Francis Chin and Stephen Wu
Department of Computer Science
University of Hong Kong
Pokfulam, Hong Kong
chin@csd.hku.hk
swu@csd.hku.hk

An Efficient Algorithm for Rhythm-finding

One of the tasks in popular music arrangement is to determine a rhythm pattern to accompany a given melody. Musical rhythm can be thought of as repeated recurrences in alternate heavy and light beats. In this paper, we shall describe a method to assign rhythm pattern(s) to a given melody, and then show how to implement the method on computer by a fast algorithm.

Background

Musical tasks for which computers have been found useful can be divided into two main categories: artistic and nonartistic. The artistic tasks, such as composition and performance, are those that require aesthetic proficiency; the nonartistic are those that require professional knowledge rather than aesthetic sense. Examples of nonartistic tasks in which computers could be of help are the synthesis of electronic music, analysis of melodies, and transcription of melodies into scores. Some nonartistic tasks are used to support artistic tasks, an example of which is computer-aided composition. Generally speaking, artistic tasks are performed by artists while nonartistic tasks can be performed by scientists.

Not much has been done for the "in-between" tasks, however. Music arrangement is one of those tasks. Music arrangement is the job of turning the composer's raw score, which may consist of the main melody only, into the final score to be performed. In classical music, the arranging is usually done by the composer, but for pop music, composing and arranging are often carried out by different persons. Before doing the actual arrangement, the arranger has to analyze the melody and extract appropriate features; while doing the arranging, the arranger has to determine various elements of the final product, such as chords, tempi, expressions, and rhythms, and to compose the full score for various instruments; thus the arranger needs both artistic and nonartistic skills. In this article, we concentrate on one of the tasks in music arranging, namely, finding a rhythm pattern to accompany a given melody. To narrow the area of investigation, we have chosen to focus on melodies from current Hong Kong popular music.

Computer Music Journal, Vol. 16, No. 2, Summer 1992,
© 1992 Massachusetts Institute of Technology.

Description

Musical rhythm can be thought of as repeated recurrences in alternate heavy and light beats. Notice that we define the word "rhythm" here slightly differently from the way it is defined by Cooper and Meyer (1963). They emphasize rhythmic groups consisting of strong and weak beats and also architectonic levels formed by grouping. Lerdahl and Jackendoff (1983) further developed the idea into a "generative theory," and they too emphasized hierarchical grouping and metrical structure. On the other hand, we are concerned with the more intuitive human perception of heavy and light beats, which is discussed below. We deliberately use the words "heavy" and "light" to avoid confusion with the "strong" and "weak" beats concept of Cooper and Meyer.

In most Hong Kong pop music, we can perceive either a constant rhythm pattern throughout the accompaniment of a melody or several rhythm patterns assigned to different parts of the melody. For example, bassanova, march, and tango are some of the well-known rhythm patterns. Of course, there are also many kinds of patterns that we could not name, since they are not widely enough used to deserve a name. These patterns would also be useful when the melody is to be accompanied by single instruments like piano, guitar, or electronic organ, or as a rough guide for drum patterns to be used along with the melody. The case of basic rhythm patterns in pop music is very different from the case for classical music, where such patterns are often not apparent. Our goal is to find suitable rhythm patterns for a given pop song melody.

It must be noted, however, that these patterns are not, and should in no way be, unique. Therefore, when carrying out the arrangement, it would be better if the computer could list some useful information and suggest several alternatives for the arranger to decide upon. It must be stressed that the final decision should be in human hands.

Different people have different methods to assign rhythm patterns to a melody. Some claim to have no formal methods, only "intuition," and some methods may even contradict others. If we are going to let the computer help us with this problem, some general and formal methods must be incorporated into the computer system. One method that we feel is workable and explicitly expressible is as follows.

