50 = q$ and so on. Moreover, duration sequence $DSeq[3]$ is a $solid$ S because of $DSeq[3] = 50 = q$.

We use an example to illustrate the q-match for a rhythm from the duration sequence. Consider the duration sequence shown in Figure 3-(a). We want to get the q-match for rhythm $Rhy = QSS$ and *q* = 50 from this sequence. First, we need to transform the $DSeq$ to the Q S representation. We have $DSeq[1] + DSeq[2]$ = 25 + 25 = 50 = *q* and $DSeq$[3] = 100 = 2*q*, so we transform $DSeq$[1..2] and $DSeq$[3] to *Q* and *S*, and so on. In fact, there are many possible results of the transforming. Next, we can find sequences $DSeq[1..5]$ and $DSeq[5..8]$ are

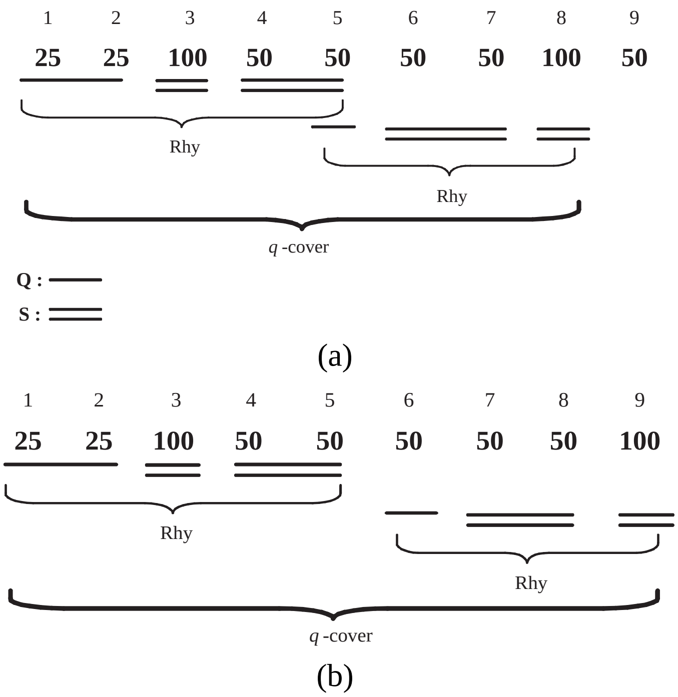

Fig. 3. q-cover of Rhy = QSS in $DSeq$, for q = 50: (a) overlapping; (b) consecutive.

matched to $Rhy = QSS$. That is, we have match sequence $MatchSeq = (1, 5), (5, 8)$,

4) q-Cover:

Definition 3.: A rhythm Rhy is said to q-**cover** substring $DSeq[i..i']$ of duration sequence $DSeq$, if and only if there exist integers $i_1, i'_1, i_2, i'_2, ..., i_k, i'_k$, for some $k \geq 1$, such that

- Rhy *q*-matches $DSeq[i_\ell..i'_\ell]$, for all $1 \leq \ell \leq k$, and
- $i'_{\ell-1} \geq i_\ell - 1$, for all $2 \leq \ell \leq k$.

In short, the *q*-covering *DSeq* consists of the overlapping or consecutive *MatchSeq*'s.

In Figure 3-(a), *MatchSeq*'s are $DSeq$[1..5] and $DSeq$[5..8]. Joining the overlapping *MatchSeq*'s becomes the *q*-cover. Therefore, we can find rhythm $Rhy = QSS$ *q*-covers $DSeq[1..8]$ for $q = 50$. In the same way, in Figure 3-(b), we can join the consecutive *MatchSeq*'s, resulting in the *q*-cover $DSeq[1..9]$.

5) The Maximal Coverability Problem: In this paper, we focus on locating $MaxCover$, the maximum-length substring of the music duration sequence, for rhythm queries. This is called the *Maximal Coverability* problem defined as follows [7]:

*Definition 4. (*Maximal Coverability *problem):* Given a duration sequence $DSeq = DSeq[1]...DSeq[n], DSeq[i] \in N^+$, and a rhythm $Rhy = Rhy[1]...Rhy[m]$, $Rhy[j] \in \{Q, S\}$, find the longest substring $DSeq[i..i']$ of $DSeq$ that is *q*-covered by Rhy among all possible values of q.

Moreover, the following restriction is applied on the above problem.

