(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号
特許第6500869号
(P6500869)
(45)発行日　平成31年4月17日(2019.4.17)　(24)登録日　平成31年3月29日(2019.3.29)

(51)Int.Cl.　F I
G10H 1/38 (2006.01)　G10H 1/38 Z

請求項の数 9　(全 36 頁)

(21)出願番号　特願2016-190423(P2016-190423)
(22)出願日　平成28年9月28日(2016.9.28)
(65)公開番号　特開2018-54854(P2018-54854A)
(43)公開日　平成30年4月5日(2018.4.5)
審査請求日　平成29年10月12日(2017.10.12)

(73)特許権者　000001443
カシオ計算機株式会社
東京都渋谷区本町１丁目６番２号
(74)代理人　100074099
弁理士　大菅　義之
(72)発明者　南高　純一
東京都羽村市栄町３丁目２番１号　カシオ計算機株式会社羽村技術センター内

審査官　上田　雄

最終頁に続く

(54)【発明の名称】コード解析装置、方法、及びプログラム

(57)【特許請求の範囲】
【請求項１】
楽曲を第１長で区切った第１区間の構成音に基づいて第１調を推定し、前記第１区間と少なくとも部分的に重なる区間であって、前記楽曲を前記第１長と異なる長さの第２長で区切った第２区間の構成音に基づいて第２調を推定する調推定処理と、
前記推定された前記第１調及び前記第２調を比較することにより最適な調を決定する調決定処理と、
を実行するコード解析装置。
【請求項２】
前記最適な調に基づいて前記楽曲の前記第１区間のコードを判定するコード判定処理を実行し、更に前記コード判定処理は、前記楽曲の小節を区分した拍毎に、当該拍の構成音を判定し、当該構成音に基づいて当該拍のコードを判定する、請求項１に記載のコード解析装置。
【請求項３】
前記第１区間、前記第２区間、又は前記拍毎の構成音の判定は、当該第１区間、当該第２区間、又は当該拍の期間内でノートオンしている前記楽曲の楽音毎に、当該楽音のベロシティと当該期間内での発音時間長とに基づいて決定されるパワー情報値を当該楽音のピッチに対応するピッチクラスに累算することにより、当該第１区間、当該第２区間、又は当該拍における前記ピッチクラス毎のパワー情報累算値を算出する処理である、請求項２に記載のコード解析装置。

【請求項４】
前記第１区間、前記第２区間、又は拍毎に、前記第１調、前記第２調、又はコードの候補に対応して、前記ピッチクラスの各々が前記第１調、前記第２調の候補の音階音又はコードの候補の構成音と一致する場合に当該ピッチクラスに対して算出されている前記パワー情報累算値を第１のパワー評価値に累算し、一致しない場合に当該ピッチクラスに対して算出されている前記パワー情報累算値を第２のパワー評価値に累算し、前記第１調、前記第２調、又はコードの候補毎に算出される前記第１のパワー評価値及び前記第２のパワー評価値を比較することにより、当該第１区間、当該第２区間、又は拍における前記第１調、前記第２調、又は前記コードを判定する、請求項３に記載のコード解析装置。
【請求項５】
前記第１区間の区間長は１小節の長さであり、前記第２区間の区間長は１小節の倍数であり、前記調決定処理は、前記第１区間と前記第２区間とで重なる小節毎に、当該小節毎に判定された前記第１調及び前記第２調を比較することにより、当該小節に対応する前記最適な調を決定する、請求項１に記載のコード解析装置。
【請求項６】
前記調決定処理は、区間開始位置を１小節ずつずらしながら前記第１区間の区間長又は前記第２区間の区間長で楽曲を区切って前記第１区間又は前記第２区間を決定する、請求項５に記載のコード解析装置。
【請求項７】
前記判定されたコードを表示する表示処理を更に実行する、請求項２乃至６の何れかに記載のコード解析装置。
【請求項８】
コード解析装置の処理部が、
楽曲を第１長で区切った第１区間の構成音に基づいて第１調を推定し、前記第１区間と少なくとも部分的に重なる区間であって、前記楽曲を前記第１長と異なる長さの第２長で区切った第２区間の構成音に基づいて第２調を推定し、
前記推定された前記第１調及び前記第２調を比較することにより最適な調を決定する、
処理を実行するコード解析方法。
【請求項９】
コード解析を行うコンピュータに、
楽曲を第１長で区切った第１区間の構成音に基づいて第１調を推定し、前記第１区間と少なくとも部分的に重なる区間であって、前記楽曲を前記第１長と異なる長さの第２長で区切った第２区間の構成音に基づいて第２調を推定するステップと、
前記推定された前記第１調及び前記第２調を比較することにより最適な調を決定するステップと、
を実行させるためのプログラム。
【発明の詳細な説明】
【技術分野】
【０００１】
本発明は、入力する楽曲のコードを解析するコード解析装置、方法、及びプログラムに関する。
【背景技術】
【０００２】
楽曲からコードを抽出したいという要請がある。例えば、スタンダードＭＩＤＩ（Ｍｕｓｉｃａｌ　Ｉｎｓｔｒｕｍｅｎｔ　Ｄｉｇｉｔａｌ　Ｉｎｔｅｒｆａｃｅ）ファイルは、一般的にメロディパートがあって伴奏パートがある。そのような楽曲を、例えば電子鍵盤楽器で弾こうとする場合、メロディは右手で比較的容易に弾けるが、左手で伴奏も楽しみたいというような場合がある。しかし、スタンダードＭＩＤＩファイルにおいては、左手用の適当な楽曲パートのデータがあればよいが、ほとんどの場合そのようなデータは含まれていない。しかし、せっかく電子鍵盤楽器があるのだからやはり両手で弾きたいとい

うような場合がある。このような場合に、楽曲のスタンダードＭＩＤＩファイルからコードを判定して提示できれば、演奏者はそのコードに応じた左手演奏等を行えて便利である。

【０００３】

従来、楽曲のコードを判定するいくつかの技術が知られている（例えば、特許文献１～４に記載の技術）。

【先行技術文献】

【特許文献】

【０００４】

【特許文献１】特開２０００－２５９１５４号公報

【特許文献２】特開２００７－２８６６３７号公報

【特許文献３】特開２０１５－４０９６４号公報

【特許文献４】特開２０１５－７９１９６号公報

【発明の概要】

【発明が解決しようとする課題】

【０００５】

しかし、上述の従来技術はいずれも、コードの和音の構成音以外の音についての配慮が十分でないため判定の精度が落ちる場合があるという課題があった。

【０００６】

また、コードの和音を判定するのに十分な発音がなく適切な判定ができない場合があるという課題があった。

【０００７】

更に、調性、特に転調についての配慮がないために、適切なコード判定ができない場合があるという課題があった。

【０００８】

そこで、本発明は、転調も適切に判定できる調判定の結果からより適切なコード判定が行えるようにすることを目的とする。

【課題を解決するための手段】

【０００９】

態様の一例では、楽曲を第１長で区切った第１区間の構成音に基づいて第１調を推定し、第１区間と少なくとも部分的に重なる区間であって、楽曲を第１長と異なる長さの第２長で区切った第２区間の構成音に基づいて第２調を推定する調推定処理と、推定された第１調及び第２調を比較することにより最適な調を決定する調決定処理と、を実行する。

【発明の効果】

【００１０】

本発明によれば、転調も適切に判定できる調判定の結果からより適切なコード判定を行うことが可能となる。

【図面の簡単な説明】

【００１１】

【図１】コード解析装置の一実施形態のハードウェア構成例を示す図である。

【図２】ＭＩＤＩシーケンスデータの構成例と調判定の結果得られる調データを示す図である。

【図３】コード判定の結果得られるコード進行データの構成例を示す図である。

【図４】コード解析装置の全体処理の例を示すメインフローチャートである。

【図５】コード判定処理の詳細例を示すフローチャートである。

【図６】調判定処理の詳細例を示すフローチャートである。

【図７】小節と拍及び調判定の説明図である。

【図８】調判定の動作結果例を示す図である。

【図９】調判定処理におけるキー判定処理の詳細例を示すフローチャートである。

【図１０】スケールノートの説明図である。

【図１１】ピッチクラスパワー作成処理の例を示すフローチャートである。
【図１２】ピッチクラスパワー作成処理の説明図である。
【図１３】調判定処理における結果保存処理の詳細例を示すフローチャートである。
【図１４】コード判定処理におけるマッチング＆結果保存処理の詳細例を示すフローチャートである。
【図１５】コードトーンの説明図である。
【図１６】最小コスト計算処理と経路確定処理の説明図である。
【図１７】最小コスト計算処理の詳細例を示すフローチャートである。
【図１８】コスト計算処理の詳細例を示すフローチャートである。
【図１９】経路確定処理の詳細例を示すフローチャートである。
【発明を実施するための形態】
【００１２】
以下、本発明を実施するための形態について図面を参照しながら詳細に説明する。図１は、コード解析装置１００の一実施形態を、ソフトウェア処理として実現できるコンピュータのハードウェア構成の一例を示す図である。
【００１３】
図１に示されるコンピュータは、ＣＰＵ１０１、ＲＯＭ（Ｒｅａｄ Ｏｎｌｙ Ｍｅｍｏｒｙ：読出し専用メモリ）１０２、ＲＡＭ（Ｒａｎｄｏｍ Ａｃｃｅｓｓ Ｍｅｍｏｒｙ：ランダムアクセスメモリ）１０３、入力手段１０４、表示手段１０５、サウンドシステム１０６、及び通信インタフェース１０７を有し、これらがバス１０８によって相互に接続された構成を有する。図１に示される構成はコード解析装置を実現できるコンピュータの一例であり、そのようなコンピュータはこの構成に限定されるものではない。
【００１４】
ＣＰＵ１０１は、当該コンピュータ全体の制御を行う。ＲＯＭ１０２は、後述する図４、図５、図８～図１０、図１３、図１４のフローチャートで示されるコード解析処理プログラムや、複数楽曲分のスタンダードＭＩＤＩファイル等を記憶する。ＲＡＭ１０３は、コード解析処理プログラムの実行時に作業用メモリとして使用される。ＣＰＵ１０１は、ＲＯＭ１０２からコード解析処理プログラムをＲＡＭ１０３に読み出して実行する。コード解析処理プログラムは、例えば特には図示しない可搬記録媒体に記録して配布してもよく、或いは通信インタフェース１０７によりインターネットやローカルエリアネットワーク等のネットワークから取得できるようにしてもよい。
【００１５】
入力手段１０４は、ユーザによるキーボードやマウス等による入力操作を検出し、その検出結果をＣＰＵ１０１に通知する。入力操作は、例えば楽曲の選択操作、コード解析の指示操作、楽曲の再生装置等である。また、ユーザによる入力手段１０４の操作により、楽曲のスタンダードＭＩＤＩファイルがネットワークから通信インタフェース１０７を介してＲＡＭ１０３にダウンロードされるようにしてもよい。
【００１６】
表示手段１０５は、ＣＰＵ１０１の制御によって出力されるコード判定データを液晶ディスプレイ装置等に表示する。
【００１７】
サウンドシステム１０６は、ユーザが入力手段１０４により、ＲＯＭ１０２やネットワークから取得した楽曲のスタンダードＭＩＤＩファイルの再生を指示したときに、当該スタンダードＭＩＤＩファイルのシーケンスを順次読み込んで解釈することにより、ユーザが指定した楽器音で楽音信号を生成し、スピーカ等から発音する。
【００１８】
図２（ａ）は、ＲＯＭ１０２からＲＡＭ１０３に読み込まれ、又はネットワークから通信インタフェース１０７を介してＲＡＭ１０３にダウンロードされるスタンダードＭＩＤＩファイルに格納されている、ＭＩＤＩシーケンスデータの構成例を示す図である。楽曲は複数パート（＝トラック）で構成され、各パートの先頭のノートイベントへのポインタ

情報が、ｍｉｄｉｅｖ［０］、ｍｉｄｉｅｖ［１］、ｍｉｄｉｅｖ［２］、・・・として保持される。ＣＰＵ１０１は、ポインタ情報ｍｉｄｉｅｖ［ｉ］（ｉ＝０，１，２，・・・）を参照することにより、ｉパートのＲＡＭ１０３に記憶されている最初のノートイベントにアクセスすることができる。

【００１９】

このノートイベントは、次の構造体データを保持する。ｌＴｉｍｅは発音開始時刻を保持する。ｌＧａｔｅはゲートタイム（発音時間長）を保持する。これらの時刻の単位としては、例えばティック（ｔｉｃｋ）が使われる。この場合、４分音符は例えば４８０ティックの時間長を有し、４分の４拍子の楽曲の場合、１拍＝４８０ティックとなる。ｂｙＤａｔａ［０］はステータスを保持する。ｂｙＤａｔａ［１］は発音されるノートのピッチを保持する。ｂｙＤａｔａ［２］は発音されるノートのベロシティを保持する。ｂｙＤａｔａ［３］はその他ノートの発音を制御するために必要な情報を保持する。ｎｅｘｔは次のノートイベントへのポインタ、ｐｒｅｖは１つ前のノートイベントへのポインタである。ＣＰＵ１０１は、ｎｅｘｔやｐｒｅｖを参照することにより、ＲＡＭ１０３に記憶されている次の又は１つ前のノートイベントにアクセスすることができる。

【００２０】

また、ＣＰＵ１０１がサウンドシステム１０６を制御して楽曲を再生するために必要な、テンポや拍子などのメタ情報は、ポインタ情報ｍｅｔａｅｖ［０］、ｍｅｔａｅｖ［１］、ｍｅｔａｅｖ［２］、・・・から参照することができる。

【００２１】

図２（ｂ）は、後述する調判定処理の結果得られる調データの構成例を示す図である。調情報は、ポインタ情報ｔｏｎａｌｉｔｙ［０］、ｔｏｎａｌｉｔｙ［１］、ｔｏｎａｌｉｔｙ［２］、・・・からアクセスされる。ｔｏｎａｌｉｔｙ［ｉ］（ｉ＝０、１、２、・・・）は、小節番号ｉに対応する調情報へのポインタである。これらのポインタから参照される調情報は、次の構造体データを保持する。ｌＴｉｃｋは楽曲のメロディに対する調の開始の時刻を保持する。ｌＴｉｃｋの時間単位は、前述したティックである。ｉＭｅａｓＮｏは調が開始される小節の番号を保持する。ｉＫｅｙは調のキーを保持する。ｉＳｃａｌｅは調のタイプを保持するが、本実施形態では未使用である。ｄｏＰｏｗｅｒＶａｌｕｅは調判定時のパワー評価値を保持する。ｉＬｅｎｇｔｈは調判定時の区間長を保持し、後述するように小節を単位とする区間長を示す１、２、又は４である。

【００２２】

図３は、後述するコード判定処理の結果得られるコード進行データの構成例を示す図である。コード進行データは、楽曲を構成する各小節の拍毎に、例えば第１候補、第２候補、第３候補、・・・の複数候補を持つことができる。いま、楽曲の先頭からの拍番号の通番を第１要素番号ｌＣｎｔ（ｌＣｎｔ＝０、１、２、・・・）、各拍における候補番号を第２要素番号ｉ（ｉ＝０、１、２、・・・）とすれば、各コード進行データは、ポインタ情報ｃｈｏｒｄＰｒｏｇ［ｌＣｎｔ］［ｉ］によってアクセスすることができる。このポインタ情報からアクセスされるコード情報は、次の構造体データを保持する。ｌＴｉｃｋはメロディに対するコードの開始の時刻を保持する。ｌＴｉｃｋの時間単位は、前述したティックである。ｉＭｅａｓＮｏは調の小節の番号を保持する。ｉＴｉｃｋＩｎＭｅａｓは小節内でのコードの開始の時刻を保持する。ｉＴｉｃｋＩｎＭｅａｓの時間単位も、前述したティックである。本実施形態では、コードは拍毎に判定されるため、ｉＴｉｃｋＩｎＭｅａｓの時間単位も拍単位となり、１拍目、２拍目、３拍目、又は４拍目の何れかとなる。図２（ａ）の説明で前述したように、１拍は一般的に４８０ティックであるため、ｉＴｉｃｋＩｎＭｅａｓは、０、４８０、９６０、１４４０の何れかの値となる。ｉＲｏｏｔはコード判定結果（ルート）を保持する。ｉＴｙｐｅはコード判定結果（タイプ）を保持する。ｄｏＰｏｗｅｒＶａｌｕｅｎはコード判定時のパワー評価値を保持する。

【００２３】

図４は、図１のＣＰＵ１０１が実行するコード解析装置の全体処理の例を示すメインフローチャートである。例えば、図１のコード解析装置１００がスマートフォン等の汎用の

コンピュータであった場合、ユーザがコード解析装置１００のアプリをタップすることにより、ＣＰＵ１０１が、図４のフローチャートで例示されるコード解析処理プログラムをスタートする。ＣＰＵ１０１はまず、レジスタやＲＡＭ１０３に記憶される変数の初期化等の初期化処理を実行する（ステップＳ４０１）。その後、ＣＰＵ１０１は、ステップＳ４０２からＳ４０８までの一連の処理を、繰り返し実行する。

【００２４】

ＣＰＵ１０１はまず、ユーザがアプリ上の特定のボタンをタップすることによりアプリの終了が指示されたか否かを判定する（ステップＳ４０２）。ステップＳ４０２の判定がＹＥＳになると、ＣＰＵ１０１は、図４のフローチャートで例示されるコード解析処理を終了する。

【００２５】

ステップＳ４０２の判定がＮＯならば、ＣＰＵ１０１は、ユーザが入力手段１０４を介して楽曲の選曲を指示したか否かを判定する（ステップＳ４０３）。

【００２６】

ステップＳ４０３の判定がＹＥＳならば、ＣＰＵ１０１は、ＲＯＭ１０２又は通信インタフェース１０７を介してネットワークから、図２（ａ）のデータフォーマットを有する楽曲のスタンダードＭＩＤＩファイルのＭＩＤＩシーケンスデータを、ＲＡＭ１０３に読み込む（ステップＳ４０４）。

【００２７】

その後、ＣＰＵ１０１は、後述するコード判定処理を実行することにより、読込みが指示された楽曲のＭＩＤＩシーケンスデータの全体に渡って、コードを判定する処理を実行する（ステップＳ４０５）。その後、ＣＰＵ１０１は、ステップＳ４０１の処理に戻る。

【００２８】

ステップＳ４０３の判定がＮＯならば、ＣＰＵ１０１は、ユーザが入力手段１０４を介して楽曲の再生を指示しているか否かを判定する（ステップＳ４０６）。

【００２９】

ステップＳ４０６の判定がＹＥＳならば、ＣＰＵ１０１は、ＲＡＭ１０３に読み込まれているＭＩＤＩシーケンスデータを解釈しながら、サウンドシステム１０６に発音指示を出力することにより、楽曲の再生を行わせる（ステップＳ４０７）。その後、ＣＰＵ１０１は、ステップＳ４０１の処理に戻る。

【００３０】

ステップＳ４０６の判定がＮＯならば、ＣＰＵ１０１は、ステップＳ４０１の処理に戻る。

【００３１】

図５は、図４のステップＳ４０５のコード判定処理の詳細例を示すフローチャートである。

【００３２】

始めに、ＣＰＵ１０１は、調判定処理を実行することにより、楽曲の小節毎に調を判定する（ステップＳ５０１）。この結果、ＲＡＭ１０３に、図２（ｂ）に例示されるデータ構成を有する調データが得られる。