For simplicity, let the pattern be binary-valued in nature. In other words, at any given time, there should be either a "heavy" beat or a "light" beat in the rhythm. We shall abbreviate these to heavy and light, respectively. The heavy can be a "thud" of a bass drum or timpani, a cymbal, or strong bass sounds from the keyboard or the guitar. Generally speaking, when there is heavy in the pattern, the human ear can perceive something loud or strong, especially at the bass. Light will then be the opposite of heavy, including complete silence and also the sustention of a heavy beat.

We also assume that the recurrence period of a rhythm pattern is one bar (measure). That is, the pattern would repeat itself for every bar of the melody. This has been found to be the case in typical pop songs. We can then slice the bar into equal time units, the number of which depends on the time signature of the melody and the notes with minimum duration that appear in the melody. If, for example, the time signature for a melody is 4/4, and the shortest note in it is a quaver, or eighth-note, then we divide each bar into eight equal time units. It is clear that in doing this, we are assuming that the duration between elements of the rhythm pattern should not be shorter than the shortest note in the melody.

An alternative and perhaps easier method is to determine the number of time units per bar from the time signature. In most cases the number of time units to a bar can be found by multiplying the upper value of the time signature by 2. For example, there would be 12 time units to a bar for a 6/8 time signature. We do not use this method here because the result would not be as satisfactory for a few of the melodies we used for testing.

We can then assign either a 1, which stands for heavy, or a 0, which stands for light, to each time unit in the bar. In this way, the rhythm pattern can be defined as a binary pattern with a constant number of elements for each melody, provided, of course, that the melody does not change its time signature during its course.

Suppose we now want to find exactly one pattern for the whole melody. It would be nice if we can have heavy in the rhythm pattern whenever there is heavy (the start of a note) in the melody, and have light in the pattern whenever there is light in the melody (when a note is being sustained or when there is a rest). (Note that the heavy and light rhythmic pattern is not the same thing as the melody.) But that would mean that the melody is made up entirely of bars with the same duration pattern, which seems unlikely for real-world music. We would then like to find the pattern that best matches the melody. "Best" means here the maximum number of coincidences the heavies in the melody and the 1's in the rhythm pattern, and also the lights in the melody and the 0's in the rhythm pattern, when the rhythm pattern is repeated for the length of the whole melody.

One point that often complicates the matter is that it may be necessary for more than one basic rhythm to be assigned to a melody; that is, a melody may be partitioned into different contiguous parts in which different patterns are needed. This is because a melody may change its "style" somewhere along its course, and thus require more than one rhythm pattern for the whole melody.

How, then, could a music arranger select the appropriate number of rhythmic pattern sections for a melody, and where should the boundary for the sections appear once the number of sections has been determined? It would be nice if some analysis could be carried out on the melody and some information extracted from it could help the human make the decision. In the following sections, we describe a method to assign rhythm pattern(s) to a given melody, then show how to implement the method on computer efficiently.

Finding Single Rhythm Patterns

We shall start by considering the simplest case—finding exactly one rhythm pattern to accompany the melody. Assuming that the time signature does not change within the melody, let B be the bar length (or cycle length in this case) with respect to minimum time units. For example, if the time signature is 4/4, and the fastest duration allowed for a rhythm beat is a quaver, then B would be 8. Let n be the total number of bars in the melody. Let $M \equiv (bar(1), \mathrm{bar}(2), \ldots, bar(n))$ be the melody and $bar(j)$ be the rhythm pattern of the jth bar of the melody, $j = 1, \ldots, n$. In other words, $bar(j)$ can be expressed as the vector $(bar(j, 1), bar(j, 2), \ldots, bar(j, B))$, where each component is either a 0 or a 1; that is,

$$bar(j) \equiv (bar(j, 1), bar(j, 2), \ldots, bar(j, B)),$$

where

$$bar(j, b) = \begin{Bmatrix} 0 \\ 1 \end{Bmatrix}, \qquad b = 1, \ldots, B.$$

Let $q = (q(1), q(2), \ldots, q(B))$ and $p = (p(1), p(2), \ldots, p(B))$ be two rhythm patterns. The error between these two rhythm patterns can be defined as

$$E(q, p) \equiv \sum_{b=1}^{B} q(b) \oplus p(b),$$

where

$$x \oplus y = \begin{Bmatrix} 0 \text{ if } x = y \\ 1 \text{ otherwise} \end{Bmatrix}.$$

For example, if $q = (1, 0, 1, 1)$, and $p = (0, 1, 1, 1)$, then $E(q, p) = 2$.

