Following an Improvisation in Real Time*

Roger B. Dannenberg
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213

(412) 268-3827,
arpanet: Dannenberg@C.CS.CMU.EDU

Bernard Mont-Reynaud
CCRMA
Music Department
Stanford University
Stanford, CA 94305
(415) 723-4971,
BMR%CCRMA@SAIL.STANFORD.EDU

ABSTRACT. *A basic level of music understanding is required to recognize the correspondence between jazz improvisations and their underlying chord progressions. In this study, techniques are examined for building a computer system that can recognize and follow a jazz solo by first deducing where beats fall in relation to the solo, and then by using a statistical matching method to find the most likely relationship to a chord progression.*

1. Introduction

One of the most challenging aspects of advanced performance systems is the necessity to analyze inputs from human performers, improvisers and conductors. To make intelligent decisions in a live performance, a system must recognize abstract features of captured human gestures. On the other hand, to be effective in a real-time situation, it must be simple enough to be amenable to some form of incremental processing.

After briefly delineating areas in which real time programming may be compatible with music understanding, the paper focuses on the concrete problem of following a traditional jazz improvisation against a known chord structure.

1.1. Real-time Understanding of Music?

In humans, the formation of appropriate responses to musical stimuli draw upon a vast and complex network of concepts, learned skills and innate behaviors that presents a formidable challenge to precise analysis. Refined, specialized skills are required of composers, conductors, and performers, but considerable portions of their musical competence are equally found in "naive" listeners, and even these non-experts don't seem so naive when one looks more closely at their behavior. A full understanding of a piece of music may require both creativity and extensive training, but it also seems fair to say that a broadly available ability to perceive patterns at various levels of organization lies close to the heart of the matter. At any rate, it seems that music understanding relies more heavily on pattern recognition capabilities, and on hierarchical grouping, than it does on logical reasoning and problem-solving techniques. It is possible that an emphasis on the latter is precisely the privilege of musical experts, who are able to name, and to reason about, entities which naive listeners also perceive, for the most part, but are unable to put in words.

Computers should not be expected to understand music the way humans do. But it is possible, desirable and instructive to build systems that attempt to deal with specific aspects of musical intelligence. By becoming more aware of both the possibilities and the limitations of such systems, one may learn how to let people do what they do best (and/or enjoy the most) while gradually raising the level of the computer's involvement. Using this cooperative approach, we can foresee that even limited or imperfect music understanding capabilities can play a role in computerized tools for musicians, and present new opportunities for composers.

As part of the general evolution of computer systems toward intelligence, interactivity and a fuller use of human sensory channels, the musical domain offers a fascinating array of challenges. Among these

*Published as: Roger B. Dannenberg and Bernard Mont-Reynaud, "Following an Improvisation in Real Time," in *Proceedings of the International Computer Music Conference,* Computer Music Association, (August 1987), pp. 241-248.

figures the question of real time implementations of musical reasoning -- which may be viewed as the construction of intelligent performer's assistants.

Naturally, many deep, analytical aspects of music understanding are best approached by global techniques that exclude real time implementations. However, intuitive and almost primitive aspects of the music listening experience, such as the acquisition of a sense of tempo and beat, or the feeling for rhythmic/metrical texture, tonal color and harmonic tension, are fundamentally time-varying "impressions" for which it seems reasonable to seek real-time models.

1.2. The Jazz Improvization Problem

On-line pattern matching techniques can be used to follow a live performance of a given score [4, 1]; however, jazz solo improvisation calls for much variation between performances, excluding the use of techniques based on direct score matching. Nevertheless, educated listeners seem able to perceive enough structure in a solo to relate it to an underlying chord progression. Can a computer program capture some essential aspects of this task?

The practical goal of this effort is to produce a program that listens to an improvised solo and joins in the performance once the tempo and location are deduced. We assume that the player follows an underlying chord progression which is specified in advance to the computer. To limit the scope of the problem further, we have only considered 12-bar blues progressions. This immediately raises questions: what styles of jazz are allowed, and what does it mean to follow a chord progression? There are no simple answers to these questions. In general, the quality of system will be judged by how well it deals with variations in style and by how directly the underlying chord progression must be implied by the soloist.

