音響信号に対するリアルタイム ビートトラッキングシステム

— 打楽器音を含まない音楽に対するビートトラッキング —

後藤 真孝　　村岡 洋一

早稲田大学 理工学部

{goto, muraoka}@muraoka.info.waseda.ac.jp

あらまし　本稿では，打楽器音を含まない音楽音響信号に対してリアルタイムにビートを認識するビートトラッキングシステムについて述べる．従来研究の多くはMIDI信号か打楽器音を含む複数の楽器による音響信号を対象にしており，打楽器音を含まない音響信号をリアルタイムに扱うことはできなかった．本研究では，音源分離が困難な音響信号に対して音楽的判断をするために，コード名にシンボル化しなくてもコード変化を検出できる手法を提案する．これにより，各ビートが強拍か弱拍かを判定でき，複数のエージェントによるビートの様々な解釈の中から適切なものを選択できる．並列計算機AP1000上に実装して実験した結果，市販のロック・ポップスのCDからサンプリングした音響信号を扱えることを確認した．

A Real-time Beat Tracking System for Audio Signals

— Beat Tracking for Music without Drum-sounds —

Masataka Goto　　Yoichi Muraoka

School of Science and Engineering, Waseda University
3-4-1 Ohkubo Shinjuku-ku, Tokyo 169, JAPAN.

Abstract　This paper presents a real-time beat tracking system that recognizes temporal positions of beats in musical audio signals without drum-sounds. Most previous systems dealt with MIDI signals and were not able to process, in real time, audio signals containing sounds of various instruments without drums. To make musical decisions for audio signals, we propose a method of detecting chord changes without symbolizing any chord names. The method enables the system to determine whether a beat is strong or weak, and to select the best hypothesis from various agent-generated hypotheses of beat positions. Our experimental results show that the system is robust enough to handle audio signals sampled from compact discs of popular songs.

1　はじめに

本研究では，人間が音楽に合わせて手拍子を打つように曲のビート(四分音符)の位置をリアルタイムに認識するビートトラッキングシステムを実現する．ビートトラッキングは様々なアプリケーションにおいて有効なだけでなく[1, 2]，音響信号に対する音楽聴取過程を計算機上で実現する第一段階としても重要である．西洋音楽において，ビートは曲を構成する様々な時間的概念の基本単位であり，音楽を理解する上で欠かすことのできない要素の一つである．たとえすべての楽音を完全に音源分離して同定できない人でも，音楽に合わせて手拍子を打つことは比較的容易にできる．そこで我々は，まずビートの知覚を計算機上で実現した後に，より高次の音楽構造を理解する方向へと研究を進めていく．そして本研究は最終的に，計算機がリアルタイムにどこまで音楽音響信号を認識・理解できるかを探求することを目的とする．

従来のビートトラッキングに関する研究の多くは，MIDI信号や発音時刻の系列を対象にしていた[3, 4, 5, 6, 7, 8, 9]．しかし，複数の楽器音が含まれた音響信号をMIDI信号に変換するのは非常に困難なため，

我々が普段 CD を再生して聞くような音響信号を入力とすることはできなかった．独奏や少数の楽器で演奏された音響信号を対象にした研究も報告されているが [10, 11]，自動採譜の一部として研究されているものが中心でリアルタイムに処理できなかった．一方，我々はこれまで，打楽器音を含む複数の楽器で演奏されたロック・ポップスの音響信号を対象にした，リアルタイムビートトラッキングシステムを実現してきた [1, 12, 13, 2]．しかし，このシステムは打楽器音 (特にバスドラムとスネアドラム) を含まない曲を扱うことはできなかった．

本稿では，打楽器音を含む音楽を対象にした従来のシステムをどのように拡張すれば，打楽器音[1]を含まない音楽を対象にしたシステムを実現できるかについて述べる．本システムは，市販の CD などから得た複数の楽器音を含む音響信号 (対象テンポ: 61〜120 M.M.) を入力とし，四分音符に相当するビートの存在する時刻 (**ビート時刻**) をリアルタイムに認識する．このビート時刻の系列を四分音符レベルと呼ぶ．本システムはさらに高次の音楽構造として，入力曲が 4/4 拍子であることを前提にそのビートが強拍か弱拍か (**ビートタイプ**)[2]も判断する．これは，二分音符レベルでビートを理解していることに相当する．

本システムでは，1) 各ビートのビートタイプ (強拍か弱拍か) を判定し，2) ビートの様々な解釈の可能性の中から適切なものを選択するために，音楽的判断をおこなう必要がある．そこで，音源分離が困難な音響信号に対してこのような判断をするために，周波数スペクトルからコード名にシンボル化しなくても直接コードの変化だけを検出できる手法を提案する．この検出結果を利用すれば，「コードは小節の頭でより変わりやすい」という音楽的知識により 1) の判定ができ，「コードはビートの位置でより変わりやすい」という音楽的知識により 2) の選択ができる．

計算量が多い処理をリアルタイムにおこなうために，並列計算機 AP1000 上に本システムを実装した．市販のロック・ポップスの CD からサンプリングした音響信号を入力して実験した結果，40 曲中 34 曲に対して正しいビートをリアルタイムに出力できた．