Let us now consider the simple case of finding exactly one rhythm pattern to accompany the melody M. The problem is to determine a "best" rhythm pattern $p = (p(1), \ldots, p(B))$, such that the total error,

$$E(M, p) \equiv \sum_{j=1}^{n} E(bar(j), p),$$

is minimum. Note that,

$$\begin{aligned} E(M, p) &= \sum_{j=1}^{n} E(bar(j), p) \\ &= \sum_{j=1}^{n} \sum_{b=1}^{B} bar(j, b) \oplus p(b) \\ &= \sum_{b=1}^{B} \sum_{j=1}^{n} bar(j, b) \oplus p(b). \end{aligned}$$

Since the $p(b)$'s do not depend on each other, for each $b = 1, \ldots, B$, we can find each $p(b)$ to minimize each element of the summation individually, that is, to find $p(b)$ such that

$$\sum_{j=1}^{n} bar(j, b) \oplus p(b)$$

is minimum. But we know that

$$bar(j, b) \oplus p(b) = \begin{cases} bar(j, b) \text{ if } p(b) = 0 \\ 1 - bar(j, b) \text{ if } p(b) = 1. \end{cases}$$

Therefore

$$\sum_{j=1}^{n} bar(j, b) \oplus p(b) = \begin{cases} sum(n, b) \text{ if } p(b) = 0 \\ n - sum(n, b) \text{ if } p(b) = 1 \end{cases}$$

where

$$sum(n, b) = \sum_{j=1}^{n} bar(j, b), \qquad b = 1, \ldots, B.$$

In order for this error value to be minimum, we choose $p(b) = 0$ if $sum(n, b) < n - sum(n, b)$, that is, $sum(n, b) < n/2$; and choose $p(b) = 1$ if $sum(n, b) > n/2$. If $sum(n, b) = n/2$, the error would be the same regardless of whether we choose $p(b)$ to be 0 or 1. In this case, we introduce an extra number, 2, to denote this "don't care" condition. In actual practice, the case of a "2" in the rhythm pattern could be assigned any type of sound, it could be heavy, light, or anything in-between. We now have the following results:

$$p(b) = \begin{Bmatrix}0\\2\\1\end{Bmatrix} \text{ if } sum(n, b) \begin{Bmatrix}<\\=\\>\end{Bmatrix} \frac{n}{2},$$
$$b = 1, \dots, B.$$

The corresponding total error of the melody, M, is thus:

$$\begin{aligned}E(M) &\equiv \min_{p}\{E(M, p)\}\\ &= \sum_{b=1}^{B} \begin{Bmatrix} sum(n, b) \text{ if } p(b) \neq 1 \\ n - sum(n, b) \text{ if } p(b) = 1 \end{Bmatrix}\\ &= \sum_{b=1}^{B} \begin{Bmatrix} sum(n, b) \text{ if } sum(n, b) \leq \frac{n}{2} \\ n - sum(n, b) \text{ if } sum(n, b) > \frac{n}{2} \end{Bmatrix}\\ &= \sum_{b=1}^{B} \left(\frac{n}{2} - \left| sum(n, b) - \frac{n}{2} \right| \right).\end{aligned}$$

An Example

Suppose we have a melody as follows (in simplified pitch notation), with $n = 8$ and $B = 4$:

|1 2 3 −|1 2 3 −|2 1 2 3|1 − 1 −|$\dot{1}$ 7 6 −|
$\dot{1}$ 7 6 −|7 $\dot{1}$ 7 $\dot{1}$|6 − 6 −|