【００３３】

次に、ＣＰＵ１０１は、全ての小節毎に、以下のステップＳ５０３からＳ５０５までの一連の処理を繰り返し実行する（ステップＳ５０２）。

【００３４】

この全ての小節毎の繰返し処理において、ＣＰＵ１０１は、更に小節内の全ての拍毎に、以下のステップＳ５０４とＳ５０５の処理を繰り返し実行する。この拍毎の繰返しにおいてＣＰＵ１０１はまず、ピッチクラスパワー作成処理を実行する（ステップＳ５０４）。ここでは、ＣＰＵ１０１は、拍の構成音をピッチクラスパワーとして判定する。この処理の詳細については図１０及び図１１の説明で後述する。

【００３５】

次に、ＣＰＵ１０１は、マッチング＆結果保存処理を実行する（ステップＳ５０５）。ここでは、ＣＰＵ１０１は、ステップＳ５０４で算出した現在の拍におけるピッチクラス毎のパワー情報累算値に基づいてその拍の構成音を判定し、その構成音に基づいてその拍のコードを判定する。この処理の詳細については図１４の説明で後述する。その後、ＣＰＵ１０１は、ステップＳ５０３の処理に戻る。

【００３６】

小節内の全ての拍について、ステップＳ５０４とＳ５０５の処理の実行が終了し、その小節内の全ての拍に対応するコード進行データが生成されると、ＣＰＵ１０１はステップＳ５０２の処理に戻る。

【００３７】

楽曲の全ての小節について、ステップＳ５０２からＳ５０５の一連の処理の実行が終了し、楽曲の全ての小節内の全ての拍に対応するコード進行データが生成されると、ＣＰＵ１０１はステップＳ５０６の処理に移行する。

【００３８】

ステップＳ５０６において、ＣＰＵ１０１は、楽曲の全ての小節及び小節内の全ての拍に対して図３に例示されるデータフォーマットで得られる複数候補からなるコード進行データの全ての組合せの中から、楽曲全体でコストが最小となるコードの組合せを算出する。この処理の詳細については、図１６から図１８の説明において後述する。

【００３９】

この結果、ＣＰＵ１０１は、楽曲全体に渡るコード進行の経路を確定し、これにより最適コードが確定する（ステップＳ５０７）。この処理の詳細については、図１６及び図１９の説明において後述する。この最適なコード進行は、特には図示しないが、ユーザによる入力手段１０４からの指示に基づいて、表示手段１０５に表示される。ユーザの指示に応じて、図４のステップＳ４０７の再生処理によるサウンドシステム１０６からの楽曲再生に同期して、表示手段１０５に表示される最適なコード進行が進んでゆく。その後、ＣＰＵ１０１は、図５のフローチャートで示される図４のステップＳ４０５のコード判定処理を終了する。

【００４０】

次に、図５のステップＳ５０１の調判定処理の詳細について、以下に説明する。図６は、図５のステップＳ５０１の調判定処理の詳細例を示すフローチャートである。また、図７（ａ）は小節と拍の説明図、図７（ｂ）は調判定の説明図である。

【００４１】

いま、読み込まれている楽曲が４拍子の楽曲の場合、図７の（ａ－２）に示されるような楽曲（Ｓｏｎｇ）の進行に従って、図７の（ａ－３）に示されるように小節番号ｉＭｅａｓＮｏが、０、１、２、・・・というように進んでゆく。そして、図７（ａ－１）に示されるように、各小節内で、拍番号ｉＢｅａｔＮが０、１、２、３というように繰り返される。

【００４２】

図６のフローチャートで例示される調判定処理において、ＣＰＵ１０１は、図７（ｂ－１）の楽曲（Ｓｏｎｇ）及び（ｂ－２）の小節番号（ｉＭｅａｓＮｏ）の進行に合わせて、図７（ｂ－３）（ｂ－４）（ｂ－５）に示されるように、１小節長、２小節長、４小節長という１小節の倍数の単位を有する複数の区間長から区間長を選択しながら、次の処理を実行する。以下の説明で、１小節の区間長をｉＦｒａｍｅＴｙｐｅ＝０、２小節の区間長をｉＦｒａｍｅＴｙｐｅ＝１、４小節の区間長をｉＦｒａｍｅＴｙｐｅ＝２と記載する。なお、区間長の選択は、１、２、４小節に限られたものではなく、例えば２、４、８小節であってもよい。ＣＰＵ１０１は、ｉＦｒａｍｅＴｙｐｅ＝０、１、２の各区間長で楽曲を区切った区間（図７（ｂ－３）（ｂ－４）（ｂ－５）の各直線矢印が示す区間）毎に（ステップＳ６０１）、区間の開始小節を１小節ずつずらしながら（ステップＳ６０２）、以下の処理を実行する。

【００４３】

ＣＰＵ１０１は、ｉＦｒａｍｅＴｙｐｅにより規定される各区間毎に、その区間の構成音を判定し、その構成音に基づいて調を判定するキー判定処理を実行する（ステップＳ６０３）（調判定手段として動作）。この処理の詳細については、図９から図１２の説明において後述する。

【００４４】

図８は、調判定の動作結果例を示す図である。この結果例において、図８（ａ）は小節番号（ｉＭｅａｓＮｏ）を示している。また、図８（ｂ）において、図８（ａ）の各小節番号に対応して記載されている音名群は、各小節番号に対応する小節毎にＭＩＤＩシーケンスデータからのノートイベントにより実際に発音が行われる楽音の各構成音の音名を示している。

【００４５】

図８の結果例において、例えばｉＦｒａｍｅＴｙｐｅ＝０（１小節区間長）に対しては、図８（ｃ）に示されるように、図８（ａ）の各小節番号（ｉＭｅａｓＮｏ）の１小節ずつを単位として、判定区間が１小節ずつずらされながら、Ｂ♭、Ｂ♭、Ｇ、Ｂ♭、Ａ♭、Ｅ♭というように調が判定される。このとき、例えば「Ｂ♭：３」という記載は、Ｂ♭の調判定時にパワー評価値＝３という評価値が得られていることを示している。この評価値については後述するが、この値が大きいほど調判定の信頼性が高いことを示している。

【００４６】

次に、例えばｉＦｒａｍｅＴｙｐｅ＝１（２小節区間長）に対しては、図８（ｄ）の上下上下・・・の順で、図８（ａ）の各小節番号（ｉＭｅａｓＮｏ）の連続する２小節ずつを単位として、判定区間が１小節ずつずらされながら、Ｂ♭、Ｃ、Ｃ、Ｂ♭、Ａ♭、Ｅ♭というように調が判定される。

【００４７】

更に、例えばｉＦｒａｍｅＴｙｐｅ＝２（４小節区間長）に対しては、図８（ｅ）の左上から右下に向かって、図８（ａ）の各小節番号（ｉＭｅａｓＮｏ）の連続する４小節ずつを単位として、判定区間が１小節ずつずらされながら、Ｂ♭、Ｃ、Ｃ、Ａ♭、Ａ♭、Ａ♭というように調が判定される。

【００４８】

図６のステップＳ６０３によるキー判定処理の動作の後、ＣＰＵ１０１は、ステップＳ６０１によりｉＦｒａｍｅＴｙｐｅ＝０（１小節）、１（２小節）、２（４小節）というように順次指定される区間長で繰り返されるステップＳ６０３のキー判定処理の結果、現在までに計算されている区間長間で重なる区間毎に、その区間に対して各区間長で判定された調同士を比較することにより現時点での最適な調を決定する、結果保存処理を実行する（ステップＳ６０４）（調決定手段として動作）。この処理の詳細については、図１３の説明において後述する。

【００４９】

図８の例においては、例えば図８（ａ）の小節番号（ｉＭｅａｓＮｏ）＝０において、ｉＦｒａｍｅＴｙｐｅ＝１までのキー判定処理（ステップＳ６０３）が終了した時点で、図８（ｃ）のｉＦｒａｍｅＴｙｐｅ＝０の調判定はキー音名がＢ♭でパワー評価値が３、図８（ｄ）のｉＦｒａｍｅＴｙｐｅ＝１の調判定はキー音名がＢ♭でパワー評価値が４となっている。従って、このキー判定処理に続く結果保存処理（ステップＳ６０４）において、パワー評価値が大きいほうの調として、その時点での最適な調はキー音名がＢ♭でパワー評価値は４と決定される。更に、ｉＦｒａｍｅＴｙｐｅ＝２までのキー判定処理が終了した時点で、その時点での最適な調は上述のようにキー音名がＢ♭でパワー評価値は４、図８（ｅ）のｉＦｒａｍｅＴｙｐｅ＝２の調判定はキー音名がＣでパワー評価値は７となっている。従って、このキー判定処理に続く結果保存処理において、最終時点での最適な調はキー音名＝Ｃと決定される。この結果、ＣＰＵ１０１は、ＲＡＭ１０３に、図２（ｂ）のデータフォーマットの調情報を生成する。この調情報において、ｌＴｉｃｋにはｉＭｅａｓＮｏ＝０の小節の先頭の開始時刻が格納される。また、ｉＭｅａｓＮｏには小節番号として０が格納される。ｉＫｅｙには最適な調として決定されたキー音名Ｃに対応す

るキー値＝０が格納される。ｄｏＰｏｗｅｒＶａｌｕｅには最適な調が決定されたときのパワー評価値７が格納される。そして、ｉＬｅｎｇｔｈには最適な調が決定されたときに採用されたｉＦｒａｍｅＴｙｐｅの値２が格納される。

【００５０】

図８の例において、例えば図８（ａ）の小節番号（ｉＭｅａｓＮｏ）＝１～３までは、上述の場合と同様にして、各小節毎に最適な調がキー音名＝Ｃと決定されパワー評価値７が得られ、図２（ｂ）に例示されるデータフォーマットで調データが生成される。続いて、小節番号（ｉＭｅａｓＮｏ）＝４では、ｉＦｒａｍｅＴｙｐｅ＝１（２小節長）から最も高いパワー評価値＝６を有するとしてキー音名＝Ｂｂの調が選択される。以下、小節番号（ｉＭｅａｓＮｏ）＝５、６では、やはりｉＦｒａｍｅＴｙｐｅ＝１から最も高いパワー評価値＝７を有するとしてキー音名＝Ｅ♭の調が選択される。これは、小節番号（ｉＭｅａｓＮｏ）が３から４に変わるときに、転調が発生したことを示している。

【００５１】

このように、本実施形態では、複数の区間長（ｉＦｒａｍｅＴｙｐｅ）での調判定結果が総合的に判断されることにより、例えば転調が発生した場合には、パワー評価値に基づいて１小節区間長や２小節区間長の短い区間長の判定結果が採用されることにより、転調を検知することが可能となる。また、１小節だけではコードを判定するのに十分な発音がないような場合であっても、パワー評価値に基づいて２小節区間長や４小節区間長のより長い区間長の判定結果が採用されることにより、適切な判定を行うことが可能となる。更に、本実施形態では、後述するようにパワー評価値を算出するときに、調の音階音以外の音についての配慮も行われるため、判定の精度を維持することが可能となる。

【００５２】

図６のステップＳ６０４の処理の後、ＣＰＵ１０１は、ステップＳ６０２の処理に戻る。ＣＰＵ１０１は、ひとつのｉＦｒａｍｅＴｙｐｅの値に対して区間の開始小節を１小節ずつずらしながら楽曲の全ての小節について、ステップＳ６０３のキー判定処理及びステップＳ６０４の結果保存処理を繰り返し実行する。全ての小節に対する上記繰返し処理が終了すると、ステップＳ６０１の処理に戻る。そして、ＣＰＵ１０１は、ｉＦｒａｍｅＴｙｐｅ＝０、１、２の全ての値（小節区間長）に対してステップＳ６０２からＳ６０４までの一連の処理を繰り返し実行する。３種類全てのｉＦｒａｍｅＴｙｐｅの値に対する上記繰返し処理が終了すると、図６のフローチャートで例示される図５のステップＳ５０１の調判定処理を終了する。

【００５３】

図９は、図６の調判定処理におけるステップＳ６０３のキー判定処理の詳細例を示すフローチャートである。ＣＰＵ１０１はまず、ピッチクラスパワー作成処理を実行する（ステップＳ９０１）。ここでは、ＣＰＵ１０１は、現在設定されている１小節、２小節、又は４小節の区間長を有する区間内でノートオンする楽曲のノートイベント毎に、そのノートイベントのベロシティとその区間内での発音時間長とに基づいて決定されるパワー情報値をそのノートのピッチに対応するピッチクラスに累算することにより、その区間におけるピッチクラス毎のパワー情報累算値を算出する。ここで、ピッチクラスとは、１オクターブを１２半音で１２分割したときの各半音に対して与えられている整数値をいい、例えば１オクターブ内の音名Ｃは整数値０、Ｃ＃又はＤ♭は１、Ｄは２、Ｄ＃又はＥ♭は３、Ｅは４、Ｆは５、Ｆ＃又はＧ♭は６、Ｇは７、Ｇ＃又はＡ♭は８、Ａは９、Ａ＃又はＢ♭は１０、Ｂは１１に、それぞれ対応している。本実施形態では、調が、１小節、２小節、又は４小節の区間長を有する区間毎に判定される。ここで、調を表すキー音名及び音階音は、オクターブに依存しない音名の組合せとして決定される。従って、本実施形態では、ＣＰＵ１０１が、区間内で発音されるノートを、ＲＡＭ１０３に記憶されている図２（ａ）のデータフォーマットを有する各ノートイベントの発音時刻ｌＴｉｍｅ及びゲートタイム（発音時間）ｌＧａｔｅから検索し、検索されたノートのピッチ（図２（ａ）のｂｙＤａｔａ［１］）を、その値を１２で除算したときの０から１１の何れかの剰余値として、ピッチクラスに変換する。そして、ＣＰＵ１０１は、そのノートのベロシティとその区間

内での発音時間長とに基づいて決定されるパワー情報値をそのノートのピッチに対応するピッチクラスに累算することにより、その拍におけるピッチクラス毎のパワー情報累算値を算出する。いま、ピッチクラスをｉＰｃ（０≦ｉＰｃ≦１１）として、ステップＳ９０１のピッチクラスパワー作成処理により作成された各ピッチクラスｉＰｃ（０≦ｉＰｃ≦１１）におけるパワー換算値を、ピッチクラスパワーｌＰｉｃｈＣｌａｓｓＰｏｗｅｒ［ｉＰｃ］とする。この処理の詳細については図１０及び図１１の説明で後述する。

【００５４】

次に、ＣＰＵ１０１は、調のキー値を示す全てのｉｋｅｙの値０から１１について、以下のステップＳ９０３からＳ９１０の一連の処理を実行する（ステップＳ９０２）。まず、ＣＰＵ１０１は、ステップＳ９０３からＳ９０８の一連の処理を実行する。

【００５５】

具体的には、ＣＰＵ１０１は始めに、共にＲＡＭ１０３に記憶される変数である第１のパワー評価値ｌＰｏｗｅｒと第２のパワー評価値ｌＯｔｈｅｒＰｏｗｅｒの各値を０にクリアする（ステップＳ９０３）。

【００５６】

次に、ＣＰＵ１０１は、０から１１までの値を有する全てのピッチクラスｉＰｃ毎に、以下のステップＳ９０５からＳ９０７の処理を実行する（ステップＳ９０４）。

【００５７】

まず、ＣＰＵ１０１は、ステップ９０４で指定された現在のピッチクラスｉＰｃが、ステップＳ９０２で指定された現在のキー値ｉｋｅｙに基づいて定まる調の音階音に含まれるか否かを判定する（ステップＳ９０５）。この判定は、「ｓｃａｌｅｎｏｔｅ［（１２＋ｉＰｃ－ｉｋｅｙ）％１２］の値が１であるか否か」を判定する演算である。図１０は、スケールノートの説明図である。図１０において、（ａ）ｍａｊｏｒ、（ｂ）ｈｍｉｎｏｒ、及び（ｃ）ｍｍｉｎｏｒの各行は、調のキー値がピッチクラス＝０（音名＝Ｃ）である場合における、メジャースケール、ハーモニックマイナースケール、及びメロディックマイナースケールの各スケールの音階を構成するピッチクラス及び音名を示している。各行で、値「１」が記載されているピッチクラス及び音名が、その行に対応するスケールの音階を構成する構成音である。値「０」が記載されているピッチクラス及び音名は、その行に対応するスケールの音階を構成しない音である。本実施形態では、処理の簡単化及び安定性確保のために、図１０の（ａ）、（ｂ）、及び（ｃ）の各スケールの音階音が比較対象とされるのではなく、これらのスケールを統合した図１０の（ｄ）のスケール（以下これを「統合スケールｓｃａｌｅ」と記載する）の音階音が比較対象とされる。図１０（ｄ）の統合スケールｓｃａｌｅの音階音又は音階を構成しない音は、図１０の（ａ）、（ｂ）、及び（ｃ）の各スケールの音階音又は音階を構成しない音に対してピッチクラス（音名）毎に論理和を演算して得られるものである。すなわち、ピッチクラス（音名）毎に、図１０の（ａ）、（ｂ）、及び（ｃ）の各スケールの値が「１」であれば統合スケールｓｃａｌｅの値は「１」であり、図１０の（ａ）、（ｂ）、及び（ｃ）の全てのスケールの値が「０」であれば統合スケールｓｃａｌｅの値は「０」である。図１のＲＯＭ１０２は、キー値がピッチクラス＝０（音名＝Ｃ）であるときの図１０（ｄ）の統合スケールｓｃａｌｅに対応する配列定数ｓｃａｌｅ［ｉ］を記憶している。ここで、ｉは図１０の０から１１までのピッチクラスの値をとり、配列要素値ｓｃａｌｅ［ｉ］には、図１０（ｄ）の統合スケールｓｃａｌｅの行のピッチクラスｉにおける値１又は０が格納されている。ＣＰＵ１０１は、ステップＳ９０５において、まず、「（１２＋ｉＰｃ－ｉｋｅｙ）％１２」の値を演算する。この演算では、ステップＳ９０４で指定されているピッチクラスｉＰｃとステップＳ９０２で指定されているキー値ｉｋｅｙとの差分値が、どのピッチクラスになるかが算出されている。括弧内で１２が加算されているのは「ｉＰｃ－ｉｋｅｙ」の値がマイナス値にならないようにするためである。また、「％」は、剰余を求める剰余演算を示している。ＣＰＵ１０１は、この演算結果を配列要素引数として、ＲＯＭ１０２から読み出した配列要素値ｓｃａｌｅｎｏｔｅ［（１２＋ｉＰｃ－ｉｋｅｙ）％１２］の値が１であるか否かを判定する。これにより、ＣＰＵ１０１は、ステップＳ９０４で

指定されているピッチクラスｉＰｃが、図１０（ｄ）に示されるキー値がピッチクラス＝０（音名＝Ｃ）のときの統合スケールｓｃａｌｅをキー値がステップＳ９０２で指定されているキー値ｉｋｅｙであるときの統合スケールｓｃａｌｅに変換したときの音階音に含まれているか否かを判定することができる。

【００５８】

ステップ９０４で指定された現在のピッチクラスｉＰｃが、ステップＳ９０２で指定された現在のキー値ｉｋｅｙに対応する統合スケールｓｃａｌｅの音階音に含まれている場合（ステップＳ９０５の判定がＹＥＳの場合）には、ＣＰＵ１０１は、そのピッチクラスｉＰｃに対応してステップＳ９０１で算出されているピッチクラスパワーｌＰｉｃｈＣｌａｓｓＰｏｗｅｒ［ｉＰｃ］を、第１のパワー評価値ｌＰｏｗｅｒに累算する（ステップＳ９０６）。図６のステップＳ９０６において、演算記号「＋＝」は、その左辺の値にその右辺の値を累算する演算を示している。後述するステップＳ９０７、図１４のステップＳ１４０６、及びステップＳ１４０７の各「＋＝」記号も同様である。

