ICMC '85

On Finding Rhythmic Patterns in Musical Lines

Bernard Mont-Reynaud, Mark Goldstein

Center for Computer Research in Music and Acoustics (CCRMA)
Department of Music, Stanford University
Stanford, California 94305

ABSTRACT

Awareness of musical patterns is an important aspect of the perception of music by human listeners. The paper presents an attempt to capture some aspects of the perception of musical patterns in formal definitions and usable algorithms. Attention is restricted to a single musical line.

We begin with an abstract framework for the simple case of exact matches on a single dimension. Viewing a piece as a string of features, patterns are defined as substrings that recur. Algorithms for the efficient enumeration of patterns, and for the selection of preferred patterns are presented. For example, ABRACADABRA yields 9 patterns. The preferred patterns are ABRA and A. The other 7 patterns are implicit in ABRA.

Exact-match rhythmic pattern detection falls out directly of the abstract approach. Further improvements require introducing a model of rhythmic elaboration. Rewrite rules allow transformations such as matching a half note with a quarter and two eighths. The extended algorithm finds rhythmic skeletons which do not occur in the piece in exact form, but receive support by elaboration.

The arrangement of pattern instances provides strong clues for the metrical organization of a piece. Another application of pattern-finding methods to automatic transcription arise when when transcribing from sound. Segmentation errors may cause over-detection or under-detection of events, but the analysis of *near-misses* for existing patterns can point to spurious notes or missing notes resulting from such errors.

This work was supported by the National Science Foundation under Contracts NSF MCS-8012476 and DCR-8214350. Xerox Corporation has provided computer equipment for the research.

INTRODUCTION

Patterns play an important and ubiquitous role in the perception of music. Without any conscious effort, listeners detect many forms of redundancy in the music presented to them, and use the discovered patterns to make predictions about future events. The acquisition of new patterns in the course of listening to a piece is essential to this phenomenon, which seems closely related to the *unsupervised learning of structural descriptions* [7].

We were led to investigate musical patterns in the course of doing research on music recognition, towards the goal of automatic transcription of performed music [1, 2, 4, 8, 9, 10]. One of the reasons for using patterns in this context is that regularities found in the arrangement of pattern instances provide strong clues to the temporal organization of a piece.

The paper focuses on providing reasonably efficient algorithms for the identification of certain classes of musical patterns, with an emphasis on rhythmic patterns. Attention is restricted to a single musical line.

We first discuss an approach based on the exact matching of *uninterpreted* features. This abstract model deals, at least in principle, with patterns in any single dimension. Basic algorithms for the enumeration and selection of patterns are described for this simplified situation.

Of course, exact matching is a crude model. To introduce a more realistic notion of match, one needs to provide a topology of the set of features examined. Different dimensions (such as duration and pitch) seem to require different topologies.

Rhythmic pattern detection is first addressed with exact matching. To improve upon the exact matching of rhythmic values, we investigate pattern-matching methods that take rhythmic elaboration into account. A method is given to determine the greatest common ancestor of two compatible patterns. It is used to generate patterns whose support is only from elaboration.

Finally, we discuss some applications of musical pattern recognition to automatic transcription. Various statistics over the arrangement of patterns are used to gather clues for the metrical hierarchy of a piece, for tracking tempo, and for locating probable errors in the acoustic analysis. The latter application is based on the notion of *near-miss*, familiar in concept formation research.

EXACT MATCHING

The kind of pattern matching performed by a human listener relies on "soft" matches, in which sequences of features that are similar to each other may be treated as instances of the same pattern. It follows that a model based on exact equality among features of events is guaranteed to have limitations. However, since exact matching is much easier to deal with, it may provide a good place to start.

This section uses a rather abstract language, so that the concepts may later be applied to varied situations. We treat a *feature* as an uninterpreted symbol from some alphabet. Features may be compared for equality, but have no further meaning at this point. A *piece* is defined to be a sequence of features, that is, a string over the given alphabet. We call *pattern* a non-empty sequence of features that occurs more than once in the piece, that is, any repeated substring.

For example, consider the piece ABRACADABRA. The symbols A, B, C, D, R may stand for durations, pitches, musical intervals, or other musical features. By the definition above, there are 9 patterns in this piece. The pattern A occurs 5 times, and each of B, R, AB, BR, RA, ABR, BRA, and ABRA occur twice.