2　実現上の課題と解決法

音楽音響信号に対するビートトラッキングを実現するには，主に次の三つの課題がある．

1. 音響信号中の特徴とビートが直接対応しない．
   ビートは音楽に対して人間が知覚する概念であり，音響信号がないところにビートが存在する場合もある．また，音響信号波形にはビートに直接関係のないエネルギーピークが多数あるため，ピーク検出した後に閾値処理をする単純な手法では，ビートをとらえることはできない．そこで，「音響信号から何を手がかりにどのようにビートを見つけるか?」を解決する必要がある．
2. ビートの様々な解釈の可能性がある．
   ビートを解釈する際には，周波数解析によって得られた複数の異なった手がかりがビートに対応する可能性があったり，いくつかの異なるビートの間隔が候補に挙がるなど，様々な曖昧な状況がある．そのため，ビートを一意に解釈することはできず，「複数の解釈の可能性をどう調べるか?」を考える必要がある．
3. ビートを音楽的に判断する必要がある．
   ビートタイプ (強拍か弱拍か) を判定したり，ビートの様々な解釈の可能性の中でどの解釈が最も適切であるかを判断するには，その音楽的状況を考慮して判断する必要がある．そこで，この判断のために「どのような音楽的知識を用いるか?」を検討する必要がある．

本研究では，これらの課題をこれまで以下のように解決してきた [1, 14, 12, 13, 2]．

2.1　発音時刻系列の自己相関・相互相関

「音響信号から何を手がかりにどのようにビートを見つけるか?」に対しては，発音時刻 (各音が鳴り始める時刻) を手がかりとして，その系列の自己相関関数・相互相関関数を用いてビートを求めた (図 1) [13, 2]．

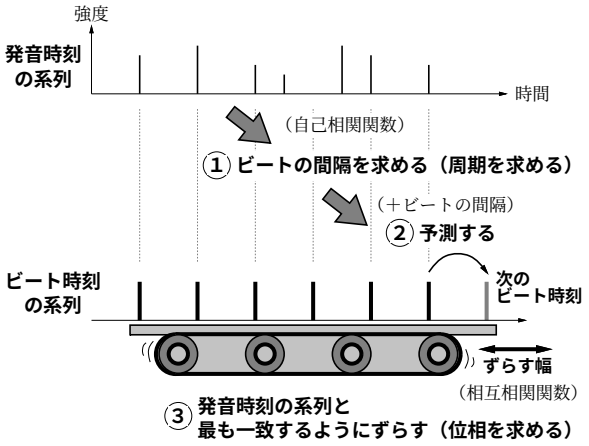


図 1: 発音時刻系列の自己相関・相互相関

[1] 本稿ではバスドラムやスネアドラムに代表されるドラムスを想定する．

[2] 各小節において 1 拍目と 3 拍目を強拍，2 拍目と 4 拍目を弱拍と定義する．

ビートをトラッキングする問題は，その**周期** (ビートの間隔) と**位相** (ある参照点からの相対位置) を求める問題と捉え直すことができる．まず周期を求めるために，発音時刻系列の自己相関関数を計算し，テンポから決まる妥当な時間の範囲内[3]のピークを周期とする．これは，頻繁に出てくる発音時刻の間隔は，多くの場合その曲における基本的な音符の長さなので，周期 (四分音符の長さ) に相当している可能性が高いからである．一方位相は，こうして得られた周期を持つビート時刻の系列と発音時刻系列との相互相関関数のピークから求める．

2.2 マルチエージェントアーキテクチャ

「複数の解釈の可能性をどう調べるか?」に対しては，異なった戦略で様々なビートの解釈をおこなう複数のエージェントを導入し，解釈の可能性を並列に調べた (図 2) [1, 14, 12].

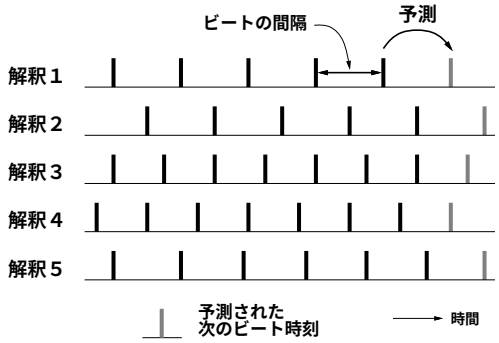


図 2: マルチエージェントによる複数のビートの解釈

各エージェントは相互作用，自己評価，適応の能力を持ち [12]，発音時刻系列に基づいて一つの解釈を生成する．そして，その解釈がどれくらい適切であるかを解釈の確信度として自己評価する．最終的にシステムが出力するビートの情報は，全エージェントの中で最も確信度の高い解釈に基づいて決定する．これにより，あるエージェントの解釈がはずれても，他のエージェントが正しく解釈している限り，ビートを見失わずにトラッキングできる．