Definition 5. (At least one event is $solid$.): For each match of Rhy with a substring $t[i..i']$, there must exists at least one S in Rhy whose match in $t[i..i']$ is $solid$; that is, there exists at least one $1 \leq j \leq m$ such that $Rhy[j] = DSeq[k] = 2q$, $i \leq k \leq i'$, for some value of q.

B. The Proposed JMC Algorithm

The basic idea of the JMC algorithm contains the following five steps:

1) Finding all occurrence of *S*.
2) Transforming the areas around all the *S* into sequences of *Q* and *S*.
3) Finding the Matchings.
4) Finding the $MaxCover$.
5) Updating *Cut-Sequence*.

Our JMC algorithm does a while loop from Step 2 to Step 5, until the cut-$sequence$ is empty.

First, we will describe a portion of our algorithm which is similar to the CIRS algorithm to generate the maximal *q*-cover ($MaxCover$) by duration sequence *DSeq* and rhythm query Rhy. Next, we will introduce our proposed data structure, cut-$sequence$, which can prevent generating useless sequences. We introduce the detail of each procedure in the following section.

1) Step 1: Finding All Occurrence of S: In this step, we use the procedure which is similar to the first step in the CIRS algorithm [7]. We need to find all occurrences of $S = DiffV[\ ].Value$ in $DSeq$, where $DiffV[\ ].Value$ means the different duration value in $DSeq$. According to the chosen $DiffV[\ ].Value$, in Step 2, we can transform the areas around each of those occurrences to sequences of Q and S. Then, we have to repeat the above process for every possible value of $DiffV[\ ].Value$. A single scan through the input string suffices to find all occurrences of $DiffV[\ ].Value$.

Basically, this step contains two parts: (a) finding all different values and recording their locations;

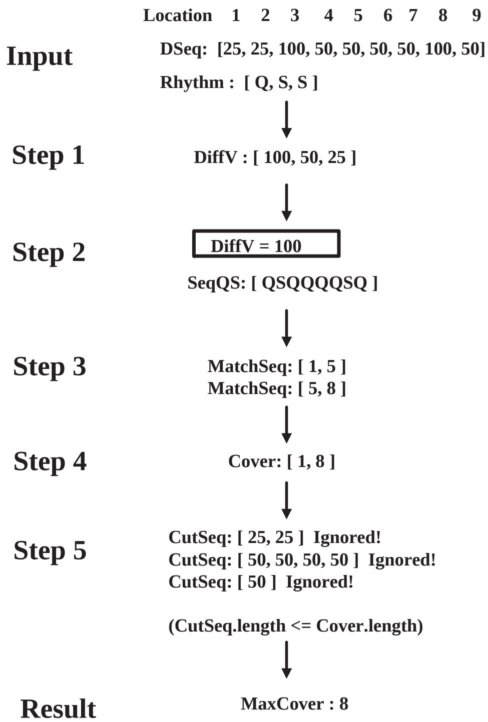

Fig. 4. A tracing example by using our JMC algorithm

TABLE I
THE RESULT OF STEP 1

DiffV[].Value	Location[ ]
100	3, 8
50	4, 5, 6, 7, 9
25	1, 2

(b) sorting all locations by $DiffV[\ ].Value$ in the descending order. Take musical sequence $DSeq$ shown in Figure 4 as a running example. According to the two parts mentioned above, we can get the result as shown in Table I.

2) Step 2: Transformation: The task of this step is to transform $DSeq$, which is a sequence of integers, to $SeqSQ$, which is a sequence consisting of *Q* and *S*, by the chosen $DiffV[\ ].Value$. Each sequence belonging to $SeqSQ$ is a sequence over Q, S for the chosen q = ($DiffV[\ ].Value$ / 2) .

In this step, our goal is to identify all the *q*-matches of Rhy in duration sequence $DSeq$. For each occurrence of the current symbol $DiffV[\ ].Value$ = $2q$ = S, we try to convert the area surrounding such an S into sequences or a tile of *Q*. When we cannot continue to make *Q*, we check whether we can make *S* instead.

Note that we first try to make Q, and in case of a failure, we try for one S. Consider $DSeq$ shown in Figure 4. It is easy to observe that in this way, we can only find *S*, if *S* is $solid$. The reason is that according to the definition, we cannot have S that cannot be divided into two consecutive *Q*'s. If we cannot make either of them, we mark the end of the sequence. Therefore, each sequence $DSeq \in SeqSQ$ consists of at least one $solid$ *S*.