Because ABRA occurs twice in ABRACADABRA, substrings of ABRA necessarily occur *at least* twice, and if they do not occur *more than* twice, we might as well consider that they are *implicit* in ABRA. Thus, among the 9 patterns found, we will select ABRA and A as the two preferred patterns. Note that A is selected, in spite of the fact it is a substring of ABRA, because it also occurs independently.

Algorithms for enumeration and selection of patterns will now be described.

Pattern enumeration

Let p(1)p(2)...p(N) be a string of length N – the piece in which we seek to find repeated substrings. We use [I,J) to denote the range of indices from the I included to J excluded.

A simple approach is to examine each range [I,J) for $1 <= I < J <= N$ to see if the substring p(I)...p(J-1) occurs elsewhere in the piece. If a match p(I)...p(J-1) = p(K)...p(L-1) exists the string p(I)...p(J-1) is a pattern. However, this method of enumeration is inefficient, and requires further work to make sure each pattern is listed only once.

A better enumeration algorithm is based on growing patterns recursively. The recursive enumerator takes two arguments, a previously formed pattern P, and the set S of instances of P. It enumerates all the patterns which are extensions of pattern P. To enumerate all patterns, a single top-level is needed: recursion starts with the empty pattern (corresponding to the empty string) whose instances are at [I,I) for every I from 1 to N+1.

The recursive enumerator operates as follows. For each range [I,J) in S, ignoring a range with J = N+1, the symbol p(J) extends an instance of P to the right. Among the symbols thus generated from S, those that occur only once are removed, and duplicates are also removed. For each symbol E that remains (that is, exactly once for each symbol found more than once among the generated symbols) a new pattern P' = PE is obtained by extending P with the symbol E at the right. The instance set S' of P' consists of all the ranges [Im,Jm+1) such that [Im,Jm) is in S, p(Jm) = E, and Jm < N. Each time a new pattern P' with instance set S' is obtained, this fact is recorded in a global data structure, and the recursive enumerator is invoked recursively on P' and S' to enumerate the extensions of P' itself, if any.

In order to follow the algorithm in action, let us follow its operation on ABRACADABRA, at the point where the 5 instances of A have been found, and the enumerator is called again to extend the pattern A, it is passed an instance set containing [1,2), [4,5), [6,7), [8,9) and [11,12). The places for extensions have indices 2, 5, 7, 9 and 12, but position 12 is ignored (N = 11 here), so we have four extending symbols B, C, D, and B in positions following an A. Among these symbols, we remove C and D because they occur only once, and after removing duplications we are left with B. This causes the creation a new pattern P' = AB whose instance set S' contains [1,3) and [8,10). A recursive call of the enumerator to extend P' = AB produces similarly ABR and then ABRA, at which point recursion stops, as the minimum of 2 instances cannot be maintained. Recursion unwinds to the top level, from where the pattern B is found,

FIGURE 1
Mozart symphony in G minor (MINUET)