Assuming four rhythm pattern components per bar, we extract the rhythm patterns for each bar as follows:

bar 1: (1, 1, 1, 0)
bar 2: (1, 1, 1, 0)
bar 3: (1, 1, 1, 1)
bar 4: (1, 0, 1, 0)
bar 5: (1, 1, 1, 0)
bar 6: (1, 1, 1, 0)
bar 7: (1, 1, 1, 1)
bar 8: (1, 0, 1, 0)

If we want to find a rhythm pattern to accompany the whole melody, and also to find the corresponding error, we would proceed as follows. Sum up every component of the patterns, giving $sum(n) = (sum(n, 1), \dots, sum(n, 4)) = (8, 6, 8, 2)$. Calculate half the number of bars, $n/2$. In this case, it is 4. Check each component sum against half the number of bars. The first three component sums [8, 6, 8] are all greater than 4, therefore they would be assigned a 1 for the rhythm pattern; on the other hand, the last component sum, 2, is less than 4, therefore it will be assigned a 0. The "best" rhythm pattern, p, will thus be (1, 1, 1, 0). This rhythm pattern matches exactly with bars 1, 2, 5, and 6, while there is a discrepancy of 1 with bars 3, 4, 7, and 8. Therefore, the error value of this pattern with the melody, $E(M)$, is 4. This value is the smallest among all possible rhythm patterns.

Finding Multiple Rhythm Patterns

In this section, we have to consider multiple rhythm patterns for a single melody, therefore we further introduce the following concepts so that our discussions above can be generalized.

Let $segerr(i, j)$ stand for the error from bar i to bar j, inclusive, when the "best" rhythm pattern is assigned to these bars; let $patt(i, j)$ denote this best rhythm pattern. Note that, in particular, $segerr(1, n) = E(M)$ and $patt(1, n) = p$. Also note that for any $0 < i \leq j \leq n$, $segerr(i, j)$ is a nonnegative integer and $patt(i, j)$ is an array of B elements where each element is a 0, 1, or 2.

Let $err(k, j)$ be the error from bar 1 to bar j of the melody when these bars are separated into k rhythm sections, and the best rhythm pattern is assigned to each of these sections, where $k = 1, \dots, n$, $j = 1, \dots, n$. The sections are always divided on bar boundaries, and we call these bar numbers *breakpoints.* There are $k - 1$ breakpoints for a melody with k sections. The error values for these measures are calculated by summing up the error of each section, and the section boundaries are determined by picking the ones that make the error smallest. In particular, for a single rhythm pattern, we already know how to find $err(1, n)$, which is exactly the corresponding $segerr(1, n)$, or equivalently, $E(M)$.

Our goal would be to identify the breakpoints, the corresponding rhythm patterns, and the error values of a given melody, $err(k, n)$, for different values of the number of sections, k, so that after the arranger has chosen the appropriate value for k, he or she can determine the breakpoints and the rhythm patterns for that melody easily.

We have chosen to solve the problem using the dynamic programming technique (Bellman 1957). The reason for this choice is the observation that this problem follows the principle of optimality (Bellman 1957). Assume that we have found a set of k breakpoints for a given melody such that the total error is minimum. We now truncate the last section away; that is, we make the melody shorter by cutting away the portion from the last breakpoint until the end of the melody. The set of the remaining $k - 1$ breakpoints will also be the optimal breakpoints for the truncated melody.