This step spends the longest time in CIRS algorithm [7]. By using the notion of *cut-sequence* mentioned in Step 5, our proposed algorithm reduces the generation of $SeqSQ$ and increases the efficiency for the later steps. Consider musical sequence $DSeq$ shown in Figure 5, where we use $DiffV[1].Value$ $(= 100)$ to be *2q*, *i.e.*, $q = 50$. In Figure 5-(a), the first solid S, $DSeq[3]$ $(= 2q)$, at location 3 is located, and the transformation before this solid S is then performed. After that, the transformation after this solid S is performed as shown in Figure 5-(b). Figure 5-(c) shows that the second solid S is located and the remaining transformation is performed. The transformed result $SeqSQ$ in the proposed JMC algorithm is shown in Step 2 of Figure 4.

3) Step 3: Finding Matchings: During the matching step, the following restriction of q-match should be obeyed [7]. One S symbol in the rhythm query can be regarded as two consecutive Q symbols in the duration sequence, but the two consecutive Q symbols in the rhythm query cannot be combined as one S symbol in the duration sequence.

In this step, we consider each $SeqSQ$, for $DiffV[\ ].Value$ and identify all the *q*-matches of Rhy in $SeqSQ$. To do that efficiently, we exploit a bit-masking technique as described below. We first define some notations that we use for sake of convenience. We define S_s and S_r to indicate an *S* in $SeqSQ$ and Rhy, respectively. Q_s and Q_r are defined analogously. We first perform a preprocessing as follows. We construct $Seq01$ from $SeqSQ$ where each S_s is replaced by 01 and each Q_s is replaced by 1, as shown in Table II. We also construct Rhy' from Rhy where each S_r is replaced by 10 and each Q_r is replaced by 0, as shown in Table II. We then construct the "Invalid set" I for

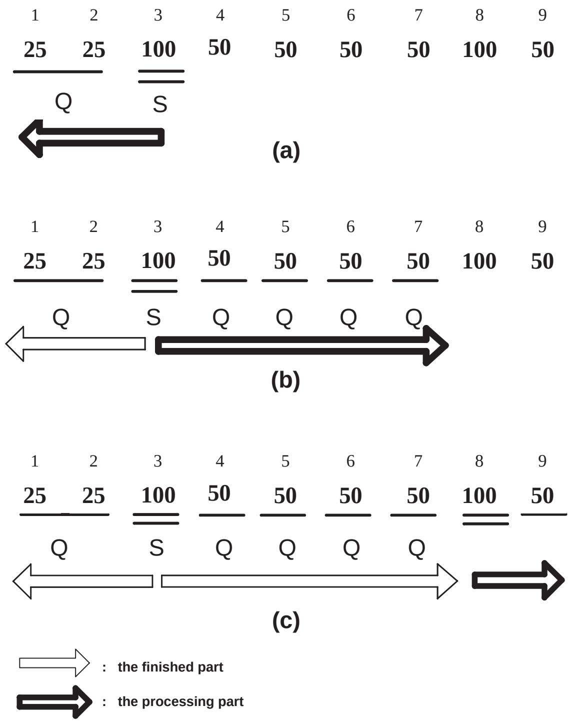

Fig. 5. Transforming *DSeq* into *SeqSQ*, for q = 50: (a) the processing before the first solid S; (b) the processing after the first solid S; (c) the processing after the second solid S.

TABLE II
THE BITMASKING TABLE

S_s	Q_s	S_r	Q_r
01	1	10	0

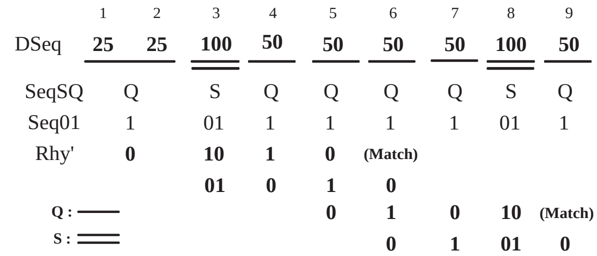

Fig. 6. Finding matchings of Rhy = QSS(01010) in $DSeq$, for q = 50

$Seq01$, where I includes each position of "1" of S_s in $SeqSQ$. This completes the preprocessing. For example, if $SeqSQ$ = $QSQQQQSQ$, we have $Seq01$ = 1011111011 and I = [3,9]. It is easy to see that no occurrence of Rhy can start at $i \in I$.