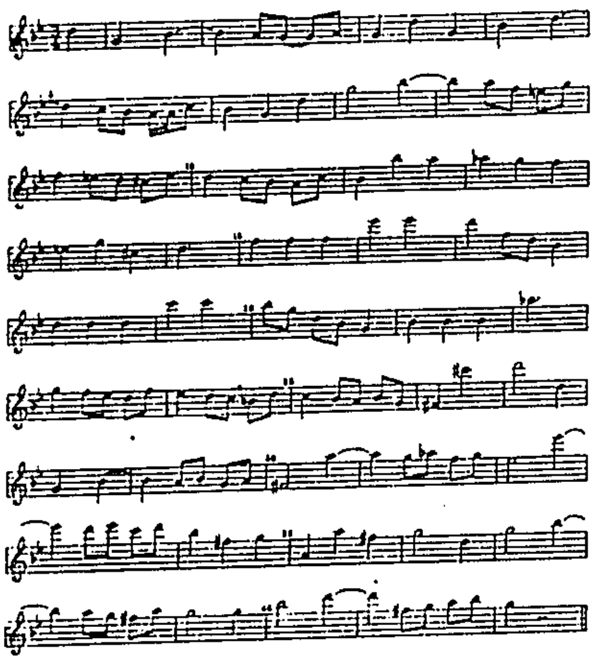

FIGURE 2
PATTERNS BY INSTANCE COUNT

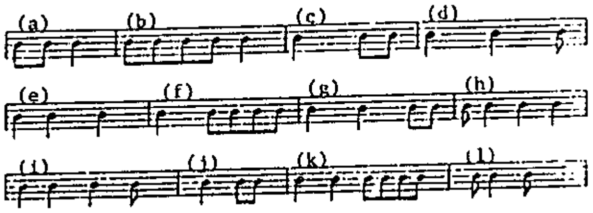

FIGURE 3
PATTERNS BY METRIC SPAN

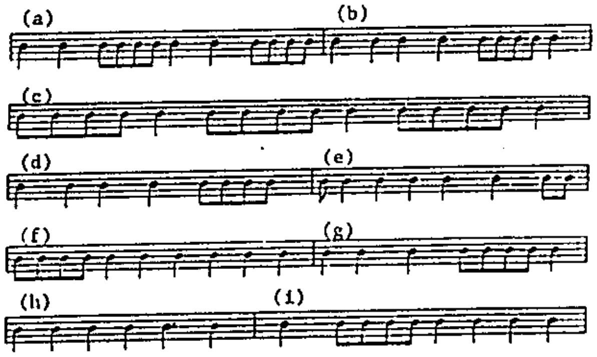

TABLE 1

PATTERN	INSTANCES	MEASURES
2a	10	8,9,10,17,20,23,24,25,33,41
2b	9	8,9,10,20,23,24,25,33,41
2c	8	4,7,19,28,30,32,37,40
2d	8	1,4,7,28,30,32,37,40
2e	7	0,3,6,26,27,36,39
2f	7	7,19,28,30,32,37,40
2g	7	4,7,28,30,32,37,40
2h	6	2,5,10,17,20,33
2i	6	0,3,6,27,36,39
2j	6	9,10,17,23,24,25
2k	6	7,28,30,32,37,40
2l	5	2,8,9,23,24

and expanded into BR and BRA, then R is found and expanded as RA.

Pattern subsumption

If a pattern P is a substring of a pattern P', we know that the number of instances of P must be *at least* equal to that of P'. If P actually occurs more often than P', it is hard to tell in general which to prefer, the longer pattern P' or more popular pattern P. But if P' occurs as often as P, the longer pattern P' is preferable for most purposes. We say that P is *subsumed* by P'. Eliminating subsumed patterns significantly reduces the number of patterns used, without causing a loss of information.

The implementation of this simple idea requires some care. To obtain an efficient algorithm, some observations must be made. First, if a pattern P of length N is subsumed at all, there must be a pattern P' of length exactly N+1 which subsumes P. Pattern P' has one of the forms PE or EP, where E is a symbol. (Note: it is possible for P to be subsumed both as a prefix, P'=PE, and as a suffix, P'=EP). The case P' = PE is easily detected during the recursive enumeration algorithm. When a new pattern P' is formed, if its instance set S' is as large as S, its *prefix* P is immediately known to be subsumed by P'.

The detection of the cases where P' = EP requires a separate pass over the patterns. To do this efficiently, we first arrange that the recursive enumerator, when forming a pattern P' = PE, takes note of the *prefix*relation between P' and P. Standard tree representation techniques may be used to store the information in the reverse direction. This is the *prefix tree*.

The *suffix* relations P' = EP may all be found in a single preorder traversal of the prefix tree, and it suffices to check the number of instances stored with P and P' to decide if P is subsumed. Efficiency-wise, the crucial observation is that "suffix of prefix equals prefix of suffix". For example, to locate the suffix pattern of ABRA (whose string is BRA) one goes from ABRA to its prefix ABR, and from there to the previously computed suffix BR. The desired BRA is then found by searching the patterns whose prefix is BR for a pattern whose last symbol equals the last symbol of ABRA.

Example: rhythmic patterns

To look for rhythmic patterns, one may simply run the basic algorithm, with note duration taken as the feature subjected to analysis. The specific notion of duration we prefer to use for many such analytical purposes is the metrical time until the next event. In other words, rests are combined with the duration of the preceding note; an eighth note followed by a quarter rest is treated as if it were a dotted-eighth note.

Figure 1 is a score that was transcribed automatically from a recorded performance. Readers familiar with the piece will notice three kinds of errors in this score: octave errors (cf. pickup note), missing notes (cf. bar 2), and spurious notes (bar 5, for example). These errors are due to imperfections of acoustic event detector.