2.3 二種類の音楽的知識に基づく判断

「どのような音楽的知識を用いるか?」に対しては，a) ビートと発音時刻との関係，b) ロックやポップスにおける典型的なドラムパターン，の二種類の音楽的知識を導入して解決した [12, 13, 2].

a) の関係として,「ビートの位置で音が鳴ることが多い」という知識を用いる．これにより，発音時刻と予測したビート時刻が一致するほどエージェントは解釈の確信度を上げる．b) はドラム音 (バスドラムとスネアドラム) のパターンとして事前に登録しておき，これらを検出したドラム音の発音時刻と照合することで，最も一致するドラムパターンを求める．そして，これがよく一致したときにエージェントは解釈の確信度を上げ，一致したパターンの開始時刻を強拍の位置とみなしてビートタイプ (強拍か弱拍か) を判定する．

3 打楽器音が含まれない場合の問題点と解決法

上記の解決法を実装した我々の従来システムでは，打楽器音を含む音楽に対しては適切にビートトラッキングできるが，打楽器音が含まれない場合には以下のような新たな問題が生じる．

1. 発音時刻がより少なく不定期である．
   打楽器音を含む場合と比較すると，発音時刻は全般に少なくなり，ビートの位置に発音時刻が存在しないことが多くなる．また，発音時刻の誤検出も増える傾向にある．
2. 局所的なテンポ変動がより大きい．
   発音時刻の検出結果に前後の揺れが多いため，ビート時刻を局所的に随時補正しないと，予測したビート時刻が正しいものからずれていってしまう．また，一時期正しい解釈だったものがずれて誤ったり，異なる解釈が高い確信度で拮抗したりして，システムの出力が安定しないことが多い．
3. 音楽的知識としてドラムパターンが使えない．
   打楽器音が含まれないために，ビートタイプを判定するための唯一の手がかりであったドラムパターンと照合することができない．さらに，複数楽器音を含む音響信号を音符レベルにシンボル化するのは困難であるため，MIDI 信号を対象とする研究などで使用できる音楽的知識の多くは直接適用できない．

本研究では，これらの問題点をそれぞれ以下のように解決する．具体的な手法については，() 内に示した節で説明する．

□「発音時刻がより少なく不定期である.」

- **発音時刻ベクトルに対する自己相関・相互相関** (**4.1.4**，**4.2.1**(1))
  従来のように各周波数帯域を別々に考慮するのではなく，それぞれの帯域の発音時刻をまとめてベ

[3] ビートの間隔は四分音符の長さに相当するので，ビートの間隔 [sec] = 60 [sec] ÷ 現在のテンポ [M.M.] により求まる．これより，テンポが 61〜120 M.M. の場合には，ビートの間隔 (周期) は 0.500〜0.984 sec となる．

クトル表現することで，全周波数帯域を同時に考慮して処理する．

- **ビートの間隔のヒストリ／ロック機能** (**4.2.1**(1))
  自己相関によって得られるビートの間隔に一時的な誤検出があっても，適切な間隔が得られるようにする．

□「局所的なテンポ変動がより大きい.」

- **発音時刻ベクトルに基づくビート時刻の補正** (**4.2.1**(2))
  相互相関で対処できない局所的な変動に追従する．
- **解釈の統合時の多段階グルーピング** (**4.2.3**)
  システムの出力を安定させるために，解釈を統合する際に同じもの同士を段階的に細かくグルーピングしていき，最も支配的なグループの解釈に基づいて出力を決定する．

□「音楽的知識としてドラムパターンが使えない.」

- **コード変化に基づく音楽的判断** (**4.2.1**(3)(4))
  音楽的知識として，コードがどのようなときに変化しやすいかの傾向を導入する．そして，ビートの解釈とコード変化の度合いとの関係から，ビートタイプを判定し，解釈の確信度を自己評価する．
- **コード変化検出手法** (**4.2.2**)
  周波数スペクトルから音符やコード名などにシンボル化せずに，ビートの解釈の結果を活用して直接コード変化の度合いを求める．たとえコード名がわからない人でもコードの変化はわかる，という現象に注目した手法である．

4 処理モデル

本システムの処理モデルの概念図を図 3 に示し，処理の流れを図 4 に示す．まず，**音響信号の入力**で A/D 変換された音響信号に対して**周波数解析**をおこなう．発音時刻の検出器が各周波数帯域ごとの発音時刻を求め，発音時刻のベクトル化がそれらをまとめる．次に，**ビート予測**の各エージェントが，過去に得られた発音時刻ベクトルからビートの間隔を求め，次のビート時刻の予測とビートタイプの判定をおこなう．この際，コード変化検出器から得た情報を用いて音楽的に判断する．そして，全エージェントの解釈を統合してビート情報 (ビート時刻，ビートタイプ，現在のテンポから成る) を生成する．最後に**ビート情報の出力**が，ネットワークを通じて他のアプリケーションプログラムへとビート情報を送信する．