【００５９】

一方、ステップ９０４で指定された現在のピッチクラスｉＰｃが、ステップＳ９０２で指定された現在のキー値ｉｋｅｙに対応する統合スケールｓｃａｌｅの音階音に含まれていない場合（ステップＳ９０５の判定がＮＯの場合）には、ＣＰＵ１０１は、そのピッチクラスｉＰｃに対応してステップＳ９０１で算出されているピッチクラスパワーｌＰｉｃｈＣｌａｓｓＰｏｗｅｒ［ｉＰｃ］を、第２のパワー評価値ｌＯｔｈｅｒＰｏｗｅｒに累算する（ステップＳ９０７）。

【００６０】

ＣＰＵ１０１は、０から１１までの値を有する全てのピッチクラスｉＰｃについて、上述のステップＳ９０５からＳ９０７の処理の実行を終えると（ステップＳ９０４の判定が「終了」になると）、第１のパワー評価値ｌＰｏｗｅｒを第２のパワー評価値ｌＯｔｈｅｒＰｏｗｅｒで除算して得られる値として、ステップＳ９０２で現在指定されているキー値ｉｋｅｙに対応するパワー評価値ｄｏＫｅｙＰｏｗｅｒを算出する（ステップＳ９０８）。ステップＳ９０８の実行時点で、第１のパワー評価値ｌＰｏｗｅｒはステップＳ９０２で現在指定されているキー値ｉｋｅｙに対応する統合スケールｓｃａｌｅの音階音がどれくらいの強さで発音されるかを示している。また、第２のパワー評価値ｌＯｔｈｅｒＰｏｗｅｒは、キー値ｉｋｅｙに対応する統合スケールｓｃａｌｅの音階音以外の音がどれくらいの強さで鳴っているかを示している。従って、「ｌＰｏｗｅｒ÷ｌＯｔｈｅｒＰｏｗｅｒ」として算出されるパワー評価値ｄｏＫｅｙＰｏｗｅｒは、現在の区間で鳴っている楽音（ノート）がどのくらい、現在のキー値ｉｋｅｙに対応する統合スケールｓｃａｌｅの音階音らしいかを示す指標となる。

【００６１】

続いて、ＣＰＵ１０１は、ステップＳ９０８で算出した現在のキー値ｉｋｅｙに対応するパワー評価値ｄｏＫｅｙＰｏｗｅｒを、現時点の直前までに指定されたキー値に対応するパワー評価値最高値ｄｏＭａｘと比較する（ステップＳ９０９）。そして、ＣＰＵ１０１は、ｄｏＫｅｙＰｏｗｅｒがｄｏＭａｘｅｒ以上であれば、パワー評価値最高値ｄｏＭａｘとパワー評価値最高キー値ｉｍａｘｋｅｙを、現在のパワー評価値ｄｏＫｅｙＰｏｗｅｒとキー値ｉＫｅｙにそれぞれ置き換える（ステップＳ９１０）。その後、ＣＰＵ１０１は、ステップＳ９０２の処理に戻って、次のキー値ｉｋｅｙに対する処理に移行する。

【００６２】

図１１は、図９のステップＳ９０１のピッチクラスパワー作成処理の例を示すフローチャートであり、図１１は、ピッチクラスパワー作成処理の説明図である。ＣＰＵ１０１はまず、図４のステップＳ４０４でＲＡＭ１０３に読み込んだ図２（ａ）のデータフォーマット例を有するＭＩＤＩシーケンスデータにおいて、全てのトラック毎に、以下のステップＳ１１０２からＳ１１１１までの一連の処理を繰り返し実行する（ステップＳ１１０１）。以下、ＣＰＵ１０１は、ステップＳ１１０１において、各トラックを、ＲＡＭ１０３に記憶される変数であるトラック番号ｉＴｒａｃｋの値として順次指定する。ここでは、

ＣＰＵ１０１は、図２（ａ）の例のＭＩＤＩシーケンスデータにおいて、トラック番号ｉＴｒａｃｋに対応するポインタ情報ｍｉｄｉｅｖ［ｉＴｒａｃｋ］を参照することにより、トラック番号ｉＴｒａｃｋに対応するパートのＲＡＭ１０３に記憶されている最初のノートイベントにアクセスする。

【００６３】

そして、ＣＰＵ１０１は、上記最初のノートイベントから順次、各ノートイベント内の図２（ａ）のｎｅｘｔポインタを参照しながら辿ることにより、トラック番号ｉＴｒａｃｋのパート内の全てのノートイベント毎に、以下のステップＳ１１０３からＳ１１１１までの一連の処理を繰り返し実行する（ステップＳ１１０２）。ここで、現在のノートイベントへのポインタを「ｍｅ」と記載する。そして、現在のノートイベント内のデータ、例えば図２（ａ）の発音開始時刻ｌＴｉｍｅへの参照は、「ｍｅ－＞ｌＴｉｍｅ」などと記載する。

【００６４】

ＣＰＵ１０１は、図６のステップＳ６０１により定まる１小節長、２小節長、又は４小節長の区間長を有してステップＳ６０２で指定される開始小節から始まる区間（以下これを「現在の該当範囲」と記載する）に、ステップＳ１１０２で指定されている現在のノートイベントが含まれるか否かを判定する（ステップＳ１１０３）。いま、ＣＰＵ１０１は、楽曲先頭から計時した現在の該当範囲の先頭時刻を計算して、その時刻を、変数である該当範囲開始時刻ｌＴｉｃｋＦｒｏｍとしてＲＡＭ１０３に記憶する。前述したように、拍及び小節の時間単位としては、何れも前述したティックが使用され、１拍は一般的に４８０ティックで、４分の４拍子の楽曲の場合、１小節は４拍である。従って、例えば４分の４拍子の楽曲の場合では、楽曲の先頭を０小節目として図６のステップＳ６０２で指定される区間の開始小節番号をカウントした場合、区間の開始小節の開始時刻は（４８０ティック×４拍×区間の開始小節番号）となり、これが該当範囲開始時刻ｌＴｉｃｋＦｒｏｍとして算出される。同様に、ＣＰＵ１０１は、楽曲先頭から計時した現在の該当範囲の終了時刻を計算して、その時刻を、変数である該当範囲終了時刻ｌＴｉｃｋＴｏとしてＲＡＭ１０３に記憶する。該当範囲終了時刻ｌＴｉｃｋＴｏは、該当範囲開始時刻ｌＴｉｃｋＦｒｏｍ＋（４８０ティック×４拍×ステップＳ６０１で指定されている区間長）である。そして、ＣＰＵ１０１は、現在のノートイベントのポインタｍｅから参照される現在のノートイベントの発音開始時刻ｌＴｉｍｅと発音時間長ｌＧａｔｅ（共に図２（ａ）参照）とから定まる現在のノートイベントの発音区間が、上記該当範囲開始時刻ｌＴｉｃｋＦｒｏｍ及び該当範囲終了時刻ｌＴｉｃｋＴｏに対して、図１２の１２０１、１２０２、又は１２０３の何れかの関係にあるか否かを判定する。これらの何れかの関係が成立すれば、現在指定されているノートイベントによる発音は、現在の該当範囲に含まれる（かかっている）ことになる。これが成立する場合、ＣＰＵ１０１は、ステップＳ１１０３の判定をＹＥＳとする。具体的には、図１２の関係より、ＣＰＵ１０１は、該当範囲終了時刻ｌＴｉｃｋＴｏが現在のノートイベントの発音開始時刻ｍｅ－＞ｌＴｉｍｅより後であって、かつ該当範囲開始時刻ｌＴｉｃｋＦｒｏｍが現在のノートイベントの発音終了時刻である（発音開始時刻ｍｅ－＞ｌＴｉｍｅ＋発音時間長ｍｅ－＞ｌＧａｔｅ）より前であれば、ステップ Ｓ１１０３の判定をＹＥＳとする。

【００６５】

ステップＳ１１０３の判定がＮＯであれば、ＣＰＵ１０１は、現在のノートイベントは現在の該当範囲に含まれていないと判定して、ステップＳ１１０２の処理に戻り、次のノートイベントに対する処理に移行する。

【００６６】

ステップＳ１１０３の判定がＹＥＳであれば、ＣＰＵ１０１は、該当範囲開始時刻ｌＴｉｃｋＦｒｏｍが、現在のノートイベントの発音開始時刻ｍｅ－＞ｌＴｉｍｅよりも後であるか否かを判定する（ステップＳ１１０４）。

【００６７】

ステップＳ１１０４の判定がＹＥＳならば、図１２の１２０１の状態が判定されたこと

になるので、ＣＰＵ１０１は、ＲＡＭ１０３に記憶される変数である現在のノートイベントの現在の該当範囲内での発音開始時刻ｌＴｉｃｋＳｔａｒｔに、該当範囲開始時刻ｌＴｉｃｋＦｒｏｍをセットする（ステップＳ１１０５）。

【００６８】

一方、ステップＳ１１０４の判定がＮＯならば、図１２の１２０２又は１２０３の状態が判定されたことになるので、ＣＰＵ１０１は、現在のノートイベントの現在の該当範囲内での発音開始時刻ｌＴｉｃｋＳｔａｒｔに、現在のノートイベントの発音開始時刻ｍｅ－＞ｌＴｉｍｅをセットする（ステップＳ１１０６）。

【００６９】

ステップＳ１１０５又はＳ１１０６の処理の後、ＣＰＵ１０１は、該当範囲終了時刻ｌＴｉｃｋＴｏが、現在のノートイベントの発音終了時刻である（発音開始時刻ｍｅ－＞ｌＴｉｍｅ＋発音時間長ｍｅ－＞ｌＧａｔｅ）よりも後であるか否かを判定する（ステップＳ１１０７）。

【００７０】

ステップＳ１１０７の判定がＹＥＳならば、図１２の１２０１又は１２０２の状態が判定されたことになるので、ＣＰＵ１０１は、ＲＡＭ１０３に記憶される変数である現在のノートイベントの現在の該当範囲内での発音終了時刻ｌＴｉｃｋＥｎｄに、現在のノートイベントの発音終了時刻である（発音開始時刻ｍｅ－＞ｌＴｉｍｅ＋発音時間長ｍｅ－＞ｌＧａｔｅ）をセットする（ステップＳ１１０８）。

【００７１】

一方、ステップＳ１１０７の判定がＮＯならば、図１２の１２０３の状態が判定されたことになるので、ＣＰＵ１０１は、現在のノートイベントの現在の該当範囲内での発音終了時刻ｌＴｉｃｋＥｎｄに、該当範囲終了時刻ｌＴｉｃｋＴｏをセットする（ステップＳ１１０９）。

【００７２】

ステップＳ１１０８又はＳ１１０９の処理の後、ＣＰＵ１０１は、ＲＡＭ１０３に記憶される変数である現在のノートイベントのピッチｉＰｉｔｃｈに、現在のノートイベントのポインタｍｅから参照されるピッチｂｙＤａｔａ［１］（図２（ａ）参照）の値を格納する（ステップＳ１１１０）。

【００７３】

そして、ＣＰＵ１０１は、現在のノートイベントのピッチｉＰｉｔｃｈを１２で除算したときの剰余（ｉＰｉｔｃｈ％１２）として算出される現在のノートイベントに対応するピッチクラスにおけるＲＡＭ１０３に記憶される配列要素値であるピッチクラスパワーｌＰｉｃｈＣｌａｓｓＰｏｗｅｒ［ｉＰｉｔｃｈ％１２］に、以下の計算値を格納する。ＣＰＵ１０１は、現在のノートイベントのベロシティとパート情報とから定まるベロシティ情報ｌＰｏｗｅｒＷｅｉｇｈｔに、現在のノートイベントの現在の該当範囲における発音時間長（ｌＴｉｃｋＥｎｄ－ｌＴｉｃｋＳｔａｒｔ）を乗算した値として、上記ピッチクラスパワーｌＰｉｃｈＣｌａｓｓＰｏｗｅｒ［ｉＰｉｔｃｈ％１２］を算出する。ここで、ベロシティ情報ｌＰｏｗｅｒＷｅｉｇｈｔは例えば、現在のノートイベントのポインタｍｅから参照されるベロシティｍｅ－＞ｂｙＤａｔａ［２］（図２（ａ）参照）に、現在のトラック番号ｉＴｒａｃｋ（ステップＳ１１０１参照）に対応するパートに対して予め規定されＲＯＭ１０２に記憶されている所定のパート係数を乗算した値として算出される。このようにして、現在のノートイベントに対応するピッチクラスパワーｌＰｉｃｈＣｌａｓｓＰｏｗｅｒ［ｉＰｉｔｃｈ％１２］の値は、現在の該当範囲内での現在のノートイベントの発音時間が長ければ長いほど、また発音の強さを示すベロシティが大きければ大きいほど、そして現在のノートイベントが属するパートに応じて、現在のノートイベントに対応するピッチクラス（ｉＰｉｔｃｈ％１２）の音の現在の該当範囲での構成割合が大きいことになる。

【００７４】

ステップＳ１１１１の処理の後、ＣＰＵ１０１は、ステップＳ１１０２の処理に戻り、

次のノートイベントに対する処理に移行する。

【００７５】

上述のステップＳ１１０３からＳ１１１１までの一連の処理が繰り返し実行されることにより、現在のトラック番号ｉＴｒａｃｋに対応する全てのノートイベントｍｅに対応する処理が終了すると、ＣＰＵ１０１は、ステップＳ１１０１の処理に戻り、次のトラック番号ｉＴｒａｃｋに対する処理に移行する。更に、上述のステップＳ１１０２からＳ１１１１の処理が繰り返し実行されることにより、全てのトラック番号ｉＴｒａｃｋに対応する処理が終了すると、ＣＰＵ１０１は、図１１のフローチャートで例示される図９のステップＳ９０１のピッチクラスパワー作成処理を終了する。

【００７６】

図１３は、図５の調判定処理の詳細例である図６のフローチャートの処理におけるステップＳ６０４の結果保存処理の詳細例を示すフローチャートである。ここでは、ＣＰＵ１０１は、図６のステップＳ６０３のキー判定処理により、現在の該当範囲（ステップＳ６０１で定まる区間長を有してステップＳ６０２で指定される開始小節から始まる区間）に対して算出されたパワー評価値ｄｏＫｅｙＰｏｗｅｒを、他の区間長に対して得られている重なる区間のパワー評価値と比較することにより、その区間に対して現時点での最適な調を決定する。

【００７７】

ＣＰＵ１０１はまず、楽曲を構成する全ての小節毎に、ステップＳ１３０２からＳ１３０４までの一連の処理を繰り返し実行する（ステップＳ１３０１）。以下、ＣＰＵ１０１は、ステップＳ１３０１において、楽曲の先頭の小節の小節番号を０としてそこから順次カウントされるＲＡＭ１０３に記憶される変数である小節番号ｉの値として、各小節を順次指定する。

【００７８】

この繰返し処理で、ＣＰＵ１０１はまず、図６のステップＳ６０２で指定される区間の開始小節番号から、それを含んで図６のステップＳ６０１で指定されている区間長分の現在の該当範囲の小節番号のグループに、小節番号ｉが含まれるか否かを判定する（ステップＳ１３０１）。

【００７９】

ステップＳ１３０２の判定がＮＯならば、ＣＰＵ１０１は、ステップＳ１３０１の処理に戻り、次の小節番号に対する処理に移行する。

【００８０】

ステップＳ１３０２の判定がＹＥＳならば、ＣＰＵ１０１は、図６のステップＳ６０３のキー判定処理（図９のフローチャートの処理）により、現在の該当範囲に対して算出されたパワー評価値ｄｏＫｅｙＰｏｗｅｒが、ＲＡＭ１０３に記憶されている図２（ｂ）のデータフォーマット例を有する調データにおいて、小節番号ｉに対応するポインタ情報ｔｏｎａｌｉｔｙ［ｉ］から参照される調情報として記憶されているパワー評価値ｔｏｎａｌｉｔｙ［ｉ］．ｄｏＰｏｗｅｒ以上であるか否かを判定する（ステップＳ１３０３）。

【００８１】

ステップＳ１３０３の判定がＮＯならば、ＣＰＵ１０１は、ステップＳ１３０１の処理に戻り、次の小節番号に対する処理に移行する。

【００８２】

ステップＳ１３０３の判定がＹＥＳならば、ＣＰＵ１０１は、小節番号ｉに対応するポインタ情報ｔｏｎａｌｉｔｙ［ｉ］から参照される調情報において、調のキーｔｏｎａｌｉｔｙ［ｉ］．ｉＫｅｙに、図９のステップＳ９１０で算出されているパワー評価値最高キー値ｉｍａｘｋｅｙをセットする。また、ＣＰＵ１０１は、調判定時のパワー評価値ｔｏｎａｌｉｔｙ［ｉ］．ｄｏＰｏｗｅｒＶａｌｕｅに、図９のステップＳ９１０で算出されているパワー評価値最高値ｄｏＭａｘをセットする。更に、ＣＰＵ１０１は、調判定時の区間長ｔｏｎａｌｉｔｙ［ｉ］．ｉＬｅｎｇｔｈに、図６のステップＳ６０１で指定されている図６のステップＳ６０１で指定されている現在の区間長をセットする（以上、ス

テップＳ１３０４）。ステップＳ１３０４の処理の後、ＣＰＵ１０１は、ステップＳ１３０１の処理に戻り、次の小節番号に対する処理に移行する。

【００８３】

なお、ＲＡＭ１０３に生成される図２（ｂ）の調データは、図４のステップＳ４０４の曲データの読込み時に、読み込まれたＭＩＤＩシーケンスデータのノートイベントの存在範囲の値から、必要小節数分のポインタ情報及びそれらから参照される調情報が初期生成される。例えば４分の４拍子の楽曲であれば、前述したように１拍＝４８０ティックとして、必要小節数Ｎ＝（末尾のノートイベントの図２（ａ）の（ｌＴｉｍｅ＋ｌＧａｔｅ）値÷４８０÷４拍）が算出される。この結果、ｔｏｎａｌｉｔｙ［０］からｔｏｎａｌｉｔｙ［Ｎ－１］までのポインタ情報と、そこから参照される図２（ｂ）の調情報の構造体データが生成される。そして、各ポインタ情報ｔｏｎａｌｉｔｙ［ｉ］（０≦ｉ≦Ｎ－１）から参照される構造体データにおいて、ｔｏｎａｌｉｔｙ［ｉ］．ｉＫｅｙには無効値が初期設定される。ｔｏｎａｌｉｔｙ［ｉ］．ｄｏＰｏｗｅｒＶａｌｕｅには、例えばマイナス値が初期設定される。ｔｏｎａｌｉｔｙ［ｉ］．ｌＴｉｃｋには、（４８０ティック×４拍×ｉ小節）のティック時刻値がセットされる。また、ｔｏｎａｌｉｔｙ［ｉ］．ｉＭｅａｓＮｏには、小節番号ｉがセットされる。なお、本実施形態では、ｔｏｎａｌｉｔｙ［ｉ］．ｉＳｃａｌｅは未使用である。