1.3. Outline of Our Approach

The first author has devised a matching method that uses a set of training performances to learn some relevant statistical patterns, and then uses a maximum likelihood technique to locate a soloist within a given chord progression. In both the training sessions and the performance, the matcher is told when each beat occurs.

In order to form a complete follower, the matcher needs a "foot tapper" module that deduces beat timing. The second author had previously developed techniques which extract tempo directly from a performance, without access to a score, by analyzing patterns of note onset times. Adapting the earlier methods to a real-time situation, he has developed an on-line beat recognition method, based on time-varying statistical clustering, that addresses the needs of the current problem, and presumably some others.

These two parts of the approach are describe more fully in the next two sections, which are followed by a discussion of our results and some concluding remarks.

2. Following the Chord Progression

Improvised solos are highly unpredictable, and thus pose a real challenge to analysis and understanding. In trying to follow an implied chord progression, one approach is to look for a set of rules that relate pitches or pitch sequences to underlying chords. An example, due to Peter Jansen assumes certain intervals are never played relative to the root of a chord. The corollary is that given a pitch, we can rule out certain chords. In many cases, after a short sequence of pitches, all but one chord is eliminated. This is especially true in the case of 3-chord, 12-bar blues, since there are only 3 choices of chords in the first place. This technique actually worked very well on some test data, but we found that it breaks down when the least bit of chromaticism is allowed in the solo.

After basically positive although limited results with a rule-based approach, better rules were sought. Some data was collected and studied for patterns in the pitches of specific intervals (e.g. a half step might

imply a leading tone), pitches of agogic accents, and other features. It was hoped that a set of features could be found that would serve to bring reliable information into focus.

Surprisingly, no reliable rules were found. Although we beleive further investigation of this approach is warranted, we turned our attention to a more statistical approach. Rather than looking for a few pitches in which information is concentrated, we attempt to gather a small amount of information from each and every pitch of the solo.

2.1. The "Correlation" Technique

The statistical method assumes that at different beats within a 12-bar blues progression, different pitches will predominate. In other words, at beat 1 we expect to see a certain probability distribution of pitches which may be entirely different from the probability distribution at beat 20. For our experiments with blues progressions, we wrote software to collect data from actual solos to tell us, for each eighth note in the 12 bar progression, what is the probability of each pitch-class. The data is stored in a 96 (eighth notes) by 12 (pitch classes) matrix.

With the data, we can see how well a solo corresponds to the measured distribution by a simple algorithm: take the product of the probability of each note in the solo, where the probability is the value of matrix element corresponding to the pitch class and location of the solo note.

More formally, the computation is

$$\prod_{b=0}^{n-1} m_{b \bmod 96, s_b}$$

where $m_{b,p}$ is the probability of pitch class p on (eighth-note) beat b, where s_b is the pitch class played during beat b, and where n is the length of the solo in eighth-note beats.

The actual implementation differs slightly from this mathematical description. Rather than storing estimated probabilities in m, cummulative times are stored. Thus, if a C (pitch class 0) is played during beat 10 on 6 repetitions of the chord progression (choruses) and the length of the eighth note is 0.25 seconds, then $m_{10,0}$ would be 1.5. In practice, no solo is going to make pitch changes exactly on the beat or hold notes through the beat. Therefore, the values in m reflect fractions of beats. If a C is played for only 0.05 seconds of beat 10, then that contributes 0.05 to the value of $m_{10,0}$. Finally, even though probability theory tells us that we should multiply probabilities here, it was discovered that addition works just as well! The result is no longer a number between 0 and 1, but here we are only interested in relative rather than absolute likelihood.