Running the algorithm on the data in Figure 1 yields 41 patterns. Patterns consisting of one repeated note value, such as a series of quarter notes, are discovered by the algorithm, but are ignored in this analysis. Figure 2 shows the 12 most frequent patterns, sorted by their number of instances (including overlapped occurrences). Table 1 further describes these patterns.

Patterns 2b, 2e, and 2k are usually perceived as segments of the major thematic material in the Minuet, and they are among the more prominent patterns disccovered.

Figure 3 lists the patterns in order of their length, without regard to number of occurrences. Again, a portion of the main theme, 3g, appears as one of the longer patterns discovered.

Note that the sound segmentation errors that appear in this score cause a reduction in the overall quality of pattern detection. But the unfortunate fact that the piece appears less regular because of these errors can be turned around to advantage. Error-detection and even error-correction schemes based on musical patterns will be discussed later.

RHYTHMIC ELABORATION

In this section, we extend the method used to detect rhythmic patterns to take into account elaborative relationships between patterns and instances. This leads to algorithms to discover groups of related patterns, and their common elaborative sources.

Basic models

In general, we think of rhythmic elaboration as a transformation which adds note attacks while leaving existing attacks unaffected. In *unconstrained elaboration*, a note is replaced by a group of notes that sums to the same duration: R ==> R1 R2 ... Rk where the sum of the rationals Ri is R. For instance, (1/16 1/8 1/16 1/4) elaborates (1/4 1/4).

Grammar-driven elaboration uses grammatical rules to constrain the allowable transformations. Strictly speaking, a grammar specifies a set of non-terminals, terminal symbols, and rules. This means we need to introduce a separate non-terminal for each note value, such a Q for quarter-note, rules such as Q => 1/4 to relate the non-terminal to an equivalent terminal value, and rules such as Q => E E (where E is the non-terminal for an eight note) to express elaborations. It is clearly easier to simplify notation by treat rationals like 1/4 as if they were non-terminals as well as terminals. The duple elaboration rule for quartets now reads 1/4 => 1/8 1/8 which is more natural. But we still need a large number of rules, since every rhythmic value, taken as a non-terminal, needs its own set of rules.

Constrained elaboration is a compromise between the tight control of grammar-driven elaboration and the lack of control of unconstrained elaboration. It uses meta-rules which are schemes for dividing note values into parts (evenly or not). When the meta-rule x => x/2 x/2 is used, the effect is intended to be the same as if duple division rules had been added to a grammar, for every non-terminal x. For instance, (1/16 1/16 1/8 1/4) elaborates (1/4 1/4) by two uses of the duple division rule, but (1/16 1/8 1/16 1/4) does not elaborate (1/4 1/4) unless we have a meta-rule x => x/4 x/2 x/4 or a number of rules combining to the same effect.

Elaboration rules can be applied repeatedly to compound transformations. One can see that checking for unconstrained elaboration is a problem in rational arithmetic, while checking for grammar-driven elaboration is a parsing problem. The useful compromise presented by constrained elaboration calls for hybrid algorithms, that are partly linguistic and partly numerical.

Rhythmic intersection

The scheme described in the next section uses the solution to the following problem: Given two rhythmic patterns (with the same metrical span) decide whether they have a common ancestor by elaboration, and what the most specific such ancestor is.

In the unconstrained case, the problem is trivial. The common ancestor always exists, and is obtained by taking the *rhythmic intersection* of the two patterns, defined as follows. If both patterns are sounded at the same time, their rhythmic intersection consists of exactly those events which are sounded together. Thus, if each sequence of durations (d1, d2, ...) is viewed instead as a set of onset times (0, d1, d1+d2, ...), the rhythmic intersection of patterns corresponds to the usual set intersection for onset times.

For example, the intersection of (1/4 1/2 1/4) and (1/2 1/4 1/4) is (3/4 1/4). Note that if one pattern elaborates the other, the intersection is equal to the simpler of the two patterns. The intersection of two patterns is always an ancestor of both, via unconstrained elaboration, and is in fact their most specific common ancestor - their *min* with respect to the partial order induced by elaboration.