【００８４】

前述した図８の例において、図６のステップＳ６０１で指定されている区間長＝１小節長（ｉＦｒａｍｅＴｙｐｅ＝０）、ステップＳ６０２で指定されている区間の開始小節番号（図８（ａ）のｉＭｅａｓＮｏ）＝０のときに、ステップＳ６０３でのキー判定処理の結果として、図８（ｃ）に示されるように、パワー評価値最高キー値ｉｍａｘｋｅｙとしてピッチクラス＝１０（音名＝Ｂ♭）が得られ、パワー評価値最高値ｄｏＭａｘとして３が得られている。この結果、図６のステップＳ６０４の結果保存処理における図１３のフローチャートにおいて、小節番号ｉ＝０のときに、ステップＳ１３０２の判定がＹＥＳとなる。そして、ステップＳ１３０３の判定処理が実行されるが、このとき、ｔｏｎａｌｉｔｙ［０］．ｄｏＰｏｗｅｒＶａｌｕｅの値はマイナスの初期値となっているため、パワー評価値最高値ｄｏＭａｘ＝３のほうが大きくなり、ステップＳ１３０３の判定はＹＥＳとなる。この結果、ステップＳ１３０４において、ｔｏｎａｌｉｔｙ［０］．ｉＫｅｙ＝ｉｍａｘｋｅｙ＝１０（音名Ｂ♭）、ｔｏｎａｌｉｔｙ［０］．ｄｏＰｏｗｅｒＶａｌｕｅ＝ｄｏＭａｘ＝３、ｔｏｎａｌｉｔｙ［０］．ｉＬｅｎｇｔｈ＝１（小節長）がセットされる。

【００８５】

次に、前述した図８の例において、図６のステップＳ６０１で指定されている区間長＝２小節長（ｉＦｒａｍｅＴｙｐｅ＝１）、ステップＳ６０２で指定されている区間の開始小節番号（図８（ａ）のｉＭｅａｓＮｏ）＝０のときに、ステップＳ６０３でのキー判定処理の結果として、図８（ｄ）に示されるように、パワー評価値最高キー値ｉｍａｘｋｅｙとしてピッチクラス＝１０（音名＝Ｂ♭）が得られ、パワー評価値最高値ｄｏＭａｘとして４が得られている。この結果、図６のステップＳ６０４の結果保存処理における図１３のフローチャートにおいて、小節番号ｉ＝０のときに、ステップＳ１３０２の判定がＹＥＳとなる。そして、ステップＳ１３０３の判定処理が実行されるが、このとき、ｔｏｎａｌｉｔｙ［０］．ｄｏＰｏｗｅｒＶａｌｕｅ＝３となっているため、パワー評価値最高値ｄｏＭａｘ＝４のほうが大きくなり、ステップＳ１３０３の判定はＹＥＳとなる。この結果、ステップＳ１３０４において、ｔｏｎａｌｉｔｙ［０］．ｉＫｅｙ＝ｉｍａｘｋｅｙ＝１０（同じ音名Ｂ♭）、ｔｏｎａｌｉｔｙ［０］．ｄｏＰｏｗｅｒＶａｌｕｅ＝ｄｏＭａｘ＝４、ｔｏｎａｌｉｔｙ［０］．ｉＬｅｎｇｔｈ＝２（小節長）がセットされる。

【００８６】

更に、前述した図８の例において、図６のステップＳ６０１で指定されている区間長＝４小節長（ｉＦｒａｍｅＴｙｐｅ＝２）、ステップＳ６０２で指定されている区間の開始小節番号（図８（ａ）のｉＭｅａｓＮｏ）＝０のときに、ステップＳ６０３でのキー判定

処理の結果として、図８（ｅ）に示されるように、パワー評価値最高キー値ｉｍａｘｋｅｙとしてピッチクラス＝０（音名＝Ｃ）が得られ、パワー評価値最高値ｄｏＭａｘとして７が得られている。この結果、図６のステップＳ６０４の結果保存処理における図１３のフローチャートにおいて、小節番号ｉ＝０のときに、ステップＳ１３０２の判定がＹＥＳとなる。そして、ステップＳ１３０３の判定処理が実行されるが、このとき、ｔｏｎａｌｉｔｙ［０］．ｄｏＰｏｗｅｒＶａｌｕｅ＝４となっているため、パワー評価値最高値ｄｏＭａｘ＝７のほうが大きくなり、ステップＳ１３０３の判定はＹＥＳとなる。この結果、ステップＳ１３０４において、ｔｏｎａｌｉｔｙ［０］．ｉＫｅｙ＝ｉｍａｘｋｅｙ＝０（音名Ｃ）、ｔｏｎａｌｉｔｙ［０］．ｄｏＰｏｗｅｒＶａｌｕｅ＝ｄｏＭａｘ＝７、ｔｏｎａｌｉｔｙ［０］．ｉＬｅｎｇｔｈ＝４（小節長）がセットされる。

【００８７】

以上のステップＳ１３０２からＳ１３０４の一連の処理の実行が、楽曲を構成する全ての小節番号ｉについて完了すると、ＣＰＵ１０１は、図１３のフローチャートで示される図６のステップＳ６０４の結果保存処理を終了する。

【００８８】

以上の例からわかるように、本実施形態では、複数の区間長（ｉＦｒａｍｅＴｙｐｅ）での調判定結果が総合的に判断されることにより、例えば転調が発生した場合、又は１小節だけではコードを判定するのに十分な発音がないような場合であっても、適切な調判定を行うことが可能となる。更に、本実施形態では、後述するようにパワー評価値を算出するときに、コードの和音の構成音以外の音についての配慮も行われるため、調判定の精度を維持することが可能となる。更に、本実施形態では、図９のステップＳ９０６とＳ９０７で、調の音階音に関わるの第１のパワー評価値ｌＰｏｗｅｒと音階音以外の音に関わる第２のパワー評価値ｌＯｔｈｅｒＰｏｗｅｒが算出され、それらに基づいて調のキー値ｉｋｅｙに対応するパワー評価値ｄｏＫｅｙＰｏｗｅｒが算出される。従って、調のキー値ｉｋｅｙに関して、その音階音と音階音以外の音の両方に配慮したパワー評価を行うことができ、判定の精度を維持することが可能となる。

【００８９】

次に、上記詳述した図５のステップＳ５０１の調判定処理により、図２（ｂ）の調データとして各小節毎に調が適切に判定された後、全ての小節毎（ステップＳ５０２）及び各小説内の全ての拍毎（ステップＳ５０３）に繰り返し実行されるステップＳ５０４のピッチクラスパワー作成処理と、ステップＳ５０５のマッチング＆結果保存処理の詳細について、以下に説明する。

【００９０】

まず、図５のステップＳ５０４のピッチクラスパワー作成処理の詳細について説明する。ここでは、ＣＰＵ１０１は、現在設定されている拍内でノートオンする楽曲のノートイベント毎に、そのノートイベントのベロシティとその拍内での発音時間長とに基づいて決定されるパワー情報値をそのノートのピッチに対応するピッチクラスに累算することにより、現在の拍におけるピッチクラス毎のパワー情報累算値を算出する。

【００９１】

図５のステップＳ５０４の詳細は、前述した図１１のフローチャートで示される。前述した図９のステップＳ９０１の詳細処理である図１１の説明では、「現在の該当範囲」は現在指定されている調判定のための小節区間であった、これに対して図５のステップＳ５０４の詳細処理である以下の図１１の説明では、「現在の該当範囲」は図５のステップＳ５０２で指定される小節内のステップＳ５０３で指定される拍に対応する範囲である。更に、図１２の該当範囲開始時刻ｌＴｉｃｋＦｒｏｍは、現在の拍の開始時刻である。前述したように、拍及び小節の時間単位としては、何れも前述したティックが使用され、１拍は一般的に４８０ティックで、４分の４拍子の楽曲の場合、１小節は４拍である。従って、例えば４分の４拍子の楽曲の場合では、楽曲の先頭を０小節目として図５のステップＳ５０２で指定される小節の小節番号をカウントした場合、その小節の開始時刻は（４８０ティック×４拍×小節番号）となり、更に、小節の先頭の拍を０として図５のステップＳ

５０３で指定される拍の拍番号をカウントした場合、その拍の小節内での開始時刻は（４８０ティック×拍番号）となる。従って、該当範囲開始時刻ｌＴｉｃｋＦｒｏｍ＝（４８０ティック×４拍×小節番号）＋（４８０ティック×拍番号）＝４８０×（４拍×小節番号＋拍番号）として算出される。また、図１２の該当範囲終了時刻ｌＴｉｃｋＴｏは、現在の拍の終了時刻である。１拍は４８０ティックであるから、該当範囲終了時刻ｌＴｉｃｋＴｏ＝該当範囲開始時刻ｌＴｉｃｋＦｒｏｍ＋４８０＝４８０×（４拍×小節番号＋拍番号＋１）として算出される。

【００９２】

ＣＰＵ１０１は、以上の置換えの後に図１１のフローチャートの処理を動作させることにより、ステップＳ１１１１で、現在のノートイベントのピッチｉＰｉｔｃｈを１２で除算したときの剰余（ｉＰｉｔｃｈ％１２）として算出される現在のノートイベントに対応するピッチクラスにおけるピッチクラスパワーｌＰｉｃｈＣｌａｓｓＰｏｗｅｒ［ｉＰｉｔｃｈ％１２］に、以下の計算値を格納する。ＣＰＵ１０１は、現在のノートイベントのベロシティとパート情報とから定まるベロシティ情報ｌＰｏｗｅｒＷｅｉｇｈｔに、現在のノートイベントの現在の拍の範囲における発音時間長（ｌＴｉｃｋＥｎｄ－ｌＴｉｃｋＳｔａｒｔ）を乗算した値として、上記ピッチクラスパワーｌＰｉｃｈＣｌａｓｓＰｏｗｅｒ［ｉＰｉｔｃｈ％１２］を算出する。このようにして、現在のノートイベントに対応するピッチクラスパワーｌＰｉｃｈＣｌａｓｓＰｏｗｅｒ［ｉＰｉｔｃｈ％１２］の値は、現在の拍の範囲内での現在のノートイベントの発音時間が長ければ長いほど、また発音の強さを示すベロシティが大きければ大きいほど、そして現在のノートイベントが属するパートに応じて、現在のノートイベントに対応するピッチクラス（ｉＰｉｔｃｈ％１２）の音の現在の拍の範囲での構成割合が大きいことになる。

【００９３】

図１４は、図５のステップＳ５０５のマッチング＆結果保存処理の詳細例を示すフローチャートである。

【００９４】

次に、ＣＰＵ１０１は、コードのルート（根音）を示す全てのｉｒｏｏｔの値０から１１について、以下のステップＳ１４０２からＳ１４１３の一連の処理を実行する（ステップＳ１４０１）。更に、ＣＰＵ１０１は、コードの種別を示す全てのコードタイプｉｔｙｐｅの値について、以下のステップＳ１４０３からＳ１４１３の一連の処理を実行する（ステップＳ１４０２）。

【００９５】

ステップＳ１４０３からＳ１４１３の繰返し処理において、ＣＰＵ１０１は始めに、共にＲＡＭ１０３に記憶される変数である第１のパワー評価値ｌＰｏｗｅｒと第２のパワー評価値ｌＯｔｈｅｒＰｏｗｅｒの各値を０にクリアする（ステップＳ１４０３）。

【００９６】

次に、ＣＰＵ１０１は、０から１１までの値を有する全てのピッチクラスｉＰｃ毎に、以下のステップＳ１４０５からＳ１４０７の処理を実行する（ステップＳ１４０４）。

【００９７】

まず、ＣＰＵ１０１は、ステップ１４０４で指定された現在のピッチクラスｉＰｃが、ステップＳ１４０１及びステップＳ１４０２で指定された現在のコードルートｉｒｏｏｔ及びコードタイプｉｔｙｐｅに基づいて定まるコードの構成音（コードトーン）に含まれるか否かを判定する（ステップＳ１４０５）。この判定は、「ｃｈｏｒｄｔｏｎｅ［ｉｔｙｐｅ］［（１２＋ｉＰｃ－ｉｒｏｏｔ）％１２］の値が１であるか否か」を判定する演算である。図１５は、コードトーンの説明図である。図１５において、（ａ）ｍａｊｏｒ、（ｂ）ｍｉｎｏｒ、（ｃ）７ｔｈ、及び（ｄ）ｍｉｎｏｒ７ｔｈの各行は、コードルートがピッチクラス＝０（音名＝Ｃ）である場合における、メジャーコード、マイナーコード、セブンスコード、及びマイナーセブンスコードの各コードタイプにおける構成音のピッチクラス及び音名を示している。各行で、値「１」が記載されているピッチクラス及び音名が、その行に対応するコードの構成音である。値「０」が記載されているピッチクラ

ス及び音名は、その行に対応するコードの構成音でない音が比較対象とされる。図１のＲＯＭ１０２は、コードルートがピッチクラス＝０（音名＝Ｃ）であるときの例えば図１５（ａ）、（ｂ）、（ｃ）、及び（ｄ）の各コードタイプｉｔｙｐｅに対応する配列定数ｃｈｏｒｄｔｏｎｅ［ｉｔｙｐｅ］［ｉ］を記憶している。なお、実際には、ｉｔｙｐｅの種類は、図１５に示される４種類より多い。ここで、ｉは図１５の０から１１までのピッチクラスの値をとり、配列要素値ｃｈｏｒｄｔｏｎｅ［ｉｔｙｐｅ］［ｉ］には、第１配列要素引数ｉｔｙｐｅに対応する図１５（ａ）、（ｂ）、（ｃ）、又は（ｄ）として例示される行の第２配列要素引数ｉに対応するピッチクラスｉにおける値１又は０が格納されている。ＣＰＵ１０１は、ステップＳ１４０５において、まず、第２配列要素引数として「（１２＋ｉＰｃ－ｉｒｏｏｔ）％１２」の値を演算する。この演算では、ステップＳ１４０４で指定されているピッチクラスｉＰｃとステップＳ１４０１で指定されているコードルートｉｒｏｏｔとの差分値が、どのピッチクラスになるかが算出されている。括弧内で１２が加算されているのは「ｉＰｃ－ｉｒｏｏｔ」の値がマイナス値にならないようにするためである。また、「％」は、剰余を求める剰余演算を示している。ＣＰＵ１０１は、この演算結果を第２配列要素引数とし、更にステップＳ１４０２で指定されているｉｔｙｐｅを第１配列要素引数として、ＲＯＭ１０２から読み出した配列要素値ｃｈｏｒｄｔｏｎｅ［ｉｔｙｐｅ］［（１２＋ｉＰｃ－ｉｒｏｏｔ）％１２］の値が１であるか否かを判定する。これにより、ＣＰＵ１０１は、ステップＳ１４０４で指定されているピッチクラスｉＰｃが、図１５に例示されるコードルートがピッチクラス＝０（音名＝Ｃ）のときのコード構成音をコードルートがステップＳ１４０１で指定されているｉｒｏｏｔであるときのコード構成音に変換したときのｉｔｙｐｅに対応する行のコード構成音に含まれているか否かを判定することができる。

【００９８】

ステップ１４０４で指定された現在のピッチクラスｉＰｃが、ステップＳ１４０１で指定された現在のコードルートｉｒｏｏｔ及びステップＳ１４０２で指定された現在のコードタイプｉｔｙｐｅに対応するコードの構成音に含まれている場合（ステップＳ１４０５の判定がＹＥＳの場合）には、ＣＰＵ１０１は、そのピッチクラスｉＰｃに対応して図５のステップＳ５０４で算出されているピッチクラスパワーｌＰｉｃｈＣｌａｓｓＰｏｗｅｒ［ｉＰｃ］を、第１のパワー評価値ｌＰｏｗｅｒに累算する（ステップＳ１４０６）。

【００９９】

一方、ステップ１４０４で指定された現在のピッチクラスｉＰｃが、ステップＳ１４０１で指定された現在のコードルートｉｒｏｏｔ及びステップＳ１４０２で指定された現在のコードタイプｉｔｙｐｅに対応するコードの構成音にに含まれていない場合（ステップＳ１４０５の判定がＮＯの場合）には、ＣＰＵ１０１は、そのピッチクラスｉＰｃに対応して図５のステップＳ５０４で算出されているピッチクラスパワーｌＰｉｃｈＣｌａｓｓＰｏｗｅｒ［ｉＰｃ］を、第２のパワー評価値ｌＯｔｈｅｒＰｏｗｅｒに累算する（ステップＳ１４０７）。

【０１００】

ＣＰＵ１０１は、０から１１までの値を有する全てのピッチクラスｉＰｃについて、上述のステップＳ１４０５からＳ１４０７の処理の実行を終えると（ステップＳ１４０４の判定結果が「終了」を示すと）、次の処理を実行する。ＣＰＵ１０１は、ステップＳ１４０１及びＳ１４０２で現在指定されているコードルート及びコードタイプで定まるコードの構成音中で、図５のステップＳ５０２で現在指定されている小節に対して図５のステップＳ５０１の調判定処理で決定された調の音階音に含まれる音数を、その調の音階音の数で除算した値を、補正係数ＴＮＲとして算出する。即ち、ＣＰＵ１０１は、下記（１）式で示される演算を実行する（ステップＳ１４０８）。

【０１０１】

ＴＮＲ＝（コード構成音中で調の音階音に含まれる音数）÷（調の音階音数）

・・・（１）

【０１０２】

より具体的には、ＣＰＵ１０１は、図５のステップＳ５０２で現在指定されている小節の小節番号を引数として、ＲＡＭ１０３に記憶されている図２（ｂ）のデータフォーマットのポインタ情報ｔｏｎａｌｉｔｙ［小節番号］から図２（ｂ）の調情報を参照する。これにより、ＣＰＵ１０１は、上記小節に対応する調のキー値を、ｔｏｎａｌｉｔｙ［小節番号］．ｉＫｅｙとして取得する。そして、ＣＰＵ１０１は、ＲＯＭ１０２に記憶されているキー値がピッチクラス＝０（音名＝Ｃ）であるときの図１０（ｄ）の統合スケールｓｃａｌｅに対応する配列定数ｓｃａｌｅ［ｉ］の各ｉ毎の音階音を、上記取得したキー値ｔｏｎａｌｉｔｙ［小節番号］．ｉＫｅｙに従って変換する。これにより、ＣＰＵ１０１は、上記取得したキー値ｔｏｎａｌｉｔｙ［小節番号］．ｉＫｅｙに対応した統合スケールｓｃａｌｅの音階音の情報を得る。この音階音を、ステップＳ１４０１とＳ１４０２で現在指定されているコードルート及びコードタイプで定まるコードの構成音と比較することにより、上記（１）式を計算する。

【０１０３】

例えば、調判定結果がハ長調のときの各コードの補正値は、下記のようになる。

Ｇ７：１、Ｂｄｉｍ：１、Ｂｄｉｍ７：０．７５、Ｂｍ７♭５＝１．０、
Ｄｄｉｍ７＝０．７５、Ｆｄｉｍ７＝０．７５

【０１０４】

続いて、ＣＰＵ１０１は、ステップＳ１４０８で算出した補正係数ＴＮＲをステップＳ１４０６で算出された第１のパワー評価値ｌＰｏｗｅｒに乗算し、また、第２のパワー評価値ｌＯｔｈｅｒＰｏｗｅｒに所定の負の定数ＯＰＲを乗算し、両者の乗算結果を加算した結果で、第１のパワー評価値ｌＰｏｗｅｒを置き換えることにより、ステップＳ１４０１とＳ１４０２で現在指定されているコードルート及びコードタイプで定まるコードに対応する新たなパワー評価値ｌＰｏｗｅｒを算出する（ステップＳ１４０９）。