This method for computing the likelihood of a solo given a distribution can be used to determine how the solo relates to the underlying chord progression. Let

$$L_i = \sum_{b=0}^{n-1} m_{(b+i) \bmod 96, s_b}.$$

Notice in this formula that the first subscript of m is shifted by i. This effectively "shifts" the probability table m as in a correlation computation. L_i estimates the likelihood that solo s starts on beat i of the progression. Figure 2-1 plots a typical L_i for i ranging from 0 to 100. Note that L_i is periodic with a period length of 96. (The redundant points were plotted to show the peak more clearly.) This graph clearly shows that the best correspondence between the solo and m occurs at offset 0 (or 96). Since the second half of every 4 bars returns to the tonic it is not surprising to see smaller peaks every 4 bars in the graph.

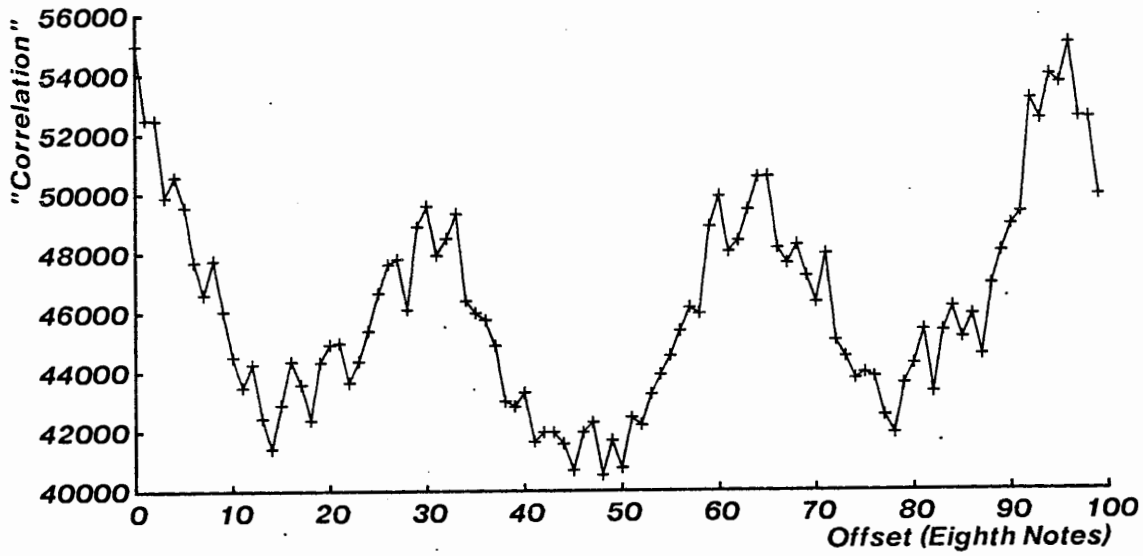


Figure 2-1: Correspondence between solo and the probability distribution at different offsets.

2.2. Real Time Strategies

Since our goal is to follow an improvisation in real time, we want a likelihood estimate $L_{i,b}$ which is updated on each new beat *b*. To track the improvisation, we want to weight recent data more heavily than older data, so some sort of windowing strategy must be used.

Exponential and square windows are attractive because they allow $L_{i,b}$ to be computed incrementally. For the exponential window, we have $L_{i,b} = cL_{i,b-1} + m_{(b+i) \bmod 96, s_b}$, where *c* is a decay constant slightly less than 1. For a square window, we have $L_{i,b} = L_{i,b-1} + m_{(b+i) \bmod 96, s_b} - L_{i,b-w}$, where *w* is the width of the window. Our experiments have shown that the exponential formulation does not work well, probably because it uses the probability table *m* non-uniformly. A square window with width 96, the size of the table *m*, appears to work better.

We are currently implementing a real-time follower based on this strategy. At any given beat, the maximum value for $0 < i \leq 95$ of $L_{i,b}$ should indicate where the soloist is relative to the 12-bar blues chord progression. The follower will use beat information from another module which is described in the next section.

