卒業論文

Q 学習を用いたメロディの作成

５年　情報工学科　７番
熊谷　洋希

指導教官　東海林　智也

目次

1. 目的

近年コンピュータが身近になりパソコンを用いて曲を作ることが出来るようなソフトウェアを簡単に入手できるようになった.それに伴い楽器を演奏できない人でも,楽器を演奏することなく曲を作ることが出来るようになった.その結果,趣味で作曲を行うような人も表れた.しかし,曲を作るためには音楽についての知識は必要不可欠であり,本人の感性や経験の問題などもあり素人が曲を作るということが難しいという状況に変わりは無い.そこでこの研究において,音楽についての知識の有無に関わりなく誰もが簡単に曲を作ることが出来るような自動作曲を行うツールを作成することにした.

自動作曲を行う手法は種々存在する.例えば,『知識ベースを用いる手法』,『あらかじめ用意されたパターンを組み合わせる手法』,『数値データを与えて音楽理論に基づいて編集する手法』,などがあげられる[1].最近では多項式などの数学的手法を利用した自動作曲の研究や,脳波や,DNA などの生体のゆらぎと呼ばれるものを用いた自動作曲の研究も行われている.さらには図形とその色彩から音符を生成したり,動画の場面変化から音符を生成するなどの自動作曲の研究や,遺伝的アルゴリズムを用いた自動作曲も行われている[2].『知識ベースを用いる手法』とは,既存の曲から『音符の遷移』,『リズム』,『コード進行』などを取り出して知識ベースを作成し,それをもとに曲を作成する手法である.『あらかじめ用意されたパターンを組み合わせる手法』とは,あらかじめ音符の遷移やリズムの変化のパターンを決めてデータベースを作成し,そのパターンをランダムにあるいはあるルールに基づいて組み合わせて曲を作成する手法である.『数値データを与えて音楽理論に基づいて編集する手法』とは,乱数などの数値データを音楽理論として用意したルールを用いて音符やリズムなどに換えて曲を作る手法である.多くの場合,これらの手法を単体で使用するのではなく,複数の手法を組み合わせて自動作曲を行う.例をあげると,多項式によって五線譜上に描かれた曲線にあらかじめ用意された音符のパターンを配置して,知識ベースによっておかしな箇所を修正するという手法を用いた自動作曲ソフトが存在する[3][4].このような手法がある中で今回の研究では『知識ベースを用いる方法』を自動作曲の方法として選択した.その理由としては数学的知識がそれほど必要ないこと,生体のゆらぎのようなものを測定する必要がないことなどがあげられるが,最も大きな理由は知識ベースに知識ベースを作成する際に使用した曲の特徴を保存することが出来ることであり,これによってジャンルに関する知識がない人が自動作曲ソフトを使用する際に迫られることがあるジャンル選択をなくすことが出来るのではないかと考えたからである.ジャンル選択を迫られる自動作曲ソフトでは,ジャンルについての知識がなければどのような雰囲気を持った曲が出力されるのかについての概要さえ掴むことが出来ない.しかし,曲を作る際に知識ベースを用い,知識ベースを作る際に自分が作りたい曲と似た雰囲気を持つ曲を入力として与えることが出来るのならば,ジャンルについての知識がなくても必要とする雰囲気を持つ曲を作ることが出来るはずである.これにより,ジャンルについての知識を持っているのであれば,曲を作る際にそのジャンルに属する曲を入力として与え,ジャンルについての知識を持っていなくても,作りたい曲と似た雰囲気を持つ曲を入力として与えることで其々が必要とする曲を出力することが出来るのではないかと考えられる.