To put it another way, suppose we now know the optimal solution for $k - 1$ breakpoints for every possible truncated melody, that is, from measure number 1 until any measure number in the original melody. Suppose also that we want to find the optimal solution for k breakpoints for the whole melody, that is, from measure number 1 until measure number n. We can divide the melody into two parts arbitrarily, and have already known the optimal solution of $k - 1$ breakpoints for the first part, and also the optimal solution of no breakpoints for the second part. We sum up the error from both parts and obtain the total error for the melody using this particular place of division into two. If we consider all of the possible division points in turn, n of them in total, we can obtain the point that minimizes the total error. Finally, we append this point to the set of $k - 1$ breakpoints for the first part and thus obtain the final solution of k breakpoints.

We can now write out the recurrence equations for dynamic programming:

```
$err(k, 0) = 0, \quad k = 1, \ldots, m$
$err(1, j) = segerr(1, j), \quad j = 1, \ldots, n$
    for $k \leftarrow 2$ to $m$ do
    for $j \leftarrow 1$ to $n$ do
        $err(k, j) = \min_{0 \le i < j}\{err(k - 1, i) + segerr(i + 1, j)\}$
```

where m is the maximum number of sections that we would like to consider. The algorithm for this is listed as C-pseudocode in the Appendix.

Analysis of the Algorithm

For the initialization part of the algorithm, it is obvious that the time complexity is $O(m)$. For the dynamic programming part, since we need $O(n)$ to find a minimum, and this $O(n)$ is nested within n and m for-loops (m stages of n possible divisions), we need $O(mn^2)$ complexity. Finally, for the backtracking part, the time complexity is $O(m^2)$. Since $m < n$, the overall time complexity of the algorithm for finding the breakpoints for all number of sections from 1 to m is therefore $O(mn^2)$. In the extreme case where $m = n - 1$, the time complexity would be $O(n^3)$. (This time complexity is the fastest that could be found by the authors for the time being. Compare this with the $O(n!)$ complexity of the brute-force method!)

Implementation and Experimental Results

The dynamic programming algorithm has been implemented on a Sun-3 workstation computer using the C programming language. Forty local popular songs of the 1980s were selected randomly from three songbooks (*Golden Song, Golden Song,* Golden Song Publishing Company, Hong Kong, and *Platinum Awards Vols. 17 and 20,* Lok Tin Publishing Company, Hong Kong) for testing.

We used a simple pitch-duration method to represent the melodies (the pitches are used in heuristics only for the time being), and the program generates the recommended breakpoints and the corresponding rhythm patterns for each k value. It also displayed the total error for each of the solutions. For simplicity and practical reasons, the maximum output k value (m in the algorithm of the "Finding Multiple Rhythm Patterns" section) is limited to 5 (an explanation will be given below), and as stated in the "Description" section, the cycle time of rhythm pattern is taken to be one measure; that is, the rhythm pattern is assumed to repeat itself for every bar. The appropriate k value and hence the breakpoints and patterns were then left for the human to determine.

We tested the results by comparing them with the authors' intuitive decisions. This was done by writing down the authors' intuitive observation of

Fig. 1. Output for song 5.

```
32 measures

k    err    avg.err/bar    dec. %

0    89     2.78125
1    63     1.96875    29.21
2    59     1.84375     6.35
3    55     1.71875     6.78
4    52     1.62500     5.45
5    48     1.50000     7.69

breakpoints:

k=1:   15
k=2:   15, 16
k=3:   15, 16, 29
k=4:   14, 15, 16, 31
k=5:   15, 16, 29, 30, 31

patterns:

k=0:   10000211
k=1:   10000111   10001000
k=2:   10000111   11110000   10001022
k=3:   10000111   11110000   10001010   10001001
k=4:   10220111   00000001   11110000   10001011   00000000
k=5:   10000111   11110000   10001010   11011001   10101101   00000000
```

the values before running the program. The authors' decisions were acquired by looking at the raw score of the songs only. A simple heuristic to choose the k value from the program output is to find a k value with a significantly smaller corresponding error than that for the $(k - 1)$ value, and the errors for the following larger k values do not decrease significantly. If no such significant decrease in error occurred, we were in favor of taking k as either 0 or the smallest k value where a "not-so-significant drop" occurs. If some of the breakpoints got too close together, or got too close to the start or end of the melody, these k values would not be chosen as breakpoints, as these situations seldom occur. We had no formal definition for "significant" or "not-so-significant"; we just used our own intuition to establish that. In this test, we used the program mainly as a rhythm analyzer, for dividing the melody into rhythm sections. The program output for three selected songs is shown in Figs. 1–3.