3. Following the Beat

In order to follow a blues solo, one needs a "beat follower" capable of counting, say, 8th notes in approximate synchrony with the performance. If such a "tracking metronome" can be designed, it will provide a handle on a closely related family of tasks woven around tempo tracking.

We should note the relation between foot-tapping, a naive skill that operates on the fly, and music transcription, an expert skill which requires multiple passes in all but trivial examples. Intuitive tapping provides an initial and approximate time-map which a transcriber must then re-examine, gaining robustness at the expense of real-time operation, by bringing in a more global view of a piece, section or phrase. A mature transcription system ought to be sensitive to most important aspects of musical context. Such a degree of musical understanding cannot be expected of a real-time foot-tapper, as discussed in the rest of this section. The reader is referred to [5, 6, 3, 2]for further discussions of music transcription.

3.1. Why is Beat Tracking a Problem?

For some musical styles, beat tracking is not an issue, but we are concerned here with the common situation where the precision and regularity of occurrence of musical events cannot be counted on with any certainty.

The first difficulty comes from inexact (non-mechanical) timing, possibly resulting from expressive performance, sloppy performance, or inaccurate collection of timing data. A quasi-periodic sequence of onset times can be smoothed by treating the successive durations as the sum of a relatively slowly-moving beat duration, and uninterpreted "timing jitter". (Note: in this paper, we ignore note releases, and call "durations" the differences between onset times in a single melody. Square windows, which require a memory buffer, and exponential windows, which do not, are the two averaging techniques most adapted to real-time implementation.

The second difficulty is that the sequence of onset times is not even quasi-periodic. Even though there are passages of running eighth notes, most music is made of a mixture of note values. Thus, one must quantize durations by rounding to an integer multiple of a specific metrical unit. The simplest approach is to keep the quantization level constant throughout. But a good transcriber doesn't, and perhaps an intelligent real-time beat tracker should not necessarily stick to a fixed level of quantization. If the level is too large, e.g. quarter notes, the frequent use of finer divisions (8ths) may confuse an adaptive tracker that seeks to readjust its tempo continuously. If the level is too small, e.g. 16th notes, the risk of rounding to the wrong side gets very high. The level of 8th notes has been selected as the correct compromise unit for quantization in the present context, but the inherent ambiguity involved in quantization remains an intrinsic characteristic of the problem.

There are further sources of difficulty, including grace notes, which do not fall squarely within a metrical framework, and notes that result from artifacts of data collection, such as spurious firings of a front-end pitch detector. Dealing with different difficulty types requires different techniques which each pull in their own directions, and the true design challenge for a beat tracker is to find a proper balance between the various defensive measures.

3.2. A Simple Real-time Foot-Tapper

The following beat tracking method has been devised after a certain amount of experimentation, and considering several more complex schemes, including multiple parallel trackers. First, we choose to track 8th notes. A different choice might be made for a differnt musical style.

The state of a beat tracker is represented by five numbers (T, B, S, W, D) as follows. The real T and integer B represent the time, in seconds, and metrical position, in beats, of the most recent established position. The reals S and W yield the current beat value D = S/W. The reason we use a homogeneous coordinate system for the current beat value, rather than a single number, is that the two numbers S and W allow one to keep running statistics: S is the running sum of beat durations, and W is a running weight. We actually use a redundant representation where the exact invariant is S = W * D. (The number S could be omitted from the state)

The beat tracker, or predictor, in the state (T, B, S, W, D) is essentially a way to expect, or predict, events at metrical times B+1, B+2, ... and physical times T+D, T+2*D, ..., where D = S/W. There are two main algorithms, one to create a predictor and install its initial state, and the other to adjust the predictor so it keeps tracking notes as they come. Any additional processes which need to be synchronized with the beat tracker, can simply schedule themselves out of the predictor state, which is globally available. This allows one to perform particular actions on every beat, or at fractions or multiples of beats, but this is independent of the tracker proper.