【０１０５】

上述の（１）の補正係数ＴＮＲを介して、本実施形態では、図５のステップＳ５０１での調判定処理による小節毎の調判定の結果を、その小節内の拍毎のコード判定に反映させることが可能となり、精度の高いコード判定が実現される。

【０１０６】

ＣＰＵ１０１は、図３のコード進行データとして得られている現在の拍番号ｌＣｎｔに対応する全てのコード候補の数ｉ（ｉ＝０、１、２、・・・）について、以下のステップＳ１４１１からＳ１４１３の一連の処理を繰り返し実行する（ステップＳ１４１０）。

【０１０７】

この繰返し処理において、ＣＰＵ１０１はまず、現在の拍番号ｌＣｎｔに対応する第ｉ＋１候補（ｉ＝０なら第１候補、ｉ＝１なら第２候補、ｉ＝２なら第３候補、・・・）のポインタ情報ｃｈｏｒｄＰｒｏｇ［ｌＣｎｔ］［ｉ］が参照する、コード情報内のパワー評価値ｃｈｏｒｄＰｒｏｇ［ｌＣｎｔ］［ｉ］．ｄｏＰｏｗｅｒＶａｌｕｅを取得する。ここで、現在の拍番号ｌＣｎｔは、楽曲の先頭からの拍の通し番号であり、４分の４拍子の楽曲の場合、「ｌＣｎｔ＝（４拍×ステップＳ５０２の小節番号）＋（ステップＳ５０３の拍番号）」として算出される。そして、ＣＰＵ１０１は、ステップＳ１４０９で算出したパワー評価値ｌＰｏｗｅｒが、上記ｃｈｏｒｄＰｒｏｇ［ｌＣｎｔ］［ｉ］．ｄｏＰｏｗｅｒＶａｌｕｅの値よりも大きいか否かを判定する（以上、ステップＳ１４１１）。

【０１０８】

ステップＳ１４１１の判定がＮＯの場合、ＣＰＵ１０１は、ステップＳ１４１０の処理に戻って、ｉをインクリメントした次のコード候補に対する処理に移行する。

【０１０９】

ステップＳ１４１１の判定がＹＥＳの場合、ＣＰＵ１０１は、第ｉ＋１番目以降のポインタ情報ｃｈｏｒｄＰｒｏｇ［ｌＣｎｔ］［ｉ＋１］、ｃｈｏｒｄＰｒｏｇ［ｌＣｎｔ］［ｉ＋２］、ｃｈｏｒｄＰｒｏｇ［ｌＣｎｔ］［ｉ＋３］、・・・が参照するコード情報を、順次いままでひとつずつ順位が高かったコード情報を参照するように、参照関係をシフトする。そして、ＣＰＵ１０１は、第ｉ番目のポインタ情報ｃｈｏｒｄＰｒｏｇ［ｌＣ

ｎｔ］［ｉ］が新たに参照するコード情報の保存場所をＲＡＭ１０３上に確保し、その保存場所に新たに判定されたコードに関するコード情報を図３に例示されるデータフォーマットで格納する。

【０１１０】

このコード情報において、ｌＴｉｃｋには現在の小節（ステップＳ５０２で決定）内の現在の拍（ステップＳ５０３で決定）に対応する開始の時刻が格納される。これは、図５のステップＳ５０４のピッチクラスパワー作成処理の説明で前述した該当範囲開始時刻ｌＴｉｃｋＦｒｏｍ＝４８０×（４拍×現在の小節番号＋小節内の現在の拍番号）である。ｉＭｅａｓＮｏには現在の小節の楽曲の先頭の小節を第０小節としてカウントした現在の小節番号が格納される。ｉＴｉｃｋＩｎＭｅａｓには、小節内の現在の拍に対応する開始のティック時刻が格納される。図２（ｂ）の説明で前述したように、ｉＴｉｃｋＩｎＭｅａｓは、１拍目に対応するティック値０、２拍目に対応するティック値４８０、３拍目に対応するティック値９６０、又は４拍目に対応するティック値１４４０の何れかの値となる。ｉＲｏｏｔとｉＴｙｐｅにはそれぞれ、ステップＳ１４０１で指定されている現在のコードルートｉｒｏｏｔ値と、ステップＳ１４０２で指定されている現在のコードタイプｉｔｙｐｅが格納される。ｄｏＰｏｗｅｒＶａｌｕｅにはステップＳ１４０９で算出されたパワー評価値が格納される。その後、ＣＰＵ１０１は、ステップＳ１４１０の処理に戻り、次のコード候補に対する処理に移行する。

【０１１１】

全てのコード候補の数ｉに対する処理が完了すると（ステップＳ１４１０の判定結果が「終了」を示すと）、ＣＰＵ１０１は、ステップＳ１４０２の処理に戻り、次のコードタイプｉｔｙｐｅについての繰返し処理に移行する。

【０１１２】

全てのコードタイプｉｔｙｐｅに対する繰返し処理が完了すると（ステップＳ１４０２の判定結果が「終了」を示すと）、ＣＰＵ１０１は、ステップＳ１４０１の処理に戻り、次のコードルートｉｒｏｏｔについての繰返し処理に移行する。

【０１１３】

全てのコードルートｉｒｏｏｔに対する繰返し処理が完了すると（ステップＳ１４０１の判定結果が「終了」を示すと）、ＣＰＵ１０１は、図１４のフローチャートで例示された図５のステップＳ５０５のマッチング＆結果保存処理を終了する。

【０１１４】

次に、図５のステップＳ５０６の最小コスト計算処理と、ステップＳ５０７の経路確定処理の詳細について、以下に説明する。楽曲データに対するコードの判定において、実際の楽曲で使われるコード以外の音の影響や逆にコードの構成音が常に鳴っていない状況があったりで、従来、適切なコード判定ができない場合があった。例えば、「シレファ」だけの発音の場合、この音を構成音として持つコードは、Ｇ７、Ｂｄｉｍ、Ｂｄｉｍ７、Ｂｍ７♭５、Ｄｄｉｍ７、Ｆｄｉｍ７がある。また、「ド、ド＃、レ、ミb、ミ」の発音の場合、これらの一部を構成音として持つコードは、Ｃａｄｄ９、Ｃｍａｄｄ９、Ｃ＃ｍＭ７などがある。これら複数のコードの候補がある場合にそのコードが存在する拍タイミングのピッチクラスのみから判定するのには困難があり、音楽的知識を使ったり時間軸上の変化要素を考慮するなどの工夫が考えられる。

【０１１５】

一般的には、“ｓｕｓ４”“ｍＭ７”などは、前後のコードの連結について音楽的に自然なルールがある。例えば、“ｓｕｓ４”のコードの次のコードは、同じコードルートを有する場合が多い。また、“ｍＭ７”のコードの前後のコードは、同じコードルートを有し、マイナーコードである場合が多い。

【０１１６】

そこで、本実施形態では、音楽的な連結規則に基づく２つのコード間の連結コストが定義される。そして、ＣＰＵ１０１は、図５のステップＳ５０６において、楽曲の全ての小節及び小節内の全ての拍に対して図３に例示されるデータフォーマットで得られる複数候

補からなるコード進行データの全ての組合せの中から、上記連結コストに基づいて楽曲全体でコストが最小となるコードの組合せを算出する。最小コストの計算には、例えばダイクストラ法などを利用することができる。

【０１１７】

図１６は、最小コスト計算処理と経路確定処理の説明図である。図１６（ａ）は、最小コスト計算処理における経路最適化処理の説明図である。図１６（ｂ）は、最小コスト計算処理及び経路確定処理による経路最適化結果の説明図である。ステップＳ５０６における最小コスト計算処理による経路最適化処理は、拍タイミング毎にコードの候補がｍ個（例えば３個）あるとすると、ｍのコード数の拍数乗の組合せの中から、最小コストとなる経路を求める処理である。以下、ｍ＝３の場合を例に説明する。

【０１１８】

いま、図１６に示されるように、図３のコード進行データとして、各拍タイミングｎ－２、ｎ－１、ｎ、ｎ＋１、・・・毎に、第１候補から第３候補までの３候補ずつのコード候補が得られている。いま、拍タイミングｎを現在の拍タイミングとし、この現在の拍タイミングが、ＲＡＭ１０３に記憶される変数ｌＣｈｏｒｄＩｄｘによって指定されるとする。また、現在の直前の拍タイミングｎ－１が、ＲＡＭ１０３に記憶される変数ｌＰｒｅｖＣｈｏｒｄＩｄｘによって指定されるとする。更に、ｌＣｈｏｒｄＩｄｘで指定される現在の拍タイミングｎにおける各候補番号（０、１、又は２）が、ＲＡＭ１０３に記憶される変数ｉＣｕｒＣｈｏｒｄによって指定されるとする。また、ｌＰｒｅｖＣｈｏｒｄＩｄｘで指定される現在の直前の拍タイミングｎ－１における各候補番号（０、１、又は２）が、ＲＡＭ１０３に記憶される変数ｉＰｒｅｖＣｈｏｒｄによって指定されるとする。

【０１１９】

本実施形態の最小コスト計算処理では、いま、楽曲の先頭の拍タイミングからコードの発音が始まって、各拍タイミング毎にコード候補が選択されながら、現在の拍タイミングｌＣｈｏｒｄＩｄｘにおける現在の候補番号ｉＣｕｒＣｈｏｒｄの現在のコード候補が選択されて発音されるまでにかかるトータルコストを、ＲＡＭ１０３に記憶される配列変数である最適コードトータル最小コストｄｏＯｐｔｉｍｉｚｅＣｈｏｒｄＴｏｔａｌＭｉｎｉｍａｌＣｏｓｔ［ｌＣｈｏｒｄＩｄｘ］［ｉＣｕｒＣｈｏｒｄ］と定義する。このコスト値は、現在の直前の拍タイミングｌＰｒｅｖＣｈｏｒｄＩｄｘにおける３つのコード候補の各々と現在のコード候補との間の各連結コストに、その３つのコード候補の各々において算出されている各最適コードトータル最小コストをそれぞれ加算した各値が最小となる値として算出される。また、その最小値をとる現在の直前の拍タイミングｌＰｒｅｖＣｈｏｒｄＩｄｘにおけるコード候補を、ＲＡＭ１０３に記憶される配列変数である現在のコード候補への直前最適コードルートｉＯｐｔｉｍｉｚｅＣｈｏｒｄＲｏｕｔｅＰｒｅｖ［ｌＣｈｏｒｄＩｄｘ］［ｉＣｕｒＣｈｏｒｄ］と定義する。ＣＰＵ１０１は、図５のステップＳ５０６の最小コスト計算処理において、楽曲の先頭の拍タイミングから楽曲の進行に沿った拍タイミング毎に順次、上述の最小コスト計算処理を実行してゆく。

【０１２０】

図１７は、図５のステップＳ５０６の最小コスト計算処理の詳細例を示すフローチャートである。ＣＰＵ１０１は、ｌＣｈｏｒｄＩｄｘ＝１以降の全ての拍タイミングについて、現在の拍タイミングｌＣｈｏｒｄＩｄｘを指定しながら、ステップＳ１７０２からＳ１７０８までの一連の処理を繰り返し実行する（ステップＳ１７０１）。ｌＣｈｏｒｄＩｄｘ＝０の場合は、それより手前に拍タイミングが存在しないため、計算を行わない。

【０１２１】

次に、ＣＰＵ１０１は、現在の直前の拍タイミングｌＰｒｅｖＣｈｏｒｄＩｄｘに、現在の拍タイミングｌＣｈｏｒｄＩｄｘの値から１減算した値を格納する（ステップＳ１７０２）。

【０１２２】

続いて、ＣＰＵ１０１は、ステップＳ１７０１で指定される現在の拍タイミングｌＣｈｏｒｄＩｄｘ毎に、全てのコード候補について現在の拍タイミングの候補番号ｉＣｕｒＣ

ｈｏｒｄを指定しながら、ステップＳ１７０４からＳ１７０９までの一連の処理を繰り返し実行する（ステップＳ１７０３）。

【０１２３】

更に、ＣＰＵ１０１は、ステップＳ１７０３で指定される現在の拍タイミングの候補番号ｉＣｕｒＣｈｏｒｄ毎に、全ての直前の拍タイミングのコード候補について直前の拍タイミングの候補番号ｉＰｒｅｖＣｈｏｒｄを指定しながら、ステップＳ１７０５からＳ１７０８までの一連の処理を繰り返し実行する（ステップＳ１７０４）。

【０１２４】

ステップＳ１７０５からＳ１７０９までの繰返し処理において、ＣＰＵ１０１はまず、ステップＳ１７０４で指定された直前の拍タイミングの候補番号ｉＰｒｅｖＣｈｏｒｄのコード候補からステップＳ１７０３で指定された現在の拍タイミングの候補番号ｉＣｕｒＣｈｏｒｄのコード候補に遷移するときの連結コストを計算し、その計算結果をＲＡＭ１０３に記憶される変数であるコストｄｏＣｏｓｔに格納する（ステップＳ１７０５）。

【０１２５】

次に、ＣＰＵ１０１は、コストｄｏＣｏｓｔに、ステップＳ１７０３で指定された直前の拍タイミングの候補番号ｉＰｒｅｖＣｈｏｒｄのコード候補に対して保持されている最適コードトータル最小コストｄｏＯｐｔｉｍｉｚｅＣｈｏｒｄＴｏｔａｌＭｉｎｉｍａｌＣｏｓｔ［ｌＰｒｅｖＣｈｏｒｄＩｄｘ］［ｉＰｒｅｖＣｈｏｒｄ］の値を加算する（ステップＳ１７０６）。なお、現在の拍タイミングｌＣｈｏｒｄＩｄｘ＝１で現在の直前の拍タイミングｌＰｒｅｖＣｈｏｒｄＩｄｘ＝０の場合における最適コードトータル最小コストｄｏＯｐｔｉｍｉｚｅＣｈｏｒｄＴｏｔａｌＭｉｎｉｍａｌＣｏｓｔ［０］［ｉＰｒｅｖＣｈｏｒｄ］（ｉＰｒｅｖＣｈｏｒｄ＝０、１、２）の値は０である。

【０１２６】

次に、ＣＰＵ１０１は、ステップＳ１７０６で更新されたコストｄｏＣｏｓｔの値が、ステップＳ１７０３で指定された現在の拍タイミングの候補番号ｉＣｕｒＣｈｏｒｄに対して現在までに得られているＲＡＭ１０３に記憶される変数であるコスト最小値ｄｏＭｉｎ以下であるか否かを判定する（ステップＳ１７０７）。なお、コスト最小値ｄｏＭｉｎの値は、ＣＰＵ１０１が、ステップＳ１７０３で新たな現在の拍タイミングの候補番号ｉＣｕｒＣｈｏｒｄを指定するときに、大きな初期値に設定される。

【０１２７】

ステップＳ１７０７の判定がＮＯならば、ＣＰＵ１０１は、ステップＳ１７０４の処理に戻って、ｉＰｒｅｖＣｈｏｒｄをインクリメントして、直前の拍タイミングの次の候補番号ｉＰｒｅｖＣｈｏｒｄに対する処理に移行する。

【０１２８】

ステップＳ１７０７の判定がＹＥＳならば、ＣＰＵ１０１は、現在までのコスト最小値ｄｏＭｉｎにコストｄｏＣｏｓｔの値を格納し、ＲＡＭ１０３に記憶される変数であるコスト最小直前コードｉＭｉｎＰｒｅｖＣｈｏｒｄに、ステップＳ１７０４で指定されている直前の拍タイミングの候補番号ｉＰｒｅｖＣｈｏｒｄを格納する。更に、ＣＰＵ１０１は、現在の拍タイミングｌＣｈｏｒｄＩｄｘ及び現在の拍タイミングの候補番号ｉＣｕｒＣｈｏｒｄのコード候補に対応する最適コードトータル最小コストｄｏＯｐｔｉｍｉｚｅＣｈｏｒｄＴｏｔａｌＭｉｎｉｍａｌＣｏｓｔ［ｌＣｈｏｒｄＩｄｘ］［ｉＣｕｒＣｈｏｒｄ］に、コストｄｏＣｏｓｔの値を格納する（以上、ステップＳ１７０８）。その後、ＣＰＵ１０１は、ステップＳ１７０４の処理に戻り、ｉＰｒｅｖＣｈｏｒｄをインクリメントして、直前の拍タイミングの次の候補番号ｉＰｒｅｖＣｈｏｒｄに対する処理に移行する。

【０１２９】

以上のステップＳ１７０５からＳ１７０８までの一連の処理がステップＳ１７０４で順次指定される直前の拍タイミングの候補番号ｉＰｒｅｖＣｈｏｒｄ毎に実行され、全ての直前の拍タイミングの候補番号ｉＰｒｅｖＣｈｏｒｄ（＝０、１、２）に対する処理が完了すると、ＣＰＵ１０１は、次の処理を実行する。ＣＰＵ１０１は、現在の拍タイミング

ｌＣｈｏｒｄＩｄｘ及び現在の拍タイミングの候補番号ｉＣｕｒＣｈｏｒｄに対応する直前最適コードルートｉＯｐｔｉｍｉｚｅＣｈｏｒｄＲｏｕｔｅＰｒｅｖ［ｌＣｈｏｒｄＩｄｘ］［ｉＣｕｒＣｈｏｒｄ］に、コスト最小直前コードｉＭｉｎＰｒｅｖＣｈｏｒｄの値を格納する。その後、ＣＰＵ１０１は、ステップＳ１７０３の処理に戻り、ｉＣｕｒＣｈｏｒｄをインクリメントして現在の拍タイミングの次の候補番号ｉＣｕｒＣｈｏｒｄに対する処理に移行する。

【０１３０】

以上のステップＳ１７０４からＳ１７０９までの一連の処理がステップＳ１７０３で順次指定される現在の拍タイミングの候補番号ｉＣｕｒＣｈｏｒｄ毎に実行され、全ての現在の拍タイミングの候補番号ｉＣｕｒＣｈｏｒｄ（＝０、１、２）に対する処理が完了すると、ＣＰＵ１０１は、ステップＳ１７０１の処理に戻り、ｌＣｈｏｒｄＩｄｘをインクリメントして、次の拍タイミングｌＣｈｏｒｄＩｄｘに対する処理に移行する。

【０１３１】

以上のステップＳ１７０２からＳ１７０９までの一連の処理がステップＳ１７０１で順次指定される現在の拍タイミングｌＣｈｏｒｄＩｄｘ毎に実行され、全ての現在の拍タイミングｌＣｈｏｒｄＩｄｘに対する処理が完了すると、ＣＰＵ１０１は、図１７のフローチャートで示される図５のステップＳ５０６の最小コスト計算処理を終了する。

【０１３２】

図１８は、図１７のステップＳ１７０５のコスト計算処理の詳細例を示すフローチャートである。ＣＰＵ１０１はまず、現在の拍タイミングｌＣｈｏｒｄＩｄｘ及び現在の拍タイミングの候補番号ｉＣｕｒＣｈｏｒｄに対応してＲＡＭ１０３に記憶されているコード情報（図３参照）へのポインタ情報ｃｈｏｒｄＰｒｏｇ［ｌＣｈｏｒｄＩｄｘ］［ｉＣｕｒＣｈｏｒｄ］の値をＲＡＭ１０３に記憶される変数である現在ポインタｃｕｒに格納する（ステップＳ１８０１）。