この研究ではこのような理由と手法より自動作曲ツールを作成するが,曲というものは様々な要素が絡み合って作られている.そのため,今回の研究のみで完全な曲を作成することは難しいと思われる.であるから,今回は曲の中でも主要な位置を占めるメロディの自動生成を行う手法につい

て考察,提案を行うこととした.その手法を考察するにあたって先ほど述べた知識ベースと強化学習の手法の 1 つである Q 学習を用いることとした.そこで提案する手法としては,知識ベースに既存の曲から抽出した特徴を記憶させ,知識ベースのなかにあるデータを Q 学習を用いてコンピュータに学習させることによって,知識ベースが持つデータの中でも最適な組み合わせを見つけてそれをメロディを出力させるというものである.さらに,シミュレーションを行うため考察に基づいたプログラムの作成とその評価を行う.

2. Q 学習について

Q 学習とは強化学習の手法の一つであり,方策オフ型 TD 学習に分類される.マルコフ決定過程において最適解に収束することが知られている.Q 学習での学習目的は,初期状態から終端状態に至るまでに得られる累積報酬が最大になる行動群を見つけることである[5][6][7][8][9].

2.1. Q 学習の学習サイクル

Q 学習の学習サイクルは以下のようになる.

ここで学習者をエージェントと呼ぶ.

(1)エージェントは『環境』から現在の『状態』を取得する.

(2)状態に応じて,実行可能な行動群の中から『評価関数』に基づいて最もよいと考えられる行動を一つ選択してそれを環境に対して実行する.

(3)エージェントは環境から行った行動の結果として『行動の価値』を受け取る.この価値はエージェントが環境にいい影響を与えた場合は『報酬』となり,そうでない場合は『罰』となる.

(4)エージェントは結果として得た価値に基づいて評価関数を更新する.

(5)状態の遷移を行う.

エージェントは(1)から(5)までを一つのエピソードとして,エージェントの状態が終端状態になるまでこれを繰り返す.この学習サイクルを図示したものを図 1 に示す.

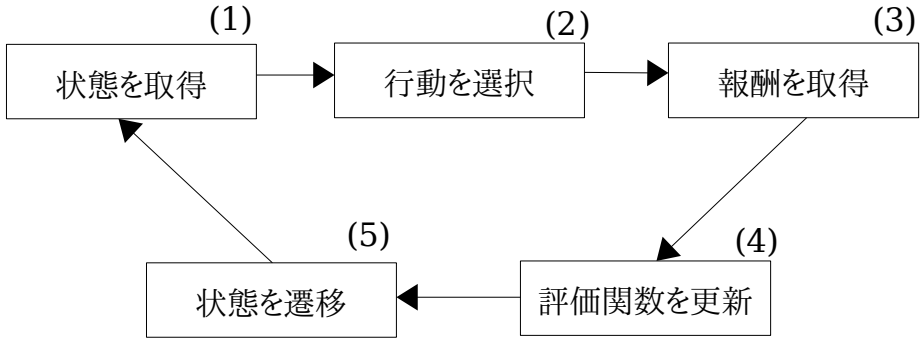

図1. Q 学習の学習サイクル

2.2. Q 学習の評価関数

Q 学習の評価関数は,状態と行動から表される関数で表現され,Q(s,a)となる.ここで[s:状態],[a:行動]とする.エージェントはこの関数をもとに現在の状態 s において評価関数の値が最大となっている行動 a を選択し,環境に対して実行する.

この関数は,環境から受け取る行動の価値によって更新されて増減して行き,最終的に最適解となる行動が最大の値を持つ用になる.図 2 に評価関数の更新式を記す.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_t, a) - Q(s_t, a_t)]$$

$Q(s_t, a_t)$: t時間における価値観数

s_t : t時間における状態

a_t : t時間において取った行動

a: 状態s_tにおいて実行できる行動

α : 学習率

γ : 割引率

図 2． 評価関数の更新式

2.3. メロディ作成での使用

Q 学習が最適解を求められることからメロディの生成にこれを応用する.具体的には度数分布よりデータを与え音同士をつなぎ合わせてそれをメロディとする.そのメロディがメロディを構成するために必要なルールに従っているようであれば報酬を与え,ルールから外れているようであれば罰を与えることで,作られるメロディを制御しデータより得られるメロディをルールに従う最適なものにできると考える.