After the preprocessing is done, at each position *i* $\in I$ of $Seq01$, we perform a bitwise "OR" operation between Seq01$[i..i + |Rhy'| - 1]$ and Rhy'. If the result of the "OR" operation is all 1's, then we have found a match at position *i* of $Seq01$. However, we need to ensure that there is a *solid S* in the match. To achieve that, we simply perform a bitwise "XOR" operation between Seq01$[i..i + |Rhy'| - 1]$ and $1^{Rhy'}$ and only if the result of this "XOR" returns a nonzero value, we go on with the "OR" operation stated above.

We now discuss the correctness of q-match. Our encoding obeys the restriction of q-match. (Recall that a match occurs when the result of the bitwise OR operation is all 1's.)

1) Q_s(= 1) and Q_r(= 0) always matches: (1 OR 0 = 1).
2) Q_sQ_s(= 11) always matches with S_r(= 10): (11 OR 10 = 11).
3) S_S(= 01) can only match with S_r(= 10) : (01 OR 10 = 11).
4) Since S_s(= 01) cannot give a match with Q_rQ_r(= 00): (01 OR 00 = 01).

According to Rhy = QSS, we can find the matching sequences, $MatchSeq$'s, in $DSeq[1..5]$ and $DSeq[5..8]$, as shown in Figure 6. In this step, we use the $DiffV[1].Value$ = 100 to be $2q$. The result is shown in Step 3 of Figure 4.

4) Step 4: Finding MaxCover: In this step, we use *MatchSeq*'s generated in Step 3 to process the *q*-cover. Checking the start and end location of each *MatchSeq*, we can combine the overlapping or consecutive *MatchSeq*'s. Overall, the running time of this step is decided by the number of *MatchSeq*'s. Therefore, the running time of this step is shorter than that of Step 2 and related to the sequence generated in Step 2. Moreover, we maintain a global variable *MaxCover* to keep track of the longest cover so far.

Figure 7 shows $MaxCover$ of the running example for rhythm Rhy = QSS and q = 50. In this figure, $MatchSeq$'s $DSeq[1..5]$ and $DSeq[5..8]$ are combined into q-cover $DSeq[1..8]$. The result is the length of $MaxCover$ = 8, as shown in Step 4 of Figure 4.

5) Step 5: Updating Cut-Sequence: In this step, we construct *cut-sequences* to prune the duration

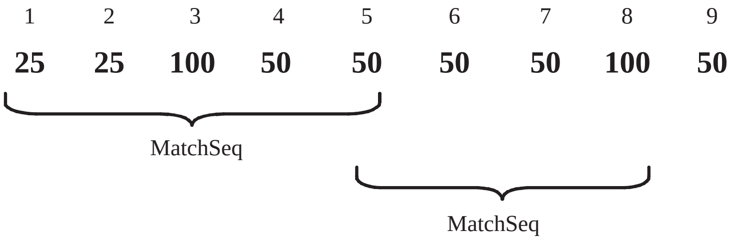

Fig. 7. Finding covers of Rhy = QSS in $DSeq$, for q = 50

Input Duration

	1	2	3	4	5	6	7	8	9
First round	**25**	**25**	**100**	**50**	**50**	**50**	**50**	**100**	**50**
Second round	**25**	**25**		**50**	**50**	**50**	**50**		**50**
Third round	**25**	**25**							

Fig. 8. The cut-sequence for input in the different rounds

sequence according to duration sequence $DSeq$ and $DiffV[].Value$, the difference duration value of $DSeq$, in each loop. We can observe that if there is a value in $DSeq$ that is larger than $DiffV[].Value$, this value will not be transformed to an S or Q rhythm in the following step. This value is a cut point of $DSeq$. It cuts off $DSeq$ into two or less subsequences. Two larger values cut off $DSeq$ into three or less subsequences, and so on. We can ignore all the cut-sequences, whose length are shorter than that of $MaxCover$, which is updated in each round. After updating cut-sequences and pruning impossible ones, the remaining cut-sequences are the input data in the next round. Figure 8 shows the cut-sequences in the different rounds.