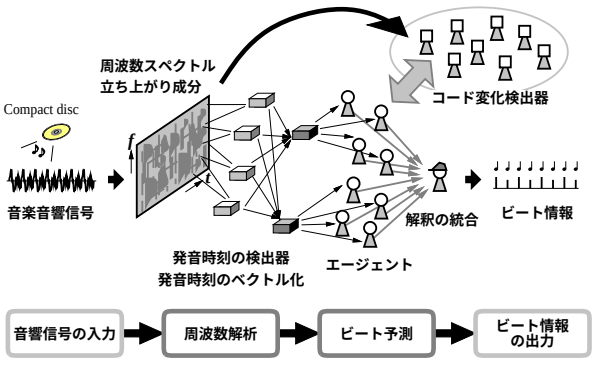


図 3: 処理モデル

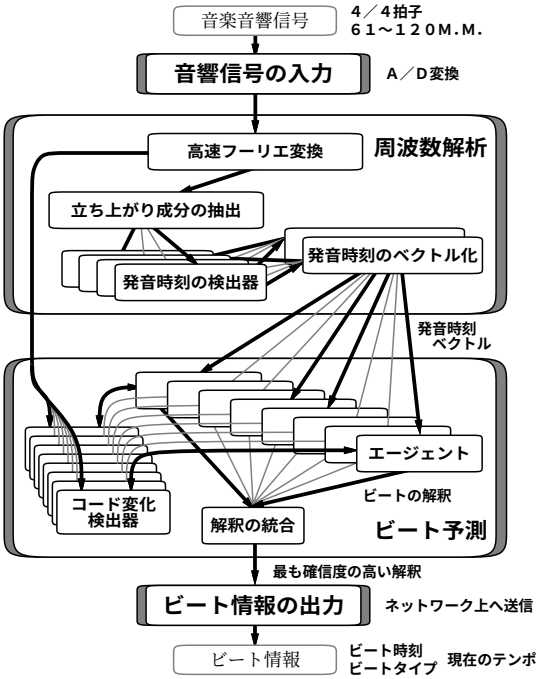


図 4: 処理の流れ

以下では，主要な処理である周波数解析とビート予測について述べる．

4.1 周波数解析

周波数解析により，入力音響信号から周波数スペクトルと n 次元の発音時刻ベクトル (現在の実装では 7 次元) の系列を求める (図 5)．発音時刻ベクトルの各次元は，分割された各周波数帯域での発音時刻に対応する．このようにベクトル表現することで，すべての帯域を同時に考慮できる．また，周波数軸方向に異なった重み付けをしてベクトル化することで，例えば低域に注目した系列や中域に注目した系列などを別々に得ることができる．

4.1.1 高速フーリエ変換 (FFT)

A/D 変換された音響信号に対してハニング窓を用いた FFT をおこない，周波数スペクトル (パワースペクトル) を得る．FFT は，観測区間 (WindowSIZE) を時間軸方向に単位時間 (ShiftSIZE) ずつずらしながら適用する．

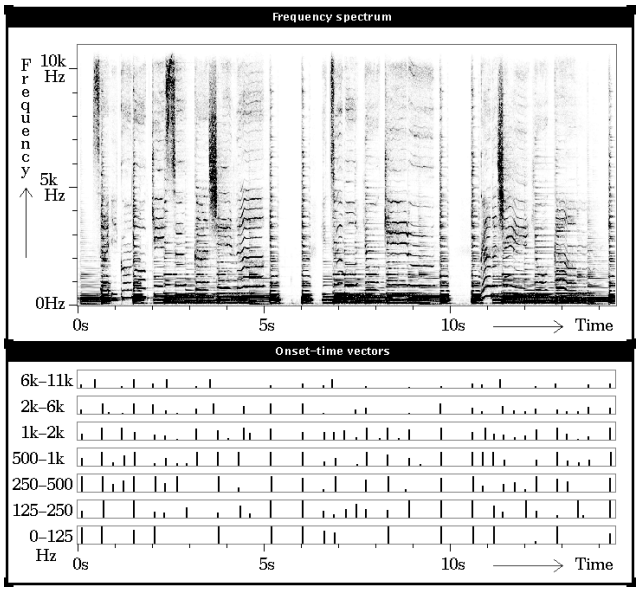


図 5: 周波数スペクトルと発音時刻ベクトル系列の例

現在の実装では，音響信号を 22.05 kHz, 16 bit, モノラルで A/D 変換し，二種類の FFT を計算する．一方は発音時刻を検出するための FFT で，WindowSIZE は 1024 点 (46.44 msec)，ShiftSIZE は 256 点 (11.61 msec) である．したがって周波数分解能は 21.53 Hz，時間分解能 (1 フレーム時間[4]: 1 ft) は 11.61 msec となる．もう一方はコード変化を検出するための FFT で，11.025kHz にダウンサンプリングした信号に対し WindowSIZE は 1024 点 (92.88 msec)，ShiftSIZE は 128 点 (11.61 msec) で計算する．この場合，周波数分解能は 10.77 Hz，時間分解能は 1 ft となる．