【０１３３】

ＣＰＵ１０１は同様に、現在の直前の拍タイミングｌＰｒｅｖＣｈｏｒｄＩｄｘ及び直前の拍タイミングの候補番号ｉＰｒｅｖＣｈｏｒｄに対応してＲＡＭ１０３に記憶されているコード情報へのポインタ情報ｃｈｏｒｄＰｒｏｇ［ｌＰｒｅｖＣｈｏｒｄＩｄｘ］［ｉＰｒｅｖＣｈｏｒｄ］の値をＲＡＭ１０３に記憶される変数である直前ポインタｐｒｅｖに格納する（ステップＳ１８０２）。

【０１３４】

次に、ＣＰＵ１０１は、連結コストｄｏＣｏｓｔの値を、０．５に初期設定する（ステップＳ１８０３）。

【０１３５】

次に、ＣＰＵ１０１は、現在の拍タイミングｌＣｈｏｒｄＩｄｘの候補番号ｉＣｕｒＣｈｏｒｄのコード情報のコードルートｃｕｒ．ｉＲｏｏｔ（図３参照）に１２を加算した後に、現在の直前の拍タイミングｌＰｒｅｖＣｈｏｒｄＩｄｘの候補番号ｉＰｒｅｖＣｈｏｒｄのコード情報のコードルートｐｒｅｖ．ｉＲｏｏｔを減算し、その結果を１２で除算したときの剰余値が５であるか否かを判定する（ステップＳ１８０４）。

【０１３６】

ステップＳ１８０４の判定がＹＥＳの場合は、現在の直前の拍タイミングｌＰｒｅｖＣｈｏｒｄＩｄｘの候補番号ｉＰｒｅｖＣｈｏｒｄのコード候補から、現在の拍タイミングｌＣｈｏｒｄＩｄｘにおける候補番号ｉＣｕｒＣｈｏｒｄのコード候補への遷移は、音程差が５度のとても自然なコード遷移である。従って、この場合には、ＣＰＵ１０１は、連結コストｄｏＣｏｓｔの値を最も良い値である最低値０．０に設定する（ステップＳ１８０５）。

【０１３７】

ステップＳ１８０４の判定がＮＯの場合は、ＣＰＵ１０１は、ステップＳ１８０５の処理はスキップし、連結コストｄｏＣｏｓｔの値は０．５のままとなる。

【０１３８】

次に、ＣＰＵ１０１は、現在の直前の拍タイミングｌＰｒｅｖＣｈｏｒｄＩｄｘの候補番号ｉＰｒｅｖＣｈｏｒｄのコード情報のコードタイプｐｒｅｖ．ｉＴｙｐｅ（図３参照）が“ｓｕｓ４”であって、かつ、そのコード情報のコードルートｐｒｅｖ．ｉＲｏｏｔと、現在の拍タイミングｌＣｈｏｒｄＩｄｘの候補番号ｉＣｕｒＣｈｏｒｄのコード情報のコードルートｃｕｒ．ｉＲｏｏｔとが同じであるか否かを判定する（ステップＳ１８０６）。

【０１３９】

ステップＳ１８０６の判定がＹＥＳの場合は、「“ｓｕｓ４”のコードの次のコードは同じコードルートを有する場合が多い」という音楽ルールに良く合っており、とても自然なコード遷移である。従って、この場合には、ＣＰＵ１０１は、連結コストｄｏＣｏｓｔの値を最も良い値である最低値０．０に設定する（ステップＳ１８０７）。

【０１４０】

ステップＳ１８０６の判定がＮＯの場合には、かなり不自然なコード遷移になるため、この場合には、ＣＰＵ１０１は、連結コストｄｏＣｏｓｔの値を悪い値１．０に設定する（ステップＳ１８０８）。

【０１４１】

次に、ＣＰＵ１０１は、現在の直前の拍タイミングｌＰｒｅｖＣｈｏｒｄＩｄｘの候補番号ｉＰｒｅｖＣｈｏｒｄのコード情報のコードタイプｐｒｅｖ．ｉＴｙｐｅが“ｍＭ７”であって、かつ、現在の拍タイミングｌＣｈｏｒｄＩｄｘの候補番号ｉＣｕｒＣｈｏｒｄのコード情報のコードタイプｃｕｒ．ｉＴｙｐｅが“ｍ７”であって、なおかつ、両方のコード情報のコードルートｐｒｅｖ．ｉＲｏｏｔとｃｕｒ．ｉＲｏｏｔとが同じであるか否かを判定する（ステップＳ１８０９）。

【０１４２】

ステップＳ１８０９の判定がＹＥＳの場合も、音楽ルールに良く合っていてとても自然なコード遷移であるため、この場合も、ＣＰＵ１０１は、連結コストｄｏＣｏｓｔの値を最も良い値である最低値０．０に設定する（ステップＳ１８１０）。

【０１４３】

ステップＳ１８０９の判定がＮＯの場合には、かなり不自然なコード遷移になるため、この場合には、ＣＰＵ１０１は、連結コストｄｏＣｏｓｔの値を悪い値１．０に設定する（ステップＳ１８１１）。

【０１４４】

更に、ＣＰＵ１０１は、現在の直前の拍タイミングｌＰｒｅｖＣｈｏｒｄＩｄｘの候補番号ｉＰｒｅｖＣｈｏｒｄのコード情報のコードタイプｐｒｅｖ．ｉＴｙｐｅが“ｍａｊ”であって、かつ、現在の拍タイミングｌＣｈｏｒｄＩｄｘの候補番号ｉＣｕｒＣｈｏｒｄのコード情報のコードタイプｃｕｒ．ｉＴｙｐｅが“ｍ”であって、なおかつ、両方のコード情報のコードルートｐｒｅｖ．ｉＲｏｏｔとｃｕｒ．ｉＲｏｏｔとが同じであるか否かを判定する（ステップＳ１８１２）。

【０１４５】

ステップＳ１８１２の判定がＹＥＳの場合は、不自然なコード遷移になるため、ＣＰＵ１０１は、連結コストｄｏＣｏｓｔに悪い値１．０を設定する（ステップＳ１８１３）。

【０１４６】

ステップＳ１８１２の判定がＮＯの場合は、ＣＰＵ１０１は、ステップＳ１８１３の処理はスキップする。

【０１４７】

最後に、ＣＰＵ１０１は、連結コストｄｏＣｏｓｔに、１から現在の拍タイミングｌＣｈｏｒｄＩｄｘの候補番号ｉＣｕｒＣｈｏｒｄのコード情報のパワー評価値ｃｕｒ．ｄｏＰｏｗｅｒＶａｌｕｅを減算した結果と、１から現在の直前の拍タイミングｌＰｒｅｖＣｈｏｒｄＩｄｘの候補番号ｉＰｒｅｖＣｈｏｒｄのコード情報のパワー評価値ｐｒｅｖ．ｄｏＰｏｗｅｒＶａｌｕｅを減算した結果とを乗算して、連結コストｄｏＣｏｓｔの値を調整する（ステップＳ１８１４）。その後、ＣＰＵ１０１は、図１８のフローチャートで

示される図１７のステップＳ１７０５のコスト計算処理を終了する。

【０１４８】

図１６（ｂ）には、説明の簡単化のために、候補数を２、拍タイミングを０、１、２、３のみとした場合における、上述の図１７の最小コスト計算処理による最小コスト計算結果の例を示してある。図１６（ｂ）において、大きな丸印は判定されたコード候補を示している。また、丸印間を連結する直線矢印付近に記載された数値は、その直線矢印の始点の丸印のコード候補から終点の丸印のコード候補への連結コストｄｏＣｏｓｔを示している。拍タイミング＝０では、Ｃｍａｊが第１コード候補、Ｃｍが第２コード候補として判定されたとする。拍タイミング＝１では、Ａｍが第１コード候補、ＡｍＭ７が第２コード候補として判定されたとする。拍タイミング＝２では、Ｄｍが第１コード候補、Ｄｓｕｓ４が第２コード候補として判定されたとする。そして、拍タイミング＝３では、Ｇ７が第１コード候補、Ｂｄｉｍが第２コード候補として判定されたとする。

【０１４９】

図１７の最小コスト計算処理において、まず、現在の拍タイミングｌＣｈｏｒｄＩｄｘ＝１で、候補番号ｉＣｕｒＣｈｏｒｄ＝０（第１候補）の場合、現在のコード候補として”Ａｍ”が得られている。この場合、現在の直前の拍タイミングｌＰｒｅｖＣｈｏｒｄＩｄｘ＝０において、候補番号ｉＰｒｅｖＣｈｏｒｄ＝０（第１候補）の直前のコード候補“Ｃｍａｊ”から現在のコード候補”Ａｍ”への連結コストｄｏＣｏｓｔは、図１８のフローチャートのアルゴリズムにより０．５と計算される。また、候補番号ｉＰｒｅｖＣｈｏｒｄ＝１（第２候補）の直前のコード候補“Ｃｍ”から現在のコード候補”Ａｍ”への連結コストｄｏＣｏｓｔも、図１８のフローチャートのアルゴリズムにより０．５と計算される。直前のコード候補“Ｃｍａｊ”及び“Ｃｍ”の各最適コードトータル最小コストｄｏＯｐｔｉｍｉｚｅＣｈｏｒｄＴｏｔａｌＭｉｎｉｍａｌＣｏｓｔ［０］［０／１］は、共に０である。図１７のステップＳ１７０７では、連結コストｄｏＣｏｓｔとコスト最小値ｄｏＭｉｎが同値の場合にはあとのコード候補が優先される。従って、現在のコード候補“Ａｍ”の最適コードトータル最小コストｄｏＯｐｔｉｍｉｚｅＣｈｏｒｄＴｏｔａｌＭｉｎｉｍａｌＣｏｓｔ［１］［０］は、“Ａｍ”の丸印の内部に示されるように０．５と計算される。また、現在のコード候補“Ａｍ”に対する直前最適コードルートｉＯｐｔｉｍｉｚｅＣｈｏｒｄＲｏｕｔｅＰｒｅｖ［１］［０］としては、“Ａｍ”の丸印に入力する太線矢印として示されるように直前のコード候補“Ｃｍ”が設定される。

【０１５０】

現在の拍タイミングｌＣｈｏｒｄＩｄｘ＝１で、候補番号ｉＣｕｒＣｈｏｒｄ＝１（第２候補）の場合のコード候補”ＡｍＭ７”についても同様の計算が実行される。現在のコード候補“ＡｍＭ７”の最適コードトータル最小コストｄｏＯｐｔｉｍｉｚｅＣｈｏｒｄＴｏｔａｌＭｉｎｉｍａｌＣｏｓｔ［１］［１］は、“ＡｍＭ７”の丸印の内部に示されるように０．５と計算される。また、現在のコード候補“ＡｍＭ７”に対する直前最適コードルートｉＯｐｔｉｍｉｚｅＣｈｏｒｄＲｏｕｔｅＰｒｅｖ［１］［１］としては、“ＡｍＭ７”の丸印に入力する太線矢印として示されるように直前のコード候補“Ｃｍ”が設定される。

【０１５１】

次に、現在の拍タイミングが１つ進んでｌＣｈｏｒｄＩｄｘ＝２となり、候補番号ｉＣｕｒＣｈｏｒｄ＝０（第１候補）の場合、現在のコード候補として“Ｄｍ”が得られている。この場合、現在の直前の拍タイミングｌＰｒｅｖＣｈｏｒｄＩｄｘ＝１において、候補番号ｉＰｒｅｖＣｈｏｒｄ＝０（第１候補）の直前のコード候補“Ａｍ”から現在のコード候補”Ｄｍ”への連結コストｄｏＣｏｓｔは、図１８のフローチャートのアルゴリズムにより０．０と計算される。また、候補番号ｉＰｒｅｖＣｈｏｒｄ＝１（第２候補）の直前のコード候補“ＡｍＭ７”から現在のコード候補”Ｄｍ”への連結コストｄｏＣｏｓｔは、図１８のフローチャートのアルゴリズムにより１．０と計算される。直前のコード候補“Ａｍ”及び“ＡｍＭ７”の各最適コードトータル最小コストｄｏＯｐｔｉｍｉｚｅＣｈｏｒｄＴｏｔａｌＭｉｎｉｍａｌＣｏｓｔ［１］［０／１］は、共に０．５である。

従って、直前のコード候補“Ａｍ”から現在のコード候補”Ｄｍ”への図１７のステップＳ１７０６で修正されたコストｄｏＣｏｓｔの値は０．５＋０．０＝０．５となる。同様に、直前のコード候補“ＡｍＭ７”から現在のコード候補”Ｄｍ”への修正されたコストｄｏＣｏｓｔの値は０．５＋１．０＝１．５となる。従って、現在のコード候補“Ｄｍ”の最適コードトータル最小コストｄｏＯｐｔｉｍｉｚｅＣｈｏｒｄＴｏｔａｌＭｉｎｉｍａｌＣｏｓｔ［２］［０］は、“Ｄｍ”の丸印の内部に示されるように０．５と計算される。また、現在のコード候補“Ｄｍ”に対する直前最適コードルートｉＯｐｔｉｍｉｚｅＣｈｏｒｄＲｏｕｔｅＰｒｅｖ［２］［０］としては、“Ｄｍ”の丸印に入力する太線矢印として示されるように直前のコード候補“Ａｍ”が設定される。

【０１５２】

現在の拍タイミングｌＣｈｏｒｄＩｄｘ＝２で、候補番号ｉＣｕｒＣｈｏｒｄ＝１（第２候補）の場合のコード候補”Ｄｓｕｓ４”についても同様の計算が実行される。現在のコード候補“Ｄｓｕｓ４”の最適コードトータル最小コストｄｏＯｐｔｉｍｉｚｅＣｈｏｒｄＴｏｔａｌＭｉｎｉｍａｌＣｏｓｔ［２］［１］は、“Ｄｓｕｓ４”の丸印の内部に示されるように０．５と計算される。また、現在のコード候補“Ｄｓｕｓ４”に対する直前最適コードルートｉＯｐｔｉｍｉｚｅＣｈｏｒｄＲｏｕｔｅＰｒｅｖ［２］［１］としては、“Ｄｓｕｓ４”の丸印に入力する太線矢印として示されるように直前のコード候補“Ａｍ”が設定される。

【０１５３】

次に、現在の拍タイミングが更に１つ進んでｌＣｈｏｒｄＩｄｘ＝３となり、候補番号ｉＣｕｒＣｈｏｒｄ＝０（第１候補）の場合、現在のコード候補として“Ｇ７”が得られている。この場合、現在の直前の拍タイミングｌＰｒｅｖＣｈｏｒｄＩｄｘ＝２において、候補番号ｉＰｒｅｖＣｈｏｒｄ＝０（第１候補）の直前のコード候補“Ｄｍ”から現在のコード候補”Ｇ７”への連結コストｄｏＣｏｓｔは、図１８のフローチャートのアルゴリズムにより０．０と計算される。また、候補番号ｉＰｒｅｖＣｈｏｒｄ＝１（第２候補）の直前のコード候補“Ｄｓｕｓ４”から現在のコード候補”Ｇ７”への連結コストｄｏＣｏｓｔは、図１８のフローチャートのアルゴリズムにより１．０と計算される。直前のコード候補“Ｄｍ”及び“Ｄｓｕｓ４”の各最適コードトータル最小コストｄｏＯｐｔｉｍｉｚｅＣｈｏｒｄＴｏｔａｌＭｉｎｉｍａｌＣｏｓｔ［２］［０／１］は、共に０．５である。従って、直前のコード候補“Ｄｍ”から現在のコード候補“Ｇ７”への修正されたコストｄｏＣｏｓｔの値は０．５＋０．０＝０．５となる。同様に、直前のコード候補“Ｄｓｕｓ４”から現在のコード候補“Ｇ７”への修正されたコストｄｏＣｏｓｔの値は０．５＋１．０＝１．５となる。従って、現在のコード候補“Ｇ７”の最適コードトータル最小コストｄｏＯｐｔｉｍｉｚｅＣｈｏｒｄＴｏｔａｌＭｉｎｉｍａｌＣｏｓｔ［３］［０］は、“Ｇ７”の丸印の内部に示されるように０．５と計算される。また、現在のコード候補“Ｇ７”に対する直前最適コードルートｉＯｐｔｉｍｉｚｅＣｈｏｒｄＲｏｕｔｅＰｒｅｖ［３］［０］としては、“Ｇ７”の丸印に入力する太線矢印として示されるように直前のコード候補“Ｄｍ”が設定される。

【０１５４】

現在の拍タイミングｌＣｈｏｒｄＩｄｘ＝３で、候補番号ｉＣｕｒＣｈｏｒｄ＝１（第２候補）の場合のコード候補”Ｂｄｉｍ”についても同様の計算が実行される。現在のコード候補“Ｂｄｉｍ”の最適コードトータル最小コストｄｏＯｐｔｉｍｉｚｅＣｈｏｒｄＴｏｔａｌＭｉｎｉｍａｌＣｏｓｔ［３］［１］は、“Ｂｄｉｍ”の丸印の内部に示されるように１．０と計算される。また、現在のコード候補“Ｂｄｉｍ”に対する直前最適コードルートｉＯｐｔｉｍｉｚｅＣｈｏｒｄＲｏｕｔｅＰｒｅｖ［３］［１］としては、“Ｂｄｉｍ”の丸印に入力する太線矢印として示されるように直前のコード候補“Ｄｍ”が設定される。

【０１５５】

次に、図５のステップＳ５０７の経路確定処理について説明する。経路確定処理において、ＣＰＵ１０１は、末尾の拍タイミングから先頭の拍タイミングに逆方向に向かって、

拍タイミングｌＣｈｏｒｄＩｄｘ毎及び候補番号ｉＣｕｒＣｈｏｒｄ毎のコード候補について算出された最適コードトータル最小コストｄｏＯｐｔｉｍｉｚｅＣｈｏｒｄＴｏｔａｌＭｉｎｉｍａｌＣｏｓｔ［ｌＣｈｏｒｄＩｄｘ］［ｉＣｕｒＣｈｏｒｄ］の小さい値を探しながら、また、直前最適コードルートｉＯｐｔｉｍｉｚｅＣｈｏｒｄＲｏｕｔｅＰｒｅｖ［ｌＣｈｏｒｄＩｄｘ］［ｉＣｕｒＣｈｏｒｄ］を辿りながら、拍タイミング毎にコード候補を選択してゆき、選択されたコード候補を第１候補に置き換えていく。