We need some terminology. We say that D1 is much shorter than D2, denoted D1 << D2, if (D1 * a + b) < D2. Here, a is a constant larger than 1, such as 1.1, and b is a small duration, such as .1 sec. We say that D1 is roughly equal to D2 (D1 == D2) if neither D1 << D2 or D1 >> D2 holds. We say that a note is "weak" if it is shorter than some absolute threshold (say .05 sec) or if it is much shorter than the

preceding note. We say that a note is "healthy" if it isn't weak. We say that a note is "accented" if it strong, and it is either the first note, or if the preceding note is substantially shorter than the note itself. We now describe the two main algorithms, which respectively create a predictor, and update its state.

The method used for creating a predictor is to notice a "rhythmic alignment" i.e. a succession of 3 healthy notes N1 N2 N3 whose onset times T1 T2 T3 are in quasi-arithmetic progression, i.e. (T2-T1) == (T3-T2). The relation of N1 to N2, and N2 to N3, is that N(i+1) is either the note immediately following N(i), or the first accented note following N(i). Among the possible choices of N1 N2 N3 one uses the first that succeeds, as note are heard.

Since metrical time does not start counting until such an alignment is found (a first sense of beat) we take B1 to be 0, but this is arbitrary. The state is initialized at (T1, B1=0, S=0.0, W=0.0, D) and the only non-obvious choice is that of D, since S/W = 0.0/0.0 is currently indeterminate. The choice of the initial beat is made by taking the average A of the two almost equal durations (T2 - T1) and (T3 - T2) and bringing it back to near the range of an expected eight note, E, as follows: D = A / round(A / E). (If A < E/2 so that round(A/E) = 0, we do not consider that we have an alignment). For example, if the first two notes we see have durations .4 and .46, with the average .43, and E is .20, we start with D = .43/2, which amounts to saying that we treat the first two notes as quarter notes. A variety of other strategies could be designed for the initialization of a predictor, or to eliminate the (mild) dependence upon the "typical eighth note" E, but the topic will not be considered further.

The "update" method is quite critical. It amounts to noticing a new note, and adjusting tempo accordingly. After we notice the alignment N1 N2 N3, we set up a new predictor at N1, to notice N2 and then N3. Thereafter, the predictor is updated on every "healthy" note. If (T, B, W, S, D) is the current state, and T' is the onset time of the new healthy note, let:

$\Delta T = T' - T$
$\Delta B = \Delta T / D$
DELTA = abs(ΔB - round(ΔB))
CONFIDENCE = 1 - 2 * DELTA
ΔB = round(ΔB)
DECAY = .9 ** (- ΔB)

Then the state is updated as follows:

T = T + ΔT i.e. T'
B = B + ΔB
W = W * DECAY + ΔB * CONFIDENCE
S = S * DECAY + ΔT * CONFIDENCE

Note that DECAY implements an exponential window over past values, at the rate of .9 per beat. Also note that the confidence varies between 0 and 1 as the rounding error varies between its minimum and maximum values. This de-emphasizes ambiguous matches in such a way that they do not change the beat value much at all.

4. Results

Overall, the current performance of the chord follower is encouraging but not high. Ideally, assuming the solo starts at beat 0 of the chord progression, $L_{0,b}$ will be greater than any other $L_{i,b}$ for $1<i<96$. If this were dependable, we could reliably follow the soloist. Figure 2-1 illustrates a case where this indeed happens. The peak at zero correctly predicts the location of the soloist, but this result was obtained without windowing.

In practice, the likelihood estimate of the "correct" answer ($L_{0,b}$) using a 12 measure (96 eighth note) square window is virtually always one of the 10 largest likelihood values, but it is the maximum likelihood only about 70% of the time. More research is needed to improve upon this figure.

The results of the beat follower are shown in Figs 3.1 - 3.5 (see captions).

5. Summary and Conclusions

The chord progression follower introduces a new method for matching an improvised solo against a given chord structure. The method is designed to tell us where the soloist is playing in relation to the chords, and it also gives an indication of how closely pitch choices correspond to an expected pitch distribution.