4.1.2 立ち上がり成分の抽出

周波数スペクトル中のパワーが確実に増加し続けている周波数成分を，立ち上がり成分として抽出する．そのために周辺の周波数成分の分布状況を考慮して，立ち上がり成分とその立ち上がりの度合 (パワーがどれくらい急激に増加しているか) を求める．具体的な抽出手法は文献 [1, 12] に詳しい．

4.1.3 発音時刻の検出器

複数の発音時刻の検出器 (現在の実装では 7 個) が，それぞれ異なる帯域 (0-125 Hz, 125-250 Hz, 250-500 Hz, 500 Hz-1 kHz, 1-2 kHz, 2-6 kHz, and 6-11 kHz) の発音時刻を求める．

時刻 t，周波数 fにおける立ち上がりの度合を $d(t,f)$ としたとき，まず各時刻における立ち上がり成分の合計値 $D(t) = \sum_f d(t,f)$ を求め，次にその時間変化のピーク時刻とピーク値を，平滑化微分を用いたピーク検出により求める．そして，こうして得られたピーク時刻を発音時刻とする．また，$\sum_f$ において周波数帯域を制限することで，異なる帯域ごとの発音時刻を別々に得られる．

[4] フレーム時間を本システムのすべての処理の時間単位とする．

4.1.4 発音時刻のベクトル化

発音時刻のベクトル化では，すべての発音時刻の検出器の結果をまとめて発音時刻ベクトルの系列へと変換する．現在の実装では，7 個の発音時刻の検出器の結果を，3 種類の発音時刻のベクトル化の処理が周波数軸方向にそれぞれ異なった重み付け (全帯域均等・低域中心・中域中心の 3 種類) をしてベクトル化する．こうして得られた 3 種類の 7 次元発音時刻ベクトルの系列は，ビート予測のエージェントへと送られる．

4.2 ビート予測

複数のエージェントが，発音時刻ベクトルの系列をそれぞれ異なった戦略により解釈し，次のビート時刻を予測する．各エージェントが出力する解釈の結果は，次のビート時刻，そのビートタイプ，ビートの間隔の三つから成る (図 6)．この結果は，解釈の統合へと集められ，最も確信度の高い解釈が選択される．

すべてのエージェントは二つずつの組に分けられる (現在の実装では，エージェントの数は 12 個 6 組である)．同じ組の二つのエージェントは**予測場**を介して協調し合い，両者が同じビートの間隔で，互いにビートの間隔の 1/2 ずれた時刻を予測する (図 7)．予測場とは次のビートが来ることが期待される度合いの分布であり，予測場中の各ピークは次のビート時刻の候補とみなされる．両エージェントは相手の予測場において，自分が解釈したビート時刻に対応する位置の周辺を抑制し合うことで相互作用する (図 7)．これにより，一方が四分音符の裏拍を予測し始めても

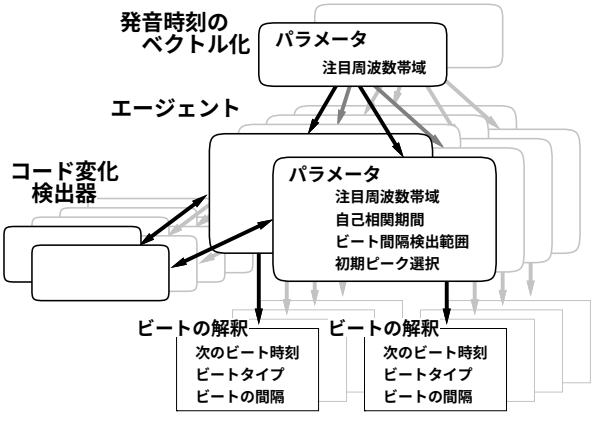


図 6: 発音時刻のベクトル化とエージェント，コード変化検出器の関係

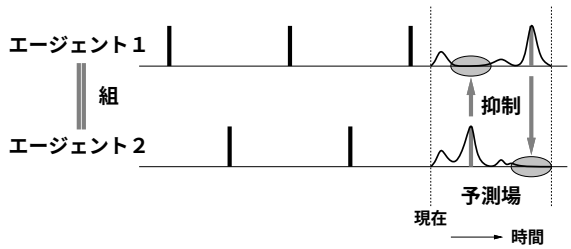


図 7: エージェント間の予測場を介した協調

他方が正しい表拍を予測できる．

エージェントは，ビートを解釈する際の戦略を決定する以下の四つのパラメータを持ち (図 6)，パラメータの値の設定はエージェントごとに異なる (現在の実装でのパラメータの初期設定値を表 1 に示す)．

1. 注目周波数帯域

   エージェントがどの発音時刻のベクトル化の処理から結果を受けとるかを定める．この値は全帯域・低域・中域のいずれかをとり，それぞれベクトル化での重み付けの種類に対応する．これはその帯域に選択的に注意を向けて音楽を聞くことに相当する．