In song 5 (Fig. 1), there is a very large drop in error value when $k = 1$, and also we have a rather satisfactory corresponding breakpoint value about halfway through the song; therefore we take k to be 1. In song 8 (Fig. 2), the drop in error value is not significant for all values of k; therefore we take k as 0. In song 12 (Fig. 3), the error value has a significant drop at $k = 2$. Since the corresponding breakpoint values 16 and 26 are neither too close to each other nor too close to the start or the end of the melody, we accept k as 2.

For convenience, the results of the comparison between the "expected" values and those generated by the program (instead of the raw program output values) are shown in Table 1.

Table 1 has six columns. The first contains the numerical codes for the selected songs. The second shows the number of measures in each song. The third and fourth show the authors' intuitive k values and the corresponding breakpoints. When the k-value of column 3 is 0, no breakpoints would be necessary; therefore there are no entries for these rows in column 4.

Column 5 shows the k values that were chosen from the program outputs according to the simple heuristics described earlier. Finally, in column 6, the breakpoints generated by the program are shown when the k values of column 3 (intuition) are used. For the same reason as above, this column would also be left blank when the k value in column 3 is 0.

From Table 1, we can see that the results obtained from the program are close to those due to our human intuition. Four of the songs used in the testing (namely, songs 10, 20, 21, and 23) are "artificial" songs, which were formed by merging two songs that have significantly different rhythm patterns. For these artificial songs, the program output helped the human to detect the boundary of the component songs correctly. For most of the cases in Table 1, the k values in columns 3 and 5 match. How-

Fig. 2. Output for song 8.

```
32 measures

k     err    avg.err/bar    dec. %

0     128    2.46154

1     124    2.38462    3.13

2     118    2.26923    4.84

3     115    2.21154    2.54

4     110    2.11538    4.35

5     107    2.05769    2.73

breakpoints:

k=1:   36

k=2:   36, 41

k=3:   36, 40, 41

k=4:   10, 16, 36, 42

k=5:   10, 16, 25, 36, 42

patterns:

k=0:   10100011

k=1:   10100011   11100111

k=2:   10100011   11000101   11100111

k=3:   10100011   11002101   10000000   11100111

k=4:   10100011   12000101   12100011   12000101   11100211

k=5:   10100011   12000101   11100011   10100011   12000101   11100211
```

Fig. 3. Output for song 12.

```
32 measures

k     err    avg.err/bar    dec. %

0      47    1.46875

1      46    1.43750    2.13

2      41    1.28125   10.87

3      38    1.18750    7.32

4      36    1.12500    5.26

5      34    1.06250    5.56

breakpoints:

k=1:   31

k=2:   16, 26

k=3:   16, 26, 27

k=4:   16, 17, 26, 27

k=5:   16, 18, 19, 26, 27

patterns:

k=0:   10000011

k=1:   10000011   00000011

k=2:   10000011   10001010   10000011

k=3:   10000011   10001010   10001101   10000011

k=4:   10000011   11011010   10001010   10001101   10000011

k=5:   10000011   12021010   11100110   10001010   10001101   10000011
```

ever, there are some small discrepancies between columns 4 and 6. This can be explained by the fact that a melody's rhythm pattern often changes on the last few bars before the end of a section. For example, a long tie or rest may occur in those bars at the cadence. In such cases, the breakpoints may be determined incorrectly by the program by including those special bars into the sections immediately after the breakpoints. Practically, these bars are the ones that deserve some special treatment such as using a "fill-in" rhythm for them.