If we place grammatical constraints, any attempt to solve the greatest common ancestor problem in purely linguistic terms is confronted with great difficulties. The problem is, given a context-free grammar and two strings A and B, to find a maximal phrase C such that derives both A and B. Fortunately, all our grammar rules leave total metrical time invariant, and it follows that the linguistic greatest common ancestor, *when it exists*, is always identical to the rhythmic intersection of the two patterns.

An efficient algorithm is thus obtained as follows. The first step uses rational arithmetic to compute the unconstrained rhythmic intersection of the patterns. The second step checks that elaboration constraints hold between the computed intersection and both given patterns. Failing either check, the *constrained intersection* does not exist.

Skeleton generation

By computing the rhythmic intersection of two patterns (of the same metrical span) one may produce new patterns, called *rhythmic skeletons*, which do not explicitly appear in the piece - only their elaborations do. A skeleton inherits instances from all the base level patterns that elaborate it.

A series of experiments was carried out to construct a repetoire of skeletons which would hopefully derive many of the base level patterns. Initially, a skeleton is defined for each actual pattern of a given span. Then, in each pass of the algorithm, all pairs of skeletons are intersected together, creating a set of new skeletons, out of which any existing skeletons are removed. This process is guaranteed to converge. It is iterated until no new skeletons are generated, i.e., we have achieved closure under intersection.

If we apply this algorithm with unconstrained elaboration, it will often generate skeletons which do not make very good musical sense. Thus, it is best to bring grammatical constraints into play. Iterative closure was tried again with grammatical constraints. Most experiments were carried out using a very simple meta-grammar, with the two rules (1/2 1/2) and (1/3 1/3 1/3) for duple and triple subdivision.

The addition of a grammatical filter pruned out many poor skeletons, and produced many musically meaningful derivations. But the closure scheme so far has failed to meet our expectations. We plan to build the elaborative matches right into the enumeration algorithm, as a way to avoid some of the problems encountered.

APPLICATIONS TO TRANSCRIPTION

There are specific reasons for pursuing the question of musical patterns in the context of an automatic transcription system. Information derived from the arrangement of patterns can provide useful hints for solving a variety of problems. Two applications are discussed here.

A first area in which knowledge about patterns is helpful is the determination of metrical organization. One view of metrical organization that lends itself well to taking hints from a variety of sources is to use a graph in which nodes represent the important metrical spans of a piece. The shorter spans represented in the graph come from the durations of individual notes, but longer spans come from levels of grouping based on accents, on periodicity analysis, or on patterns.

Metrical span hints may be obtained from patterns by collecting simple statistics designed to reveal periodicities. If a pattern occurs at metrically regular intervals, the metrical distance from one instance to the next is a noteworthy period. If another pattern occurs at irregular intervals, the distances between instances wander about without pointing strongly to any period in particular. Thus, the computation used is to collect into a single histogram the metrical spans between successive instances of all patterns.

The statistics obtained for the Minuet example are as follows. The periodicities of one measure (3 quarters) and three measures are the strongest hints, each with a count of 29. Then comes the beat, with a count of 22, and the span of two measures, with a count of 19. Anything beyond this has a count of 8 or less, and may be safely ignored. The first four hints are very significant. It is reassuring to see that besides the beat and the bar, one gets the 2-bar and 3-bar levels of grouping which are both quite strong. Such hints, possibly combined with others derived from accent patterns, play a role in choosing the length of the bar.

Another interesting application of patterns arises when transcribing from sound. During the course of acoustic analysis, segmentation errors cause re-detection of certain notes, or the non-detection of others. Some of these errors may be noticeable on musical grounds, because the use of a particular pitch or duration is unexpected in a particular context, or because patterns suggest that the piece would seem to fall in place better if a certain change is made, like inserting or removing a note, or changing a pitch.

How does one measure the degree to which a particular event is expected in the context of its appearance? One simple answer is provided by patterns. Suppose we count, for each event, how many patterns it belongs in. These counts are not significant in absolute terms, but take meaning relative to neighbors. Clearly unexpected events (such as the two sixteenth notes in Figure 1, due to a spurious note detection) tend to get low counts. Their immediate neighbors also have somewhat lowered counts, but the effect do not spread very far. The ratio between a note's count and a moving average of the counts of neighbor events tends to fluctuate around 1.0 in normal circumstances. But the ratio drops close to 0 at unexpected events.

Thus, we now have a measure of expectedness based on participation in patterns. Low values point to least expected events. This gives a somewhat valuable hint, but only a negative one. A positive hint would also suggest how to fix the problem by adding, deleting or modifying an event. Positive hints may be obtained from pattern information by looking for *near-misses*.