Following the previous example, Figure 8 shows that the input *DSeq*'s for the next round, which are generated by cut-sequence $CutSeq$'s, are [25,25], [50, 50, 50, 50] and [50]. All of the lengths of *DSeq*'s are shorter than $MaxCover$ = 8 at present. Therefore, we prune all of *DSeq*'s, as shown in Step 5 of Figure 4. That is, there are no existing *DSeq* for the next round. At this point, the processing of the proposed JMC algorithm is completed. The final result is as follows:

TABLE III
THE RHYTHM QUERIES [8]

Dancing rhythms	**SQ Representations**
Bolero	$SQQSQQ$
Cha-Cha	$SSQQSSSQQS$
Foxtrot	$SSQQSSQQ$
Jive	$SSQQSQQS$
Mambo	$QQSQQS$
Quickstep	$SQQSSQQS$
Rumba	$SQSSQ$
Tango	$SSQQS$
Waltz	SSS

- Given a duration sequence $DSeq$ = [25, 25, 100, 50, 50, 50, 50, 100, 50], and a rhythm Rhy = QSS, the length of the longest substring, *MaxCover*, is 8 for *q* = 50.

III. PERFORMANCE

In order to evaluate the performance of our proposed algorithm, we compare our JMC (Jumping-by-$MaxCover$) algorithm with the CIRS Algorithm [7]. We generate the synthetic data that are similar to the duration sequence of the "ballroom dance" music. Moreover, we use the duration sequence as the input data to compare these two algorithms in the total running time.

A. Generation of Synthetic Data

The musical sequences (*e.g.* a song) can be considered as a series of onsets (or events) that correspond to music signals, such as drum beats. They are the intervals between those events, which characterize the song. In order to obtain the reliable results, we generate synthetic duration sequences as one song. Therefore, we generate several different duration sequences by using a set of different duration values ($DiffV$). Moreover, we evaluate the time of the algorithm for answering $MaxCover$ of duration sequences, which is the maximal *q-cover*, for the rhythm queries [8] shown in Table III.

The parameters used in the generation of synthetic data are shown in Table IV. N means the number of events in the duration sequence. For example, N = 1000 means that there are 1000 duration events in the song. ND means the number of different duration values (*DiffV*) in the duration sequence. For example, ND = 3 means that the

TABLE IV
PARAMETERS USED IN THE EXPERIMENT

Parameters	Meaning
N	The number of events in the duration sequence
MC	The percentage of $MaxCover$ in the duration sequence
ND	The number of different duration values (*DiffV*) in the duration sequence
Rhy	The rhythm query

duration sequence is created randomly from three $DiffV$'s. ***Rhy*** means the sequence of the rhythm query, for example, the Tango rhythm is represented by $SSQQS$. We choose nine of the most popular rhythms, listed in Table III and compare the running time of two algorithms by using each rhythm separately. ***MC*** means the percentage of $MaxCover$ in the duration sequence. According to the definition of $MaxCover$, the correct rhythm query will be repeated through the music. Therefore, the value of ***MC*** is close to 100% with querying the rhythm correctly. Beside, how to choose the $DiffV$'s is also an important issue. We describe the details as follows:

- First, we define the duration of the Q rhythm, *i.e.*, Q = 50.
- Then, the duration of the S rhythm is regarded as 2Q, *i.e.*, S = 100.
- Other $DiffV$'s must be combined as the duration of one Q rhythm. For example, we can choose $DiffV$ = [25] (25 + 25 = 50) and $DiffV$ = [30, 20] (30 + 20 = 50).

Some examples of $DiffV$ under different ND's are shown in Table V. In the case of ND = 5, we first define $Q = 50$ and $S = 100$, and then we need other three $DiffV$'s. Therefore, we choose $DiffV$ = [25] (25 + 25 = 50) and the set of $DiffV$'s = [30, 20] (30 + 20 = 50) to be the other three $DiffV$'s. Using $DiffV$ which is assigned by the user, if we also design an order to the duration sequence, we can control the value of MC.

Observing the form of the real music data, we set the default values of parameters to generate synthetic data that are similar to the real music data. In our simulation, we define a base case as shown in Table VI. According to the property of the duration events that two adjacent events can be combined to one large event, we need to generate the combination of events. Therefore, we assume that the duration of rhythm S is 100, and the duration of rhythm Q is 50. The combination case of duration events is shown in Table VII. Due to the property of S that must be combined by two consecutive Q's, we do not consider the combination case of [25, 50, 25]. An example of the synthetic data generation with $DiffV$ = [100, 50, 25], Rhy = SSQQS, N = 17 and MC = 100% is shown in Table VIII.