【０１５６】

図１６（ｂ）の例では、まず、末尾の拍タイミングｌＣｈｏｒｄＩｄｘ＝３において、最適コードトータル最小コストの値が最小値０．５である候補番号ｉＣｕｒＣｈｏｒｄ＝０のコード候補“Ｇ７”が選択され、ｌＣｈｏｒｄＩｄｘ＝３における第１候補とされる。次に、ｌＣｈｏｒｄＩｄｘ＝３で第１候補となったコード候補“Ｇ７”に設定されている直前最適コードルートｉＯｐｔｉｍｉｚｅＣｈｏｒｄＲｏｕｔｅＰｒｅｖ［３］［０］が参照されることにより、１つ手前の拍タイミングｌＣｈｏｒｄＩｄｘ＝２において、候補番号ｉＣｕｒＣｈｏｒｄ＝０のコード候補 “Ｄｍ”が選択され、ｌＣｈｏｒｄＩｄｘ＝２における第１候補とされる。続いて、ｌＣｈｏｒｄＩｄｘ＝２で第１候補となったコード候補“Ｄｍ”に設定されている直前最適コードルートｉＯｐｔｉｍｉｚｅＣｈｏｒｄＲｏｕｔｅＰｒｅｖ［２］［０］が参照されることにより、１つ手前の拍タイミングｌＣｈｏｒｄＩｄｘ＝１において、候補番号ｉＣｕｒＣｈｏｒｄ＝０のコード候補 “Ａｍ”が選択され、ｌＣｈｏｒｄＩｄｘ＝１における第１候補とされる。最後に、ｌＣｈｏｒｄＩｄｘ＝１で第１候補となったコード候補“Ａｍ”に設定されている直前最適コードルートｉＯｐｔｉｍｉｚｅＣｈｏｒｄＲｏｕｔｅＰｒｅｖ［１］［０］が参照されることにより、１つ手前の先頭の拍タイミングｌＣｈｏｒｄＩｄｘ＝０において、候補番号ｉＣｕｒＣｈｏｒｄ＝１のコード候補 “Ｃｍ”が選択され、ｌＣｈｏｒｄＩｄｘ＝０における第１候補とされる。以上の経路確定処理の結果、楽曲の先頭の拍タイミングから順次、各拍タイミングの第１候補のコード候補“Ｃｍ”、“Ａｍ”、“Ｄｍ”、及び“Ｇ７”が最適なコード進行として選択され、表示手段１０５等に表示される。

【０１５７】

図１９は、図５のステップＳ５０７の経路確定処理の詳細例を示すフローチャートであり、上述した動作を実現する。ＣＰＵ１０１はまず、全ての拍タイミングについて、末尾の拍タイミングから先頭の拍タイミングに向かって、現在の拍タイミングｌＣｈｏｒｄＩｄｘをディクリメントしながら指定してゆき、ｌＣｈｏｒｄＩｄｘ毎に、ステップＳ１９０２からＳ１９０６までの一連の処理を繰り返し実行する（ステップＳ１９０１）。

【０１５８】

ステップＳ１９０２からＳ１９０６までの一連の繰返し処理において、ＣＰＵ１０１はまず、終端コードがあるか否か、即ち末尾の拍タイミングを指定しているか否かを判定する（ステップＳ１９０２）。

【０１５９】

次に、ＣＰＵ１０１は、ステップＳ１９０１で指定される末尾の拍タイミングｌＣｈｏｒｄＩｄｘについて、全てのコード候補について末尾の拍タイミングの候補番号ｉＣｕｒＣｈｏｒｄを指定しながら、ステップＳ１９０４からＳ１９０６までの一連の処理を繰り返し実行する（ステップＳ１９０３）。この処理は、図１６（ｂ）で説明したように、末尾の拍タイミングｌＣｈｏｒｄＩｄｘにおいて、最適コードトータル最小コストｄｏＯｐｔｉｍｉｚｅＣｈｏｒｄＴｏｔａｌＭｉｎｉｍａｌＣｏｓｔ［ｌＣｈｏｒｄＩｄｘ］［ｉＣｕｒＣｈｏｒｄ］の値が最小となる候補番号ｉＣｕｒＣｈｏｒｄを探索する処理である。

【０１６０】

ステップＳ１９０４からＳ１９０６までの一連の繰返し処理において、ＣＰＵ１０１は、ステップＳ１９０１で指定されているｌＣｈｏｒｄＩｄｘとステップＳ１９０３で指定されているｉＣｕｒＣｈｏｒｄに対応する最適コードトータル最小コストｄｏＯｐｔｉｍｉｚｅＣｈｏｒｄＴｏｔａｌＭｉｎｉｍａｌＣｏｓｔ［ｌＣｈｏｒｄＩｄｘ］［ｉＣｕｒ

Ｃｈｏｒｄ］の値が、ＲＡＭ１０３に記憶されている変数であるコスト最小値ｄｏＭｉｎ以下である否かを判定する（ステップＳ１９０４）。コスト最小値ｄｏＭｉｎの値は、図１９のフローチャートの処理の開始時に大きな値に初期設定されている。

【０１６１】

ステップＳ１９０４の判定がＮＯならば、ＣＰＵ１０１は、ステップＳ１９０３の処理に戻り、ｉＣｕｒＣｈｏｒｄをインクリメントして、次の候補番号ｉＣｕｒＣｈｏｒｄに対する処理に移行する。

【０１６２】

ステップＳ１９０４の判定がＹＥＳになると、ＣＰＵ１０１は、コスト最小値ｄｏＭｉｎに、ステップＳ１９０１で指定されているｌＣｈｏｒｄＩｄｘとステップＳ１９０３で指定されているｉＣｕｒＣｈｏｒｄに対応する最適コードトータル最小コストｄｏＯｐｔｉｍｉｚｅＣｈｏｒｄＴｏｔａｌＭｉｎｉｍａｌＣｏｓｔ［ｌＣｈｏｒｄＩｄｘ］［ｉＣｕｒＣｈｏｒｄ］の値を格納する（ステップＳ１９０５）。

【０１６３】

そして、ＣＰＵ１０１は、ＲＡＭ１０３に記憶される変数である最適コード候補番号ｉＣｈｏｒｄＢｅｓｔに、ステップＳ１９０３で現在指定されているｉＣｕｒＣｈｏｒｄの値を格納する（ステップＳ１９０６）。その後、ＣＰＵ１０１は、ステップＳ１９０３の処理に戻り、ｉＣｕｒＣｈｏｒｄをインクリメントして、次の候補番号ｉＣｕｒＣｈｏｒｄに対する処理に移行する。

【０１６４】

以上のようにして、ステップＳ１９０４からＳ１９０６の一連の処理の実行がｉＣｕｒＣｈｏｒｄとして指定される全ての候補番号について完了すると、ＣＰＵ１０１は、ステップＳ１９０８の処理に移行する。この状態において、最適コード候補番号ｉＣｈｏｒｄＢｅｓｔに、末尾の拍タイミングにおいて、最適コードトータル最小コストが最小となるコード候補の候補番号が得られている。ステップＳ１９０８において、ＣＰＵ１０１は、現在の末尾の拍タイミングｌＣｈｏｒｄＩｄｘと最適コード候補番号ｉＣｈｏｒｄＢｅｓｔに対応するコード情報のコードルートｃｈｏｒｄＰｒｏｇ［ｌＣｈｏｒｄＩｄｘ］［ｉＣｈｏｒｄＢｅｓｔ］．ｉＲｏｏｔの値を、現在の末尾の拍タイミングｌＣｈｏｒｄＩｄｘの第１候補のコード情報のコードルートｃｈｏｒｄＰｒｏｇ［ｌＣｈｏｒｄＩｄｘ］［０］．ｉＲｏｏｔに格納する（ステップＳ１９０８）。

【０１６５】

次に、ＣＰＵ１０１は、現在の末尾の拍タイミングｌＣｈｏｒｄＩｄｘと最適コード候補番号ｉＣｈｏｒｄＢｅｓｔに対応するコード情報のコードタイプｃｈｏｒｄＰｒｏｇ［ｌＣｈｏｒｄＩｄｘ］［ｉＣｈｏｒｄＢｅｓｔ］．ｉＴｙｐｅの値を、現在の末尾の拍タイミングｌＣｈｏｒｄＩｄｘの第１候補のコード情報のコードタイプｃｈｏｒｄＰｒｏｇ［ｌＣｈｏｒｄＩｄｘ］［０］．ｉＴｙｐｅに格納する（ステップＳ１９０９）。

【０１６６】

その後、ＣＰＵ１０１は、現在の末尾の拍タイミングｌＣｈｏｒｄＩｄｘと最適コード候補番号ｉＣｈｏｒｄＢｅｓｔに対応するコード候補の直前最適コードルートｉＯｐｔｉｍｉｚｅＣｈｏｒｄＲｏｕｔｅＰｒｅｖ［ｌＣｈｏｒｄＩｄｘ］［ｉＣｈｏｒｄＢｅｓｔ］の値を、直前の拍タイミングの候補番号ｉＰｒｅｖＣｈｏｒｄに格納する（ステップＳ１９１０）。そして、ＣＰＵ１０１は、ステップＳ１９０１の処理に戻り、ｌＣｈｏｒｄＩｄｘをデクリメントして、１つ手前の拍タイミングｌＣｈｏｒｄＩｄｘに対応する処理に移行する。

【０１６７】

末尾から手前の拍タイミングになると、ステップＳ１９０２の判定がＮＯとなる。この結果、ＣＰＵ１０１は、ステップＳ１９１０で直前の拍タイミングの候補番号ｉＰｒｅｖＣｈｏｒｄに格納されている直前最適コードルートを、最適コード候補番号ｉＣｈｏｒｄＢｅｓｔに格納する（ステップＳ１９０７）。

【０１６８】

続いて、ＣＰＵ１０１は、前述したステップＳ１９０８、Ｓ１９０９を実行することにより、現在の拍タイミングｌＣｈｏｒｄＩｄｘと最適コード候補番号ｉＣｈｏｒｄＢｅｓｔに対応するコード情報のコードルートｃｈｏｒｄＰｒｏｇ［ｌＣｈｏｒｄＩｄｘ］［ｉＣｈｏｒｄＢｅｓｔ］．ｉＲｏｏｔとコードタイプｃｈｏｒｄＰｒｏｇ［ｌＣｈｏｒｄＩｄｘ］［ｉＣｈｏｒｄＢｅｓｔ］．ｉＴｙｐｅの各値を、現在の拍タイミングｌＣｈｏｒｄＩｄｘの第１候補のコード情報のコードルートｃｈｏｒｄＰｒｏｇ［ｌＣｈｏｒｄＩｄｘ］［０］．ｉＲｏｏｔとコードタイプｃｈｏｒｄＰｒｏｇ［ｌＣｈｏｒｄＩｄｘ］［０］．ｉＴｙｐｅに格納する。

【０１６９】

その後、ＣＰＵ１０１は、現在の末尾の拍タイミングｌＣｈｏｒｄＩｄｘと最適コード候補番号ｉＣｈｏｒｄＢｅｓｔに対応するコード候補の直前最適コードルートｉＯｐｔｉｍｉｚｅＣｈｏｒｄＲｏｕｔｅＰｒｅｖ［ｌＣｈｏｒｄＩｄｘ］［ｉＣｈｏｒｄＢｅｓｔ］の値を、直前の拍タイミングの候補番号ｉＰｒｅｖＣｈｏｒｄに格納する（ステップＳ１９１０）。そして、ＣＰＵ１０１は、再びステップＳ１９０１の処理に戻り、ｌＣｈｏｒｄＩｄｘをデクリメントして、１つ手前の拍タイミングｌＣｈｏｒｄＩｄｘに対応する処理に移行する。

【０１７０】

以上の処理が、拍タイミングｌＣｈｏｒｄＩｄｘ毎に繰り返し実行されることにより、各拍タイミングｌＣｈｏｒｄＩｄｘの第１候補のコード情報のコードルートｃｈｏｒｄＰｒｏｇ［ｌＣｈｏｒｄＩｄｘ］［０］．ｉＲｏｏｔとコードタイプｃｈｏｒｄＰｒｏｇ［ｌＣｈｏｒｄＩｄｘ］［０］．ｉＴｙｐｅとして、最適なコード進行を出力することができる。

【０１７１】

以上説明した図５のステップＳ５０６の最小コスト計算処理では、コードの連結規則を使うので、複数候補が出てきたときに、より自然なコード判定結果を得ることが可能となる。

【０１７２】

以上説明した実施形態により、転調も適切に判定できる調判定の結果からより適切なコード判定を行うことが可能となる。

【０１７３】

以上説明した実施形態では、楽曲データ例として、ＭＩＤＩシーケンスデータからのコード判定について説明したが、音楽音響信号からのコード判定を行ってもよい。その場合は、高速フーリエ変換などの音響分析を行うことにより、ピッチクラスパワーを求めることになる。

【０１７４】

以上の実施形態に関して、更に以下の付記を開示する。

（付記１）

楽曲を第１長で区切った第１区間の構成音に基づいて第１調を推定し、前記第１区間と少なくとも部分的に重なる区間であって、前記楽曲を前記第１長と異なる長さの第２長で区切った第２区間の構成音に基づいて第２調を推定する調推定処理と、

前記推定された前記第１調及び前記第２調を比較することにより最適な調を決定する調決定処理と、

を実行するコード解析装置。

（付記２）

前記最適な調に基づいて前記楽曲の前記第１区間のコードを判定するコード判定処理を実行し、更に前記コード判定処理は、前記楽曲の小節を区分した拍毎に、当該拍の構成音を判定し、当該構成音に基づいて当該拍のコードを判定する、付記１に記載のコード解析装置。

（付記３）

前記第１区間、前記第２区間、又は前記拍毎の構成音の判定は、当該第１区間、当該第

２区間、又は当該拍の期間内でノートオンしている前記楽曲の楽音毎に、当該楽音のベロシティと当該期間内での発音時間長とに基づいて決定されるパワー情報値を当該楽音のピッチに対応するピッチクラスに累算することにより、当該第１区間、当該第２区間、又は当該拍における前記ピッチクラス毎のパワー情報累算値を算出する処理である、付記２に記載のコード解析装置。

（付記４）

前記第１区間、前記第２区間、又は拍毎に、前記第１調、前記第２調、又はコードの候補に対応して、前記ピッチクラスの各々が前記第１調、前記第２調の候補の音階音又はコードの候補の構成音と一致する場合に当該ピッチクラスに対して算出されている前記パワー情報累算値を第１のパワー評価値に累算し、一致しない場合に当該ピッチクラスに対して算出されている前記パワー情報累算値を第２のパワー評価値に累算し、前記第１調、前記第２調、又はコードの候補毎に算出される前記第１のパワー評価値及び前記第２のパワー評価値を比較することにより、当該第１区間、当該第２区間、又は拍における前記第１調、前記第２調、又は前記コードを判定する、付記３に記載のコード解析装置。

（付記５）

前記第１区間の区間長は１小節の長さであり、前記第２区間の区間長は１小節の倍数であり、前記調決定処理は、前記第１区間と前記第２区間とで重なる小節毎に、当該小節毎に判定された前記第１調及び前記第２調を比較することにより、当該小節に対応する前記最適な調を決定する、付記１に記載のコード解析装置。

（付記６）

前記調決定処理は、区間開始位置を１小節ずつずらしながら前記第１区間の区間長又は前記第２区間の区間長で楽曲を区切って前記第１区間又は前記第２区間を決定する、付記５に記載のコード解析装置。

（付記７）

前記判定されたコードを表示する表示処理を更に実行する、付記２乃至６の何れかに記載のコード解析装置。

（付記８）

コード解析装置の処理部が、

楽曲を第１長で区切った第１区間の構成音に基づいて第１調を推定し、前記第１区間と少なくとも部分的に重なる区間であって、前記楽曲を前記第１長と異なる長さの第２長で区切った第２区間の構成音に基づいて第２調を推定し、

前記推定された前記第１調及び前記第２調を比較することにより最適な調を決定する、

処理を実行するコード解析方法。

（付記９）

コード解析を行うコンピュータに、

楽曲を第１長で区切った第１区間の構成音に基づいて第１調を推定し、前記第１区間と少なくとも部分的に重なる区間であって、前記楽曲を前記第１長と異なる長さの第２長で区切った第２区間の構成音に基づいて第２調を推定するステップと、

前記推定された前記第１調及び前記第２調を比較することにより最適な調を決定するステップと、

を実行させるためのプログラム。

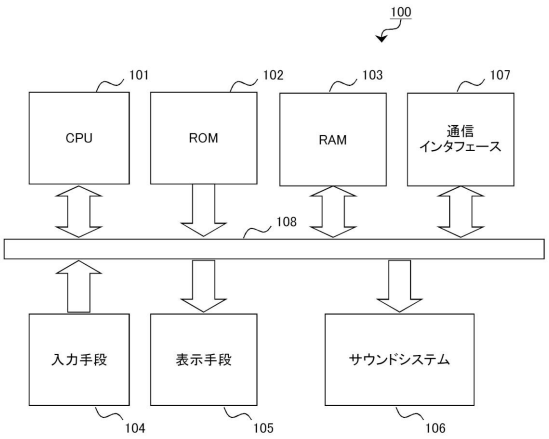
【符号の説明】

【０１７５】

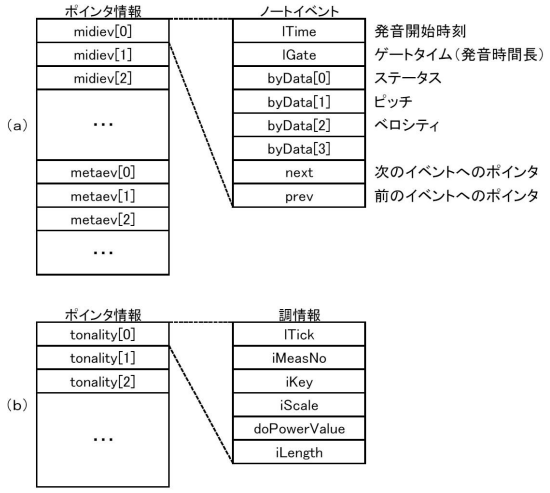
１０１　ＣＰＵ
１０２　ＲＯＭ
１０３　ＲＡＭ
１０４　入力手段
１０５　表示手段
１０６　サウンドシステム
１０７　通信インタフェース

１０８　バス

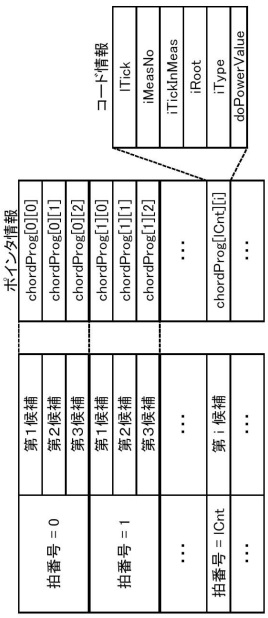
【図１】


100
101
102
103
107
CPU
ROM
RAM
通信
インタフェース
108
入力手段
表示手段
サウンドシステム
104
105
106

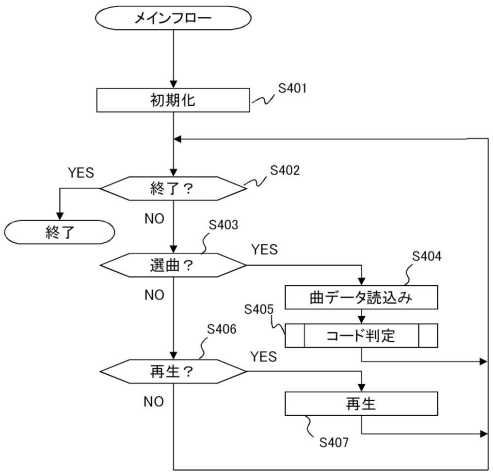
【図２】


ポインタ情報
ノートイベント
midiev[0]
midiev[1]
midiev[2]
…
metaev[0]
metaev[1]
metaev[2]
…
(a)
lTime
lGate
byData[0]
byData[1]
byData[2]
byData[3]
next
prev
発音開始時刻
ゲートタイム（発音時間長）
ステータス
ピッチ
ベロシティ
次のイベントへのポインタ
前のイベントへのポインタ
ポインタ情報
調情報
tonality[0]
tonality[1]
tonality[2]
…
(b)
lTick
iMeasNo
iKey
iScale
doPowerValue
iLength