2. 自己相関期間

   後述する発音時刻ベクトル系列の自己相関関数を計算する際の，窓の大きさを定める．この値が大きいほど，より古い発音時刻まで考慮される．

3. ビート間隔検出範囲

   ビートの間隔がとり得る値の範囲を表す．自己相関関数の結果からピークを選ぶ際に，この範囲内のみに限定する．

4. 初期ピーク選択

   primary か secondary をとり，予測場からピークを選ぶ際の方針を定める．このパラメータにより，より多様な解釈が得られやすくなる．

以下では各エージェントの動作 (**4.2.1**)，エージェントが音楽的判断をするための手がかりを提供するコード変化検出器 (**4.2.2**)，全解釈の結果の統合 (**4.2.3**) について順に説明する．

表 1: 戦略を決定するパラメータの初期設定値

組 – エージェント	注目周波数帯域	自己相関期間	ビート間隔検出範囲	初期ピーク選択
1–1	全帯域	500 ft	43-85 ft	primary
1–2	全帯域	500 ft	43-85 ft	secondary
2–1	全帯域	1000 ft	43-85 ft	primary
2–2	全帯域	1000 ft	43-85 ft	secondary
3–1	低域	500 ft	43-85 ft	primary
3–2	低域	500 ft	43-85 ft	secondary
4–1	低域	1000 ft	43-85 ft	primary
4–2	低域	1000 ft	43-85 ft	secondary
5–1	中域	500 ft	43-85 ft	primary
5–2	中域	500 ft	43-85 ft	secondary
6–1	中域	1000 ft	43-85 ft	primary
6–2	中域	1000 ft	43-85 ft	secondary

4.2.1 エージェントの動作

各エージェントは，以下の 4 つの処理によってビートの解釈を生成し，結果をコード変化検出器と解釈の統合へ送る．

(1) ビートの間隔の決定 (周期の決定)

発音時刻ベクトル用に拡張した自己相関関数を計算することで，それぞれの周波数帯域の発音時刻の間隔を同時に考慮しながらビートの間隔を求める．ベクトル化した窓つき正規化自己相関関数 $Ac(\tau)$ を式 (1) のように定義する．

$$Ac(\tau) = \frac{\sum_{t=c-W}^{c} (\vec{o}(t) \cdot \vec{o}(t-\tau))\, w(c-t)}{\sum_{t=c-W}^{c} (\vec{o}(t) \cdot \vec{o}(t))\, w(c-t)} \qquad (1)$$

ここで，$\vec{o}(t)$ は時刻 t における n 次元発音時刻ベクトルであり，c は現在時刻，W は自己相関期間 (戦略パラメータ) とする．窓関数 $w(t)$ は式 (2) で与えられる．

$$w(t) = 1.0 - 0.5\frac{t}{W} \qquad (2)$$

ビートの間隔は，$Ac(\tau)$ においてビート間隔検出範囲 (戦略パラメータ) 内でピーク値をとるτとして得られる．ただし，この結果は一時的に不安定になることがあるため，こうして得られたビートの間隔のヒストリを作る．ヒストリ中に最もよく現れる間隔を用いることで，正しい間隔が安定して得られるようになる．さらに，確信度が高いときには，ビート間隔検出範囲を現在の間隔の近傍に制限 (ロック) することで，現在の適切な間隔を維持して誤検出を防止する．

(2) 次のビート時刻の予測 (位相の決定)

次のビート時刻を予測するために予測場を形成する (図 8)．予測場は，発音時刻ベクトルの全次元の合計値の系列と，(1) で得られたビートの間隔を持つビート時刻の系列との相互相関関数の結果として求める．この予測場が同じ組のエージェントによって抑制された後，エージェントは予測場のピークの中から次のビート時刻となるものを選択する．その際，解釈の確信度が低い場合には，初期ピーク選択 (戦略パラメータ) の値に従ってピークを選ぶ．primary のときには最大ピークを，secondary のときには二番目に大きいピークを次のビート時刻とする．その後は，前回のビート時刻にビートの間隔を加えた時刻にできるだけ近いピークを次のビート時刻とする．

こうして予測されたビート時刻は，テンポの局所的な変動に追従できない場合があるため，ビート時刻

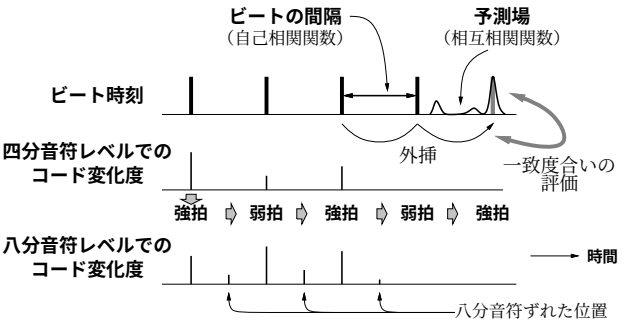


図 8: 次のビート時刻の予測

の近傍の大きな発音時刻ベクトルへと補正する．ただし，予測された時刻の近傍の発音時刻はまだ得ることができない未来の情報のため，既に得られている発音時刻ベクトルをもとに補正量を推定し，次のビート時刻に反映する．