We can also see that for normal songs, the k value would seldom be greater than 2. This justifies limiting the maximum k value to 5 when running the program, so as to save computer time and outputs. However, theoretically speaking, the maximum k value can be extended to n (the number of measures of the song).

It is expected that if a few simple heuristics were added to the algorithm, the results would be improved. For example, we usually prefer to place breakpoints after even bar numbers (since a section

Table 1. Expected (intuition) and actual (program) results for selected songs

		Intuition		*Program*	
Song No.	*No. of Bars*	*K*	*Break-points*	*Recom-mended K*	*Break-points*
01	20	2	8,11	0	8,11
02	32	1	14	1	15
03	20	0		0	
04	24	0		0	
05	32	1	16	1	15
06	26	1	18	1	18
07	64	0		0	
08	52	0		0	
09	64	1	32	1	31
10	58	1	26	1	26
11	56	1	24	1	23
12	32	2	16,27	2	16,26
13	32	0		0	
14	34	0		0	
15	16	0		0	
16	23	1	8	1	7
17	27	2	12,18	2	7,18
18	50	1	34	1	33
19	24	1	16	1	16
20	52	1	20	1	19
21	57	1	32	1	32
22	32	0		0	
23	44	1	16	1	15
24	40	1	16	1	14
25	28	0		0	
26	32	0		0	
27	26	1	16	1	16
28	20	0		0	
29	39	0		0	
30	30	1	18	1	18
31	47	2	16,32	1	15,28
32	48	1	32	1	31
33	25	0		0	
34	24	1	16	1	13
35	33	1	16	1	15
36	40	1	24	1	22
37	32	0		0	
38	24	1	16	1	14
39	24	0		0	
40	44	1	24	1	25

is usually made up of an even number of bars) and to partition the melody into sections of roughly equal lengths. We would also like to avoid rhythm patterns that have a long rest (or a long consecutive sequence of light beats) within them, especially for slow songs; otherwise, long unstressed periods may be created. Therefore, for better results, the speed of the melody should also be taken into account when determining the rhythm pattern. Since such heuristics are not the main concern of the current project, we will leave that for future development.

Future Directions

There are many ways to improve the current system. The error (distance) function described above is certainly not good enough to cover a wide range of popular songs. A different error function may be devised so that a different "style" of determining rhythm can be achieved. For example, some music arrangers may determine that the rhythm pattern should complement the main melody instead of following it; in this case, the error function would be completely different from the one we have been using here. In any case, if we choose another error function, it is not complicated to modify the program and implement the change.

Multiple-valued patterns instead of binary ones as used in this article can also be implemented; that is, there could be other intermediate values between the absolute heavy and light beats, such as "not-so-heavy" and "not-so-light" beats. We can have several discrete values to represent them, or, to be more precise, continuous values can also be used to represent the "heaviness" of beats. When making the actual recording, appropriate instruments and strengths must be used to reflect the value of weight that is generated.

Another enhancement that can be carried out is to allow the pattern cycle period to vary. In the simplest case, as in our case, we are using one measure as the cycle period. For real-world applications, this may not be the best choice. The most appropriate cycle period would have to be determined by analyzing the given melody and considering other factors such as the speed of performance.

The presence of nonessential notes in melodies also introduces a complication. For better results, notes such as passing notes and auxiliary notes should not be included in the melody for rhythm-finding. In order to achieve this, a method must be devised to identify the nonessential notes within the melody. The need to identify these will become more apparent when chords are to be found to accompany the melody. To devise such a method is not an easy task, because it involves the contentious issue of identifying accents (Cooper and Meyer 1963).

The methods to implement these enhancements are now under investigation.