The beat follower has not received sufficient experimentation to evaluate its full potential. It does appears to offer a good starting point for building a beat tracker with a broad range of applicability, which could solve an important outstanding problem in performance following.

These modules are intended to work together to implement a rudimentary understanding of a jazz solo. They could be used to drive an intelligent accompaniment system or used to assist in further analysis and understanding.

In conclusion, we offer some new techniques for real-time analysis of improvised music with the goal of following a soloist playing according to a fixed chord progression. We claim that programs based on these techniques can exhibit at least a small degree of music understanding, consistent with real time constraints. In the future, intelligent programs will extend the role of the computer in music and enrich the possibilities for combined human and computer performance.

Systems that we have built [5, 6, 3, 2, 4, 1] have pointed in this direction. By continuing to build systems like these and the one described in this paper, we accumulate knowledge and experience that guides us along this evolutionary path.

References

1. Joshua J. Bloch and Roger B. Dannenberg. Real-Time Computer Accompaniment of Keyboard Performances. Proceedings of the 1985 International Computer Music Conference, Computer Music Association, 1985, pp. 279-290.

2. Chris Chafe, Bernard Mont-Reynaud, and Loren Rush. "Toward an Intelligent Editor of Digital Audio: Recognition of Musical Constructs". *Computer Music Journal 6*, 1 (Spring 1982), 30-41.

3. John Chowning, Bernard Mont-Reynaud, et. al. An Intelligent System for the Analysis of Digitized Acoustic Signals. Tech. Rept. STAN-M-15, Stanford University, 1984.

4. Roger B. Dannenberg. An On-Line Algorithm for Real-Time Accompaniment. Proceedings of the 1984 International Computer Music Conference, Computer Music Association, 1984, pp. 193-198.

5. Bernard Mont-Reynaud. Problem-solving Strategies in a Music Transcription System. IJCAI Proceedings, 1985.

6. Bernard Mont-Reynaud and Mark Goldstein. On Finding Rhythmic Patterns in Musical Lines. Proceedings of the 1985 International Computer Music Conference, Computer Music Association, 1985, pp. 391-397.