(3) ビートタイプの判定

コード変化検出器から二種類のコード変化の度合い (**四分音符レベルでのコード変化度**，**八分音符レベルでのコード変化度**) を受けとり，音楽的判断をおこなう．前者は現在のビートの解釈における各四分音符 (ビート) の位置でコードがどれくらい変わった可能性があるかを表し，後者は八分音符の位置での可能性を表す．

ビートタイプを判定するために，「コードは他の位置よりも小節の頭で変わりやすい」という音楽的知識を導入する．これは，四分音符レベルでのコード変化度は，強拍 (小節の頭) の位置の方が弱拍の位置よりも大きいということを意味する．そこで，四分音符レベルでのコード変化度が他よりも十分大きい位置を強拍とみなし，強拍と弱拍が交互に来る性質を利用して予測したビート時刻のビートタイプを判定する (図 8)．

(4) 解釈の確信度の自己評価

解釈の確信度を評価するために，従来システムと同様の「ビートの位置で音が鳴ることが多い」という音楽的知識に加え，「コードはビートからずれた位置よりもビートの位置で変わりやすい」という音楽的知識を導入する．これは，八分音符レベルでのコード変化度は，ビートの位置の方が八分音符ずれた位置よりも大きいということを意味する (図 8)．

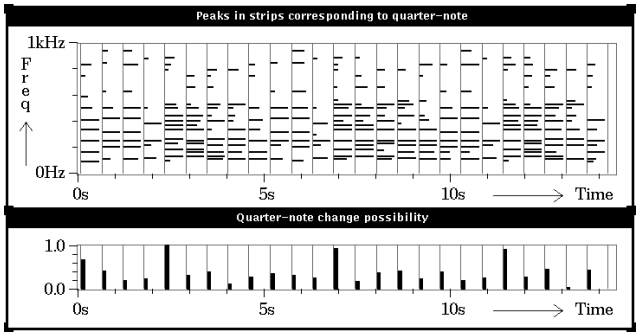
前者の音楽的知識により，発音時刻に基づいて予測されたビート時刻が，過去のビート時刻から外挿された時刻と一致するほど確信度を上げる (図 8)．また後者の音楽的知識により，ビートの位置における八分音符レベルでのコード変化度が，ずれた位置よりも大きいほど確信度を上げる．そして，以上を満たさない場合には確信度を下げる．

4.2.2 コード変化検出器

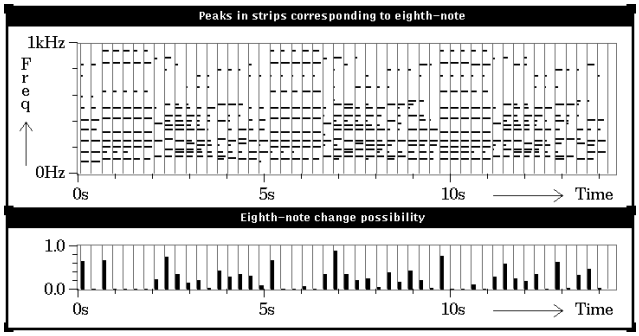
コード変化検出器は，一対一に対応するエージェントから受けとったビートの解釈に基づいて周波数スペクトルを分析し，四分音符レベルと八分音符レベルでのコード変化度を算出する．そして，結果を同じエージェントへ送り返す (図 6)．

まずコード変化度を求める準備として，周波数スペクトルを短冊状に切断し，各短冊内で支配的な音の成分を求める．四分音符レベルでのコード変化度を求める際には，エージェントの解釈によるビート時刻 (四分音符の時刻) で切断し，八分音符レベルでのコード変化度を求める際には，ビート時刻から内挿された八分音符の時刻で切断する (図 9)．次に，こうしてできた各短冊内において，時間軸方向にパワーを合計してヒストグラムを作成し，その周波数軸方向のピークを求める．これらのピークはその短冊内で支配的な音の成分であり，コードやメロディーの周波数成分に相当することが多い．

コード変化度は，隣接する短冊間でピークを比較することで求める．前の短冊に比べてより多くのピークやより大きいピークが生じるほど，その間でコードが変化した可能性が高い．そこで，四分音符レベルでのコード変化度は四分音符の幅の隣接短冊を，八分音符レベルでのコード変化度は八分音符の幅の隣接短冊を，それぞれ比較することで求める．ただし現



(a) 四分音符レベルでのコード変化度



(b) 八分音符レベルでのコード変化度

図 9: 周波数スペクトルが切断された短冊内のピークとコード変化度の例

在の実装では，1 kHz 以下のピークだけを考慮する．

実際に得られたコード変化度の例を図 9 に示す．上段の横線はヒストグラム中のピークを表し，下段の太い縦線はコード変化度を表す．細い縦線は解釈に基づいて短冊状に切断した時刻であり，(a) では一番左から四つごとに小節の先頭，(b) では一番左から二つごとにビート時刻 (四分音符の時刻) となる．