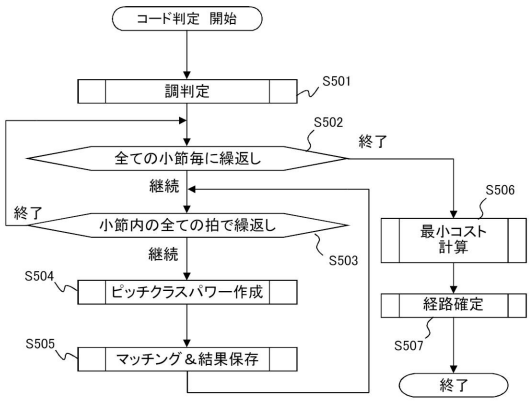
【図３】


コード情報
lTick
iMeasNo
iTickInMeas
iRoot
iType
doPowerValue
ポインタ情報
chordProg[0][0]
chordProg[0][1]
chordProg[0][2]
chordProg[1][0]
chordProg[1][1]
chordProg[1][2]
chordProg[lCnt][i]
第1候補
第2候補
第3候補
第i候補
拍番号 = 0
拍番号 = 1
拍番号 = lCnt

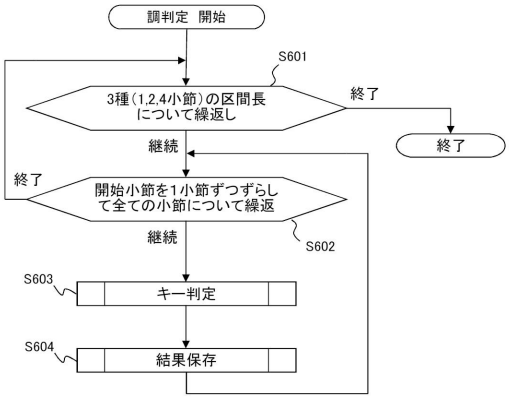
【図４】


メインフロー
初期化
S401
終了？
S402
YES
NO
終了
選曲？
S403
YES
NO
曲データ読込み
S404
コード判定
S405
再生？
S406
YES
NO
再生
S407

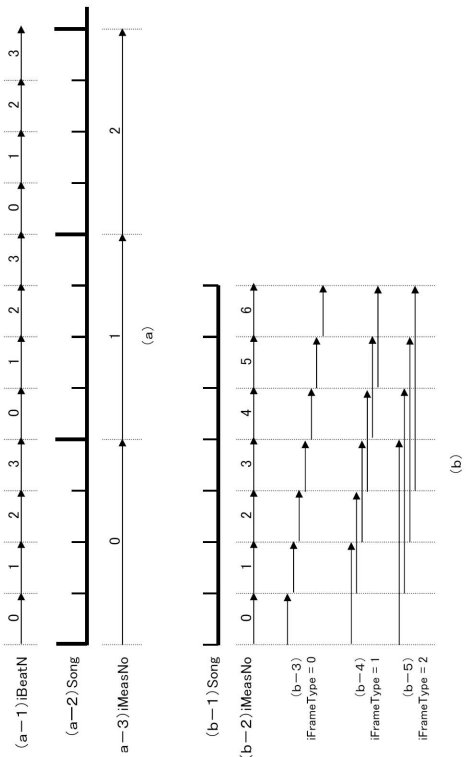
【図５】


コード判定 開始
調判定
S501
全ての小節毎に繰返し
S502
終了
継続
小節内の全ての拍で繰返し
S503
継続
ピッチクラスパワー作成
S504
マッチング＆結果保存
S505
最小コスト計算
S506
経路確定
S507
終了

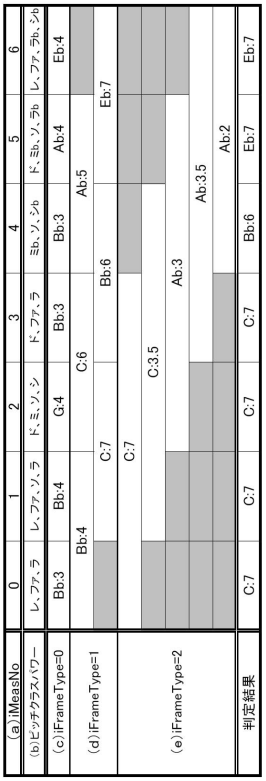
【図６】


調判定 開始
3種(1,2,4小節)の区間長について繰返し
S601
終了
継続
開始小節を1小節ずつずらして全ての小節について繰返
S602
継続
キー判定
S603
結果保存
S604
終了

【図７】

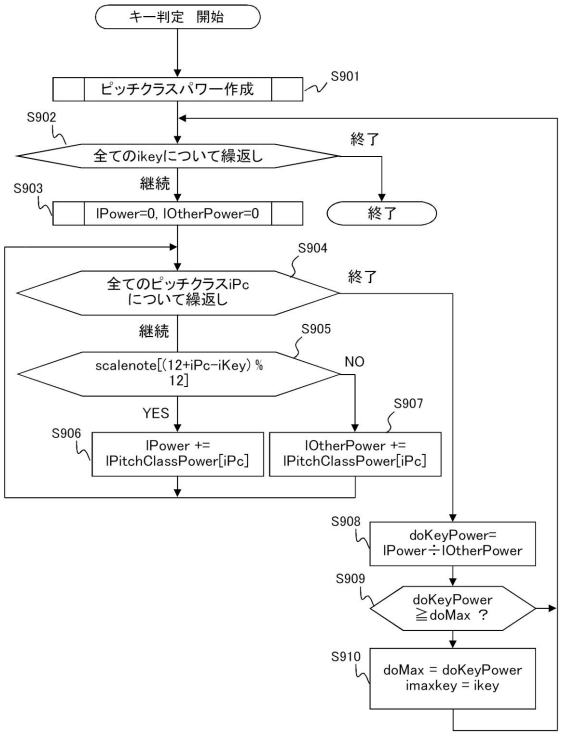

(a−1) iBeatN
(a−2) Song
a−3) iMeasNo
(a)
(b−1) Song
(b−2) iMeasNo
(b−3) iFrameType = 0
(b−4) iFrameType = 1
(b−5) iFrameType = 2
(b)

【図８】



(a)iMeasNo	0	1	2	3	4	5	6
(b)ピッチクラスパワー	レ、ファ、ラ	レ、ファ、ラ	ド、ミ、ソ、シ	ド、ファ、ラ	ミb、ソ、シb	ド、ミb、ソ、ラb	レ、ファ、ラb、シb
(c)iFrameType=0	Bb:3	Bb:4	G:4	Bb:3	Bb:3	Ab:4	Eb:4
(d)iFrameType=1	Bb:4	C:7	C:6	Bb:6	Ab:5	Eb:7	
(e)iFrameType=2	C:7	C:3.5	Ab:3	Ab:3.5			
判定結果	C:7	C:7	C:7	C:7	Bb:6	Eb:7	Eb:7

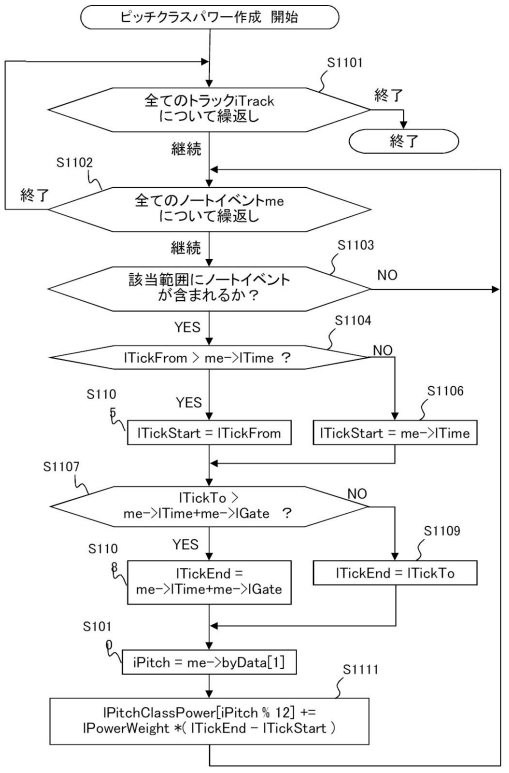
【図９】



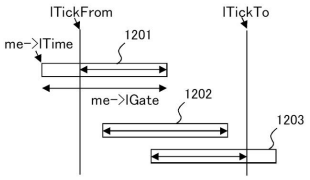
【図１０】

ピッチクラス / スケール	0	1	2	3	4	5	6	7	8	9	10	11
音名	C	C#	D	E♭	E	F	F#	G	A♭	A	B♭	B
(a)major	1	0	1	0	1	1	0	1	0	1	0	1
(b)hminor	1	0	1	0	1	1	0	0	1	1	0	1
(c)mminor	1	0	1	0	1	0	1	0	1	1	0	1
(d)scale	1	0	1	0	1	1	1	1	1	1	0	1

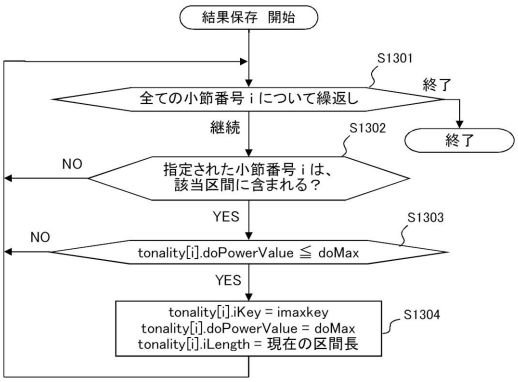
【図１１】



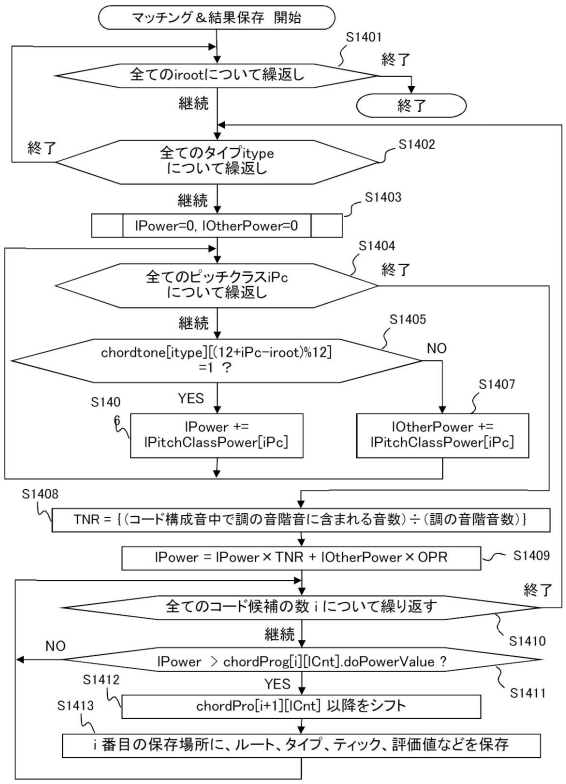
【図１２】



【図１３】



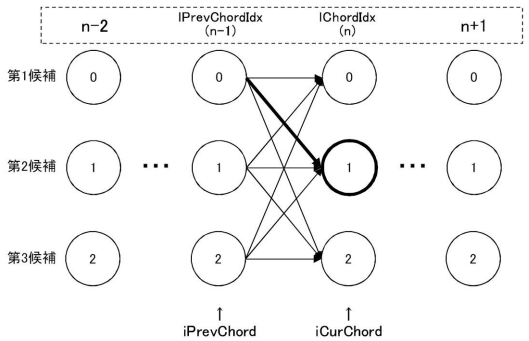
【図１４】



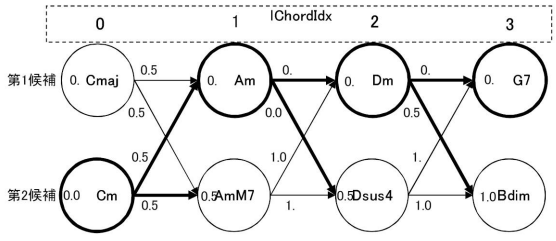
【図１５】

ピッチクラス / 音名 / itype	0 C	1 C#	2 D	3 E♭	4 E	5 F	6 F#	7 G	8 A♭	9 A	10 B♭	11 B
(a) major	1	0	0	0	1	0	0	1	0	0	0	0
(b) minor	1	0	0	1	0	0	0	1	0	0	1	0
(c) 7th	1	0	0	0	1	0	0	1	0	0	1	0
(d) minor7th	1	0	0	1	0	0	0	1	0	0	1	0

【図１６】

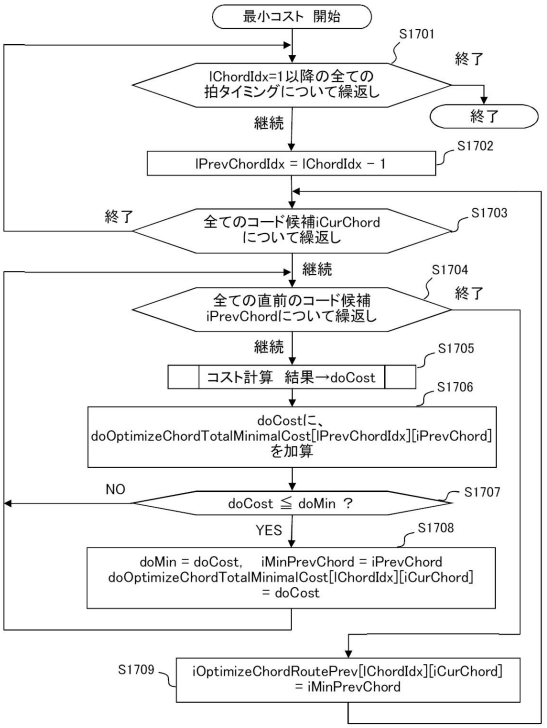


(a)

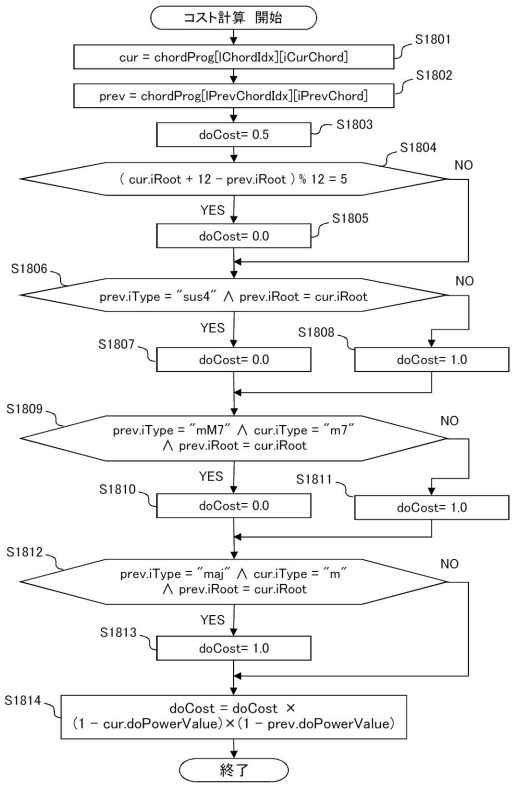


(b)

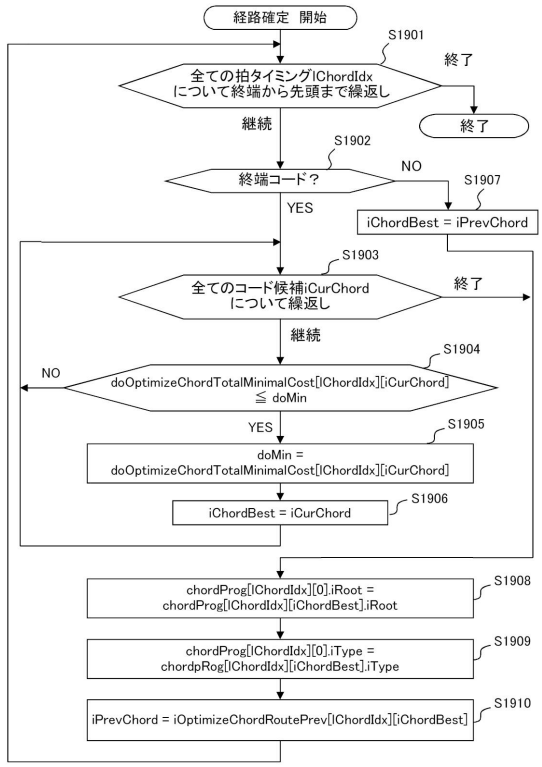
【図１７】


最小コスト 開始
S1701
lChordIdx=1以降の全ての
拍タイミングについて繰返し
終了
終了
継続
S1702
lPrevChordIdx = lChordIdx − 1
終了
全てのコード候補iCurChord
について繰返し
S1703
継続
S1704
全ての直前のコード候補
iPrevChordについて繰返し
終了
継続
S1705
コスト計算 結果→doCost
S1706
doCostに、
doOptimizeChordTotalMinimalCost[lPrevChordIdx][iPrevChord]
を加算
NO
doCost ≦ doMin ？
S1707
YES
S1708
doMin = doCost, iMinPrevChord = iPrevChord
doOptimizeChordTotalMinimalCost[lChordIdx][iCurChord]
= doCost
S1709
iOptimizeChordRoutePrev[lChordIdx][iCurChord]
= iMinPrevChord

【図１８】


コスト計算 開始
S1801
cur = chordProg[lChordIdx][iCurChord]
S1802
prev = chordProg[lPrevChordIdx][iPrevChord]
S1803
doCost= 0.5
S1804
(cur.iRoot + 12 − prev.iRoot) % 12 = 5
NO
YES
S1805
doCost= 0.0
S1806
prev.iType = "sus4" ∧ prev.iRoot = cur.iRoot
NO
YES
S1807
doCost= 0.0
S1808
doCost= 1.0
S1809
prev.iType = "mM7" ∧ cur.iType = "m7"
∧ prev.iRoot = cur.iRoot
NO
YES
S1810
doCost= 0.0
S1811
doCost= 1.0
S1812
prev.iType = "maj" ∧ cur.iType = "m"
∧ prev.iRoot = cur.iRoot
NO
YES
S1813
doCost= 1.0
S1814
doCost = doCost ×
(1 − cur.doPowerValue)×(1 − prev.doPowerValue)
終了

【図１９】


経路確定 開始
S1901
全ての拍タイミングlChordIdx
について終端から先頭まで繰返し
終了
終了
継続
S1902
終端コード？
NO
S1907
YES
iChordBest = iPrevChord
S1903
全てのコード候補iCurChord
について繰返し
終了
継続
S1904
NO
doOptimizeChordTotalMinimalCost[lChordIdx][iCurChord]
≦ doMin
YES
S1905
doMin =
doOptimizeChordTotalMinimalCost[lChordIdx][iCurChord]
S1906
iChordBest = iCurChord
S1908
chordProg[lChordIdx][0].iRoot =
chordProg[lChordIdx][iChordBest].iRoot
S1909
chordProg[lChordIdx][0].iType =
chordpRog[lChordIdx][iChordBest].iType
S1910
iPrevChord = iOptimizeChordRoutePrev[lChordIdx][iChordBest]

フロントページの続き

(56)参考文献 特開平０５－１７３５５７（ＪＰ，Ａ）
特開平０５－３４６７８１（ＪＰ，Ａ）
特開平１１－１０９９７２（ＪＰ，Ａ）
特開平１１－１２６０７５（ＪＰ，Ａ）
特開昭６３－０８０２９９（ＪＰ，Ａ）

(58)調査した分野(Int.Cl.，ＤＢ名)
Ｇ１０Ｈ　　１／００－７／１２
Ｇ１０Ｇ　　１／００－３／０４