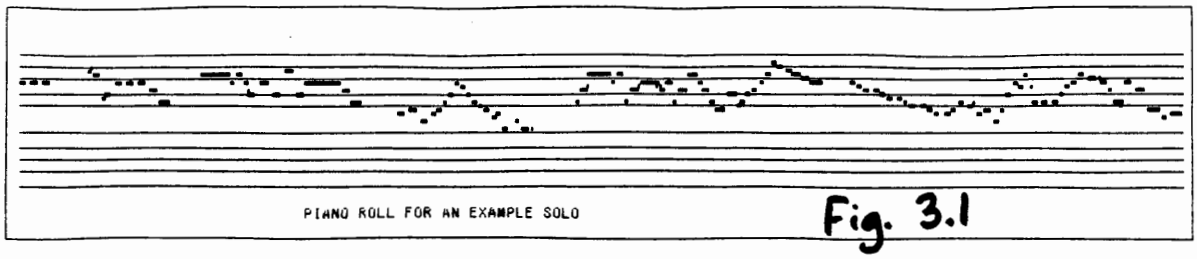
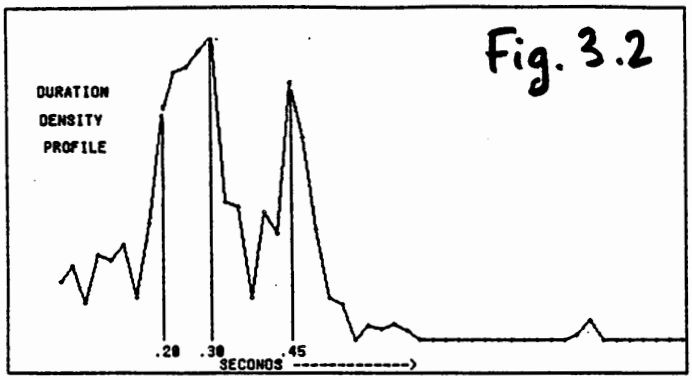
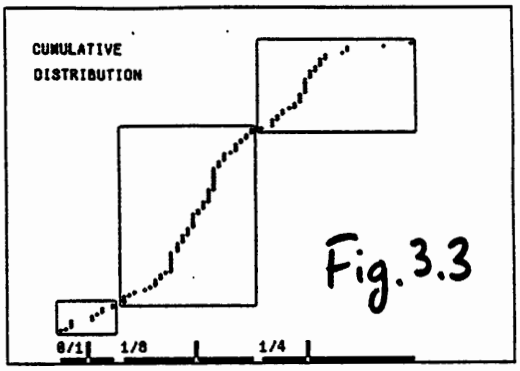
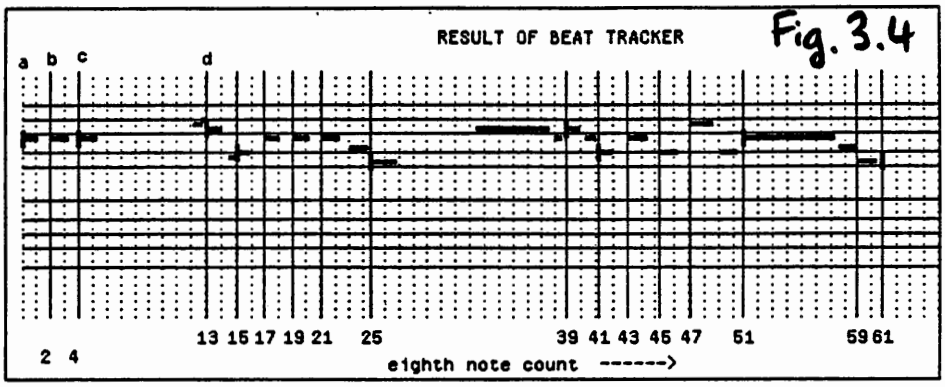
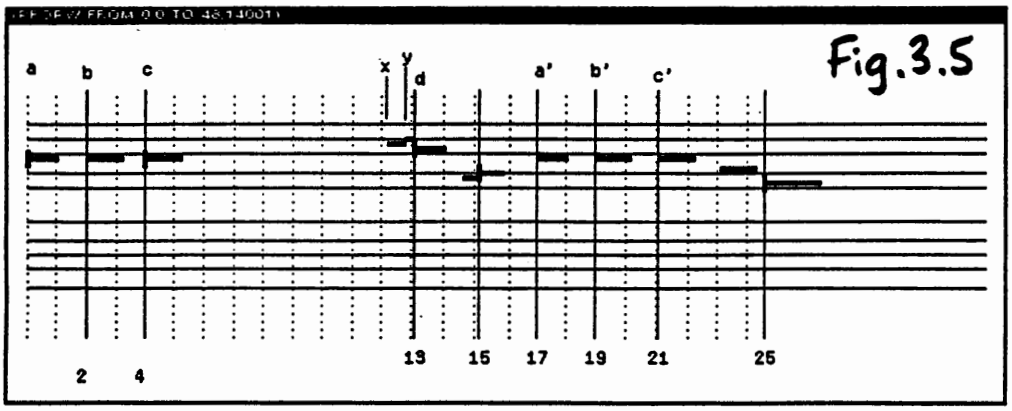

PIANO ROLL FOR AN EXAMPLE SOLO

Fig. 3.1


DURATION
DENSITY
PROFILE
.20
.30
.45
SECONDS ---------->
Fig. 3.2


CUMULATIVE
DISTRIBUTION
0/1
1/8
1/4
Fig. 3.3


RESULT OF BEAT TRACKER
Fig. 3.4
a b c d
13 15 17 19 21 25 39 41 43 45 47 51 59 61
2 4
eighth note count ------>


a b c x y d a' b' c'
Fig. 3.5
13 15 17 19 21 25
2 4