4.2.3 解釈の統合

すべてのエージェントから集められた解釈の結果を，ビートの間隔が同じでビート時刻が近いもの同士グルーピングする．そして各グループにおいてグループ内の全解釈の確信度を合計し，最も合計値の高い支配的なグループを選ぶ．その際，ビート時刻の一時的な変動などにより，本来支配的であるべきグループがばらばらになり，誤ったグループが選択される可能性がある．そこで，同じグループにするためのビート時刻の近さの基準を徐々に狭めながら三段階グルーピングと選択を繰り返すことで，適切な解釈が選択されるようにする．最終的に，こうして得られた最も支配的なグループの解釈に基づいて出力を決定する．

5 実験結果

以上の処理モデルを富士通の分散メモリ型並列計算機 AP1000 上に実装し，実験をおこなった．実験には，市販の CD からサンプリングしたモノラルの音響信号を用いた．ドラムスが含まれない[5]テンポがほぼ一定なロック・ポップス 40 曲の，最初の 1〜2 分間を入力して実験した．これらの曲はテンポが 62〜116 M.M. と広い範囲から選び，28 アーティストによって演奏されたものである．

実験の結果，40 曲中 34 曲に対して正しいビートをリアルタイムに出力できた．本実験では，適切なビート時刻とビートタイプを共に出力した曲を正しいと判定した．正しかったどの曲も最初はビート時刻を認識できてもビートタイプは判定できないが，演奏が始まって十数小節以内にはビートタイプも正しく出力していた．以上から，我々が普段聞くのと同程度の複雑さを持った，様々な楽器音や歌声の含まれた音楽音響信号を扱えることが確認された．

誤認識した曲の多くでは，音数が非常に少なかったり，テンポが局所的に変動したりして，ビート時刻が正しく得られていなかった．そのためコード変化も適切に検出できず，正しい解釈が出力されていなかった．また，ビート時刻は正しいがビートタイプだけを誤る曲もあった．

[5]あるいは，曲中でドラムスが演奏されていない部分．例えば，曲の後半になってはじめてドラムスが鳴り始める曲の前半など．

6 おわりに

本稿では，音響信号に対するビートトラッキングシステムにおいて，打楽器音が含まれない音楽を扱う際の課題と解決法を述べ，実際のシステムの実現方法について処理モデルを中心に述べた．実験の結果，市販の CD による複数の楽器で演奏されたロック・ポップスの音響信号に対し，四分音符と二分音符のレベルでビートをリアルタイムに理解することができることを確認した．

今後は，他のジャンルの音楽への対応やテンポ変化への追従など入力に対する制約を減らしていくと共に，より高次の音楽構造を理解できるようシステムを改良していく予定である．

参考文献

[1] Masataka Goto and Yoichi Muraoka. A beat tracking system for acoustic signals of music. In *Proc. of the Second ACM Intl. Conf. on Multimedia*, pp. 365–372, 1994.

[2] 後藤真孝．計算機は音楽に合わせて手拍子を打てるか? — リアルタイムビートトラッキングシステム —. *bit*, Vol. 28, No. 3, pp. 4–11, 1996.

[3] Roger B. Dannenberg and Bernard Mont-Reynaud. Following an improvisation in real time. In *Proc. of the 1987 ICMC*, pp. 241–248, 1987.

[4] Peter Desain and Henkjan Honing. The quantization of musical time: A connectionist approach. *Computer Music Journal*, Vol. 13, No. 3, pp. 56–66, 1989.

[5] Paul E. Allen and Roger B. Dannenberg. Tracking musical beats in real time. In *Proc. of the 1990 ICMC*, pp. 140–143, 1990.

[6] Anthonie Driesse. Real-time tempo tracking using rules to analyze rhythmic qualities. In *Proc. of the 1991 ICMC*, pp. 578–581, 1991.

[7] David Rosenthal. *Machine Rhythm: Computer Emulation of Human Rhythm Perception*. PhD thesis, Massachusetts Institute of Technology, 1992.

[8] Peter Desain and Henkjan Honing. Advanced issues in beat induction modeling: syncopation, tempo and timing. In *Proc. of the 1994 ICMC*, pp. 92–94, 1994.

[9] Edward W. Large. Beat tracking with a nonlinear oscillator. In *Working Notes of the IJCAI-95 Workshop on Artificial Intelligence and Music*, pp. 24–31, 1995.

[10] W. Andrew Schloss. *On The Automatic Transcription of Percussive Music – From Acoustic Signal to High-Level Analysis*. PhD thesis, CCRMA, Stanford University, 1985.

[11] H. Katayose, H. Kato, M. Imai, and S. Inokuchi. An approach to an artificial music expert. In *Proc. of the 1989 ICMC*, pp. 139–146, 1989.

[12] Masataka Goto and Yoichi Muraoka. Music understanding at the beat level – real-time beat tracking for audio signals –. In *Working Notes of the IJCAI-95 Workshop on Computational Auditory Scene Analysis*, pp. 68–75, 1995.

[13] Masataka Goto and Yoichi Muraoka. A real-time beat tracking system for audio signals. In *Proc. of the 1995 ICMC*, pp. 171–174, 1995.

[14] David Rosenthal, Masataka Goto, and Yoichi Muraoka. Rhythm tracking using multiple hypotheses. In *Proc. of the 1994 ICMC*, pp. 85–87, 1994.