TABLE V
AN EXAMPLE OF THE $DiffV$ (UNDER DIFFERENT ND)

ND	$DiffV$
2	[50, 25]
3	[100, 50, 25]
4	[200, 100, 50, 25]

TABLE VI
BASE VALUES FOR PARAMETERS USED IN THE SIMULATION

Parameters	Default values
N	10000
MC	100%
ND	3 (Different duration values are 100, 50 and 25)
Rhy	$[S, S, Q, Q, S]$

TABLE VII
THE COMBINATION CASE OF DURATION EVENTS [100, 50, 25] FOR S =100

Rhythm	Combination
S	[100]
	[50, 50]
	[50, 25, 25]
	[25, 25, 50]
	[25, 25, 25, 25]
Q	[50]
	[25, 25]

TABLE VIII
AN EXAMPLE OF THE DATA GENERATION (ND = 3, $DiffV$ = [100, 50 ,25] , Rhy = SSQQS, N = 17 AND MC = 100%)

N	**S**	**S**	**Q**	**Q**	**S**
1–8	[100]	[25, 25, 50]	[50]	[25, 25]	[100]
9–17	[50, 25, 25]	[50, 50]	[25, 25]	[50]	[100]

TABLE IX

A COMPARISON OF THE RUNNING TIME (MILLISECONDS) OF THE JMC ALGORITHM AND THE CIRS ALGORITHM (UNDER THE BASE CASE)

Algorithm	The running time
CIRS	44
JMC	8 (reduced 81.8%)

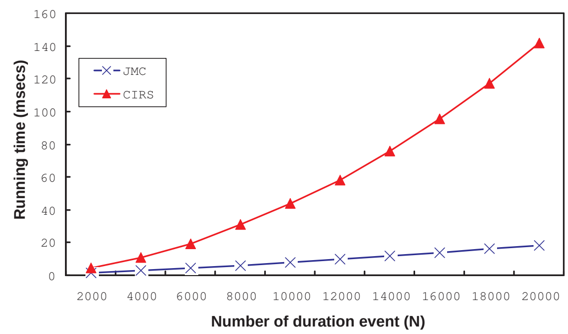

Fig. 9. A comparison of the running time of the JMC algorithm and the CIRS algorithm by using the different number of duration sequences (N)

In order to control *MC* = 100%, we use the combination, as shown in Table VII, to be one element of the input. Moreover, we use the rhythm query as the order of data generation. For example, the first symbol of the rhythm query $SQQSS$ is S, and we generate the combination of duration events from five cases of Rhythm S in Table VII randomly, and so on. In this way, we can generate the duration sequence that is covered by the rhythm query, and that is $MaxCover$ which we need.

B. Simulation Results of Synthetic Data

Now, we make a comparison of our JMC algorithm with the CIRS algorithm by using the synthetic data. For the base case shown in Table VI, we make a comparison of the running time of our algorithms and the CIRS algorithm. The result is shown Table IX, which is the average of 20 duration sequences. On the average, our algorithm can reduce about the 81.8% running time of the CIRS algorithm. The value of the reduced percentage can be calculated by using the formula described as follows:

$$reduced\ percentage = \\ (1 - \frac{the\ running\ time\ of\ the\ JMC\ algorithm}{the\ running\ time\ of\ the\ CIRS\ algorithm}) \times 100\%.$$

In the first case, we vary the value of N, the number of events in the duration sequence. The range of N is set to 2000, 4000, 6000, ..., and 20000, while the other parameters are kept as their base values. Under changing the value of N, a comparison of the running time by using the JMC algorithm and the CIRS algorithm is shown in Figure 9. We can observe that when the value of N increases, the running time by using the JMC algorithm and the CIRS algorithm also increases. However, our algorithm needs shorter time to answer the same rhythm query than the CIRS algorithm. This is because that our algorithm can filter the false cut sequence (piece of the duration sequence) in advance, whereas the CIRS algorithm does not use the pruning strategy. In this case, according to ND = 3, the CIRS algorithm needs to run the algorithm three times completely. In our algorithm, we get the result of MC = 100% for q = 100 at the first round. Then, we can observe that the length of $MaxCover$ is long enough to prune all cut sequences (the duration sequence of the second round for q = 50). Therefore, we do not need to run our algorithm in the second round. As compared to the CIRS algorithm, our JMC algorithm can reduce up to 66.7% of the running time.