Conclusion

An algorithm for finding the rhythm of song melodies based on a dynamic programming approach has been designed and tested. There are several ways to improve this algorithm. In fact, rhythm-finding is only one of the steps toward automatic arrangement. Aside from this, there are many other aspects, such as automatic chord-finding and automatic orchestration, to be studied. It is hoped that more research will be encouraged in this interdisciplinary area in the near future. Interested readers are referred to Chafe, Mont-Reynaud, and Rush (1982) and to Hirschberg and Larmore (1987) for further reading in this area.

References

Bellman, R. E. 1957. *Dynamic Programming.* Princeton, New Jersey: Princeton University Press.

Chafe, C., B. Mont-Reynaud, and L. Rush. 1982. "Toward an Intelligent Editor of Digital Audio: Recognition of Musical Constructs." *Computer Music Journal* 6(1): 30–41. Reprinted in C. Roads, ed. 1989. *The Music Machine.* Cambridge: MIT Press.

Cooper, G., and L. B. Meyer. 1988. *The Rhythmic Structure of Music.* Chicago: University of Chicago Press.

Hirschberg, D. S., and L. L. Larmore. 1987. "The Set LCS Problem." *Algorithmica* 1:91–95.

Lerdahl, F., and R. Jackendoff. 1983. *A Generative Theory of Tonal Music.* Cambridge: MIT Press.

Appendix: The Dynamic Algorithm for Rhythm-finding

```
/*
   n ≡ the number of measures in the melody
   B ≡ bar length in minimum time units
   m ≡ maximum number of sections considered
   i, j, b, and k are indices used in loops
 */
for j=1 to n
  for b=1 to B
   compute sum(j, b)
  end for
end for
/*
   sum(j, b) is the prefix-sum up to and
   including bar(j, b)
```

$$\text{i.e., } sum(j, b) = \sum_{i=1}^{j} bar(i, b).$$

```
 */
for i=0 to n-1
  for j=i+1 to n
   compute segerr(i+1, j) and patt(i+1, j)
  end for
end for
/*
```

$$segerr(i+1, j) = \sum_{b=1}^{B} \left(\frac{j-i}{2} - \left| sum(j, b) - sum(i, b) - \frac{j-i}{2} \right| \right)$$

for $0 < i + 1 \leq j \leq n$,
and $= 0$ otherwise.

$$patt(i+1, j) \equiv (patt(i+1, j, 1), \ldots, patt(i+1, j, B)), \; patt(i+1, j, b) = \begin{Bmatrix} 0 \\ 2 \\ 1 \end{Bmatrix} \text{ if } \frac{sum(j, b) - sum(i, b)}{j - i} \begin{Bmatrix} < \\ = \\ > \end{Bmatrix} 0.5$$

```
*/

/* initialization */
for k=1 to m
/*
```

```
   m is the maximum number of sections to be
   considered; m ≤ n.
  */
  err(k, 1) = 0
  lastbrk(k, 1) = 0
  /*
   lastbrk(k, j) is one of the (k-1)th break-
   points (the one nearest to bar j, i.e., the
   one with the greatest value) to minimize
   err(k, j).
  */
end for

/* dynamic programming */
for j=1 to n
  err(1, j) = segerr(1, j)
  lastbrk(1, j) = 0
end for
for k=2 to m
  for j=2 to n
    $err(k, j) = \min_{0 \le i < j}\{err(k-1, i) + segerr(i+1, j)\}$
$lastbrk(k, j) = \{ii\ minimizes\ the\ value\ of\ err(k, j)\}$
  end for
end for

/* backtrack breakpoints */
for k=2 to m
  bp(k, k-1) = lastbrk(k, n)
  for i=k-1 downto 1
    /* k-1 breakpoints only */
    bp(k, i) = lastbrk(i+1, bp(k, i+1)-1)
  end for
end for
/*
   bp(k, i) is the ith optimal breakpoint when
   the melody is divided into k sections.
 */
/*
   Once we have the breakpoints, the patterns
   can simply be found by getting the values
   of the already calculated patt(i, j), where
   i and j are pairs of adjacent breakpoints.
 */
```