A near-miss of a pattern arises when an instance has *almost* been matched, but there is a small, easily described difference between the instance as it is and what it needed to be to match successfully. This difference can be described as a suggested modification to the data that would make the specific pattern gain another instance. In the case of rhythmic patterns, the suggestions obtained in the near-miss detection mode take one of two forms: "delete this event" or "create an event at metrical position so-and-so". Of course, one is wise not to put too much trust in the suggestions of a single pattern. Wisdom cannot come from statistics, but if a suggestion comes up repeatedly when all the patterns are tried in turn, one starts to take it seriously, especially if the events involved have low on the "expectedness" measure defined above.

When testing the implementation of this scheme on the Minuet example, we found that indeed there is overwhelming evidence for (a) deleting the *second* sixteenth note in bar 5, turning the first sixteenth into an eighth; (b) adding the missing event in bar 2. One of the shortcoming of this scheme is that if the number of errors increase, at some point the algorithm begins to want to see more errors of similar types. Another difficulty, perhaps more important in practice, is that many changes are suggested where there really isn't an error. It is important to remember that these are only suggestions. When the "pattern expert" tells the "acoustic expert" to look for an undetected attack near beat 3 of bar 2, the latter must confirm the suggestion before the change is made in the score.

Since the uses of musical patterns in our application are always mediated by statistical filters, it is not so critical for the pattern identification methods to be able to rank patterns by their musical interest. This is fortunate, because doing the latter requires a level of musical intuition that seems difficult to capture in algorithms.

EPILOGUE

We have discussed several algorithms for musical pattern detection. The simplest ones handle exact matches only. This is a useful first step, whose generality is at first an advantage. Further refinements require that the topology needed to define softer matches be explicated for each musical dimension of interest, at the expense of generality.

The topology of rhythm patterns has been examined in some detail, and algorithms that match "modulo elaboration" have been described. Omitted from discussion were other important patterns, such as melodic patterns derived from the combination of duration and pitch. The role of dynamics and timbre has not been addressed, but it is clear that may enter into patterns. In fact, it would seem that anything that is worth calling a feature is also worth including in patterns. This includes features that are not direct properties of the sound, but are mostly implied by context, such as the metrical position of an attack, the harmonic function of a pitch, the position of phrase boundaries, and the type and strength of accentuation. Selecting the relevant dimensions for pattern formation and combining their topologies is one of many remaining challenges.

References

[1] C. Chafe, B. Mont-Reynaud and L. Rush, "Toward an Intelligent Editor of Digital Audio: Recognition of Musical Constructs," *Computer Music Journal,* vol. 6, no. 1, 1982.

[2] C. Chafe, D. Jaffe, K. Kashima, B. Mont-Reynaud and J. Smith, "Techniques for Note Identification in Polyphonic Music," *Proceedings ICMC 1985 (this volume),* also *Department of Music Technical Report STAN-M-30,* 1985.

[3] Cooper, G. and L. Meyer, *The Rhythmic Structure of Music,* University of Chicago Press, Chicago, 1960.

[4] S. Foster, J. Rockmore and W. Schloss. "Toward an Intelligent Editor of Digital Audio: Signal Processing Methods," *Computer Music Journal,* vol. 6, no. 1, 1982.

[5] D. Hofstader, "Analogies and Roles in Human and Machine Thinking," in *MetaMagical Themas,* Basic Books, New York, 1985.

[6] L. Meyer, *Emotion and Meaning in Music,* University of Chicago Press, Chicago, 1956.

[7] R. Michalsky, J. Carbonell, and T. Mitchell, eds. *Machine Learning,* Tioga Publishing Co., Palo Alto, 1983.

[8] B. Mont-Reynaud, et al. "Intelligent Systems for the Analysis of Digitized Acoustic Signals, Final Report.," *Department of Music Technical Report STAN-M-15.,* 1984.

[9] B. Mont-Reynaud, "Problem-Solving Strategies in a Music Transcription System," *Proceedings of the IJCAI,* 1985.

[10] A. Schloss, "On the Automatic Transcription of Percussive Music," Ph.D. Thesis, Department of Speech and Hearing, Stanford University, Stanford, California, *Department of Music Technical Report STAN-M-27,* 1985.