In the second case, we vary the kinds of Rhy, the query rhythm. The nine kinds of Rhy are listed in Table III, and the base values are used for the other parameters. Under changing the different Rhy, a comparison of the running time by using the JMC algorithm and the CIRS algorithm is shown in Figure 10. We can observe that no matter what kind of *Rhy* is applied, the running time of the JMC algorithm is shorter than that of the CIRS algorithm. Moreover, our algorithm needs shorter time to answer the same rhythm query than the CIRS algorithm. This is because that our algorithm can filter the false cut sequence (piece of the duration sequence) in advance, whereas the CIRS algorithm does not use the pruning strategy. As compared to the CIRS algorithm, our JMC algorithm can reduce up to 78.7% of the running time.

IV. CONCLUSION

In this paper, we have presented the JMC (Jumping-By-$MaxCover$) algorithm to locate the

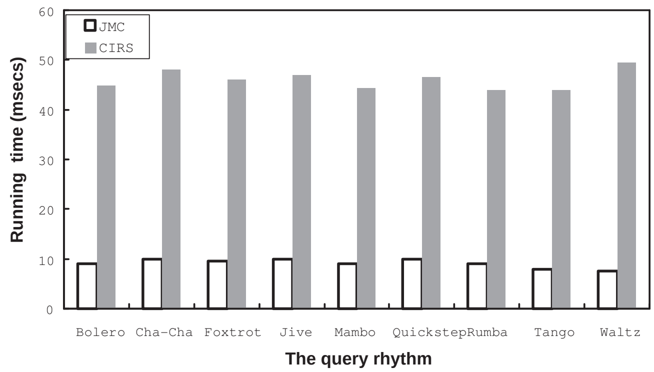

Fig. 10. A comparison of the running time of the JMC algorithm and the CIRS algorithm by using different rhythm queries (Rhy)

maximum-length substring of the music duration sequence for rhythm queries, which can reduce the process cost of the CIRS algorithm [7]. Our proposed algorithm follows the definition of the CIRS algorithm and provides the pruning strategy to generate the result incrementally. From our simulation results, we have shown that our algorithm can reduce up to the 81.8% running time of the CIRS algorithm.

REFERENCES

[1] A. Ghias, J. Logan, D. Chamberlin, and B. C. Smith, "Query by Humming-musical Information Retrieval in an Audio Database," in *ACM Multimedia*, 1995, pp. 231–236.

[2] D. Little, D. Raffensperger, and B. Pardo, "A Query by Humming System That Learns from Experiences," in *Proc. of the 8th Int. Conf. on Music Information Retrieval*, 2007, pp. 23–27.

[3] E. Unal, E. Chew, P. G. Georgiou, and S. S. Narayanan, "Challenging Uncertainty in Query by Humming Systems: A Fingerprinting Approach," *IEEE Trans. on Audio, Speech, and Language Processing*, vol. 16, no. 2, pp. 359–371, Feb. 2008.

[4] J. S. Downie, "Music Information Retrieval," *Annual Review of Information Science and Technology*, vol. 37, pp. 295–340, 2003.

[5] N. Orio, "Music Retrieval: A Tutorial and Review," *Foundations and Trends in Information Retrieval*, vol. 1, no. 1, pp. 1–96, Jan. 2006.

[6] H. C. Chen, Y. H. Wu, Y. C. Soo, and A. L. P. Chen, "Continuous Query Processing over Music Streams Based on Approximate Matching Mechanisms," *Multimedia Systems*, vol. 14, no. 1, pp. 51–70, June 2008.

[7] M. Christodoulakis, C. S. Iliopoulos, and M. S. Rahman, "Identifying Rhythms in Musical Texts," *Int. Journal of Foundations of Computer Science*, vol. 19, no. 1, pp. 37–51, Feb. 2008.

[8] A. L. P. Chen, C. S. Iliopoulos, S. Michalakopoulos, and M. S. Rahman, "Implementation of Algorithms to Classify Musical Texts According to Rhythms," in *Proc. 4th of Int. Sound and Music Computing Conf.*, 2007, pp. 11–13.