3　メロディ

メロディとは音楽を構成する要素の 1 つであり,『リズム』,『ハーモニー』と並んで音楽の 3 要素と呼ばれる.この研究では,このメロディを作成することを目的としている.メロディは種々の条件を持ち,クラシックの場合メロディを作る際にいくつかのルールを使用する[10][11][12][13][14].

3.1.　メロディと呼ばれる

良いメロディと呼ばれるメロディには様々な条件が存在する.例えば,『音が跳びすぎない』,『調性感やコード感がある』,『リズミカルである』,『繰り返しと変化を適度に含んでいる』などがあげられ,この他にも様々な条件を持っている[15].

今回の研究では「音が跳びすぎない」という点のみに主点をおいて研究を進めた.

3.2.　音の進行

『音が跳びすぎない』ということがどういうことかを説明するため例を示す(図 3).

左側の図では前の音と後の音の高さの差が小さくなっている.このような進行を順次進行と呼ぶ.右の図では前の音と後の音の高さの差が大きくなっている.このような進行を跳躍進行と呼ぶ.一般的に五線譜で線 1 本分以上離れている場合に跳躍進行であるとされる[16][17].『音が跳びすぎない』と言うことはあまりにも大きな跳躍進行を行ってはいけないと言うことである.

音が跳んでいない
(隣の音との高さの差が小さい)

音が跳んでいる
(隣の音との高さの差が大きい)

図 3．　音の高さの差

一般的にメロディの進行は順次進行と跳躍進行によって構成されているが,前の音と後の音の高さの差に大きな変化がなく,音符を線で結んだ場合滑らかな曲線を描く(図 4).このことから人に自然なメロディであると認められるためにはなだらかな変化を持つ『音が跳びすぎないメロディ』を構成する必要があると考えられる.それを実現するために作曲家が用いるような作曲に関するルールを使用する.

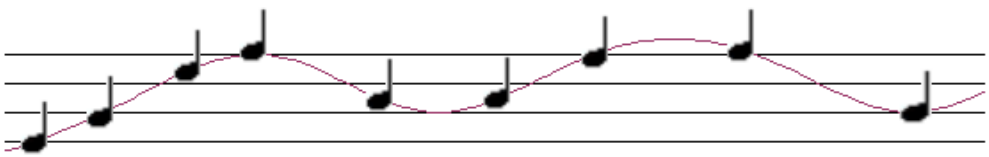

図 4．　メロディの流れの例

3.3. クラシックの教え

クラシックの理論では作曲の基本とされているルールが存在する.この研究では,このルールのうちのいくつかを利用する.以下に基本とされるルールを記す[18].

1. メロディは順次進行を中心に作り,跳躍進行は 3 回以上続けない
2. 大きな跳躍進行の後は,逆方向へ進行させる
3. 1つのフレーズの中に音の頂点を2つ以上作らない
4. 大きな跳躍新行の前は,逆方向への順次進行をおく

今回は 1,2 の教えより,2 つのルールを作ってそれを利用することで学習中のエージェントに報酬を与えるか,罰を与えるかを決めることとした.以下にそのルールを示す.

1.メロディは順次進行を中心に作る
2.大きな跳躍進行の後は,逆方向へ進行させる

4. 度数分布

自動作曲を行う方法はいくつか存在するが,今回はその中の 1 つである知識ベースを参照する自動作曲を行う[19].知識ベースを用いた自動作曲では,既存曲よりなんらかの特徴をとらえてその特徴より知識ベースを作成し,知識ベースを参照しながら作曲を行う方法である.今回は音程の遷移回数記録した度数分布と,音符の遷移回数記録した度数分布を知識ベースとして用いる.

例えば以下のような音程進行(図 5)があったとき,それぞれの遷移を調べて頻度を数えて表を作る(表 1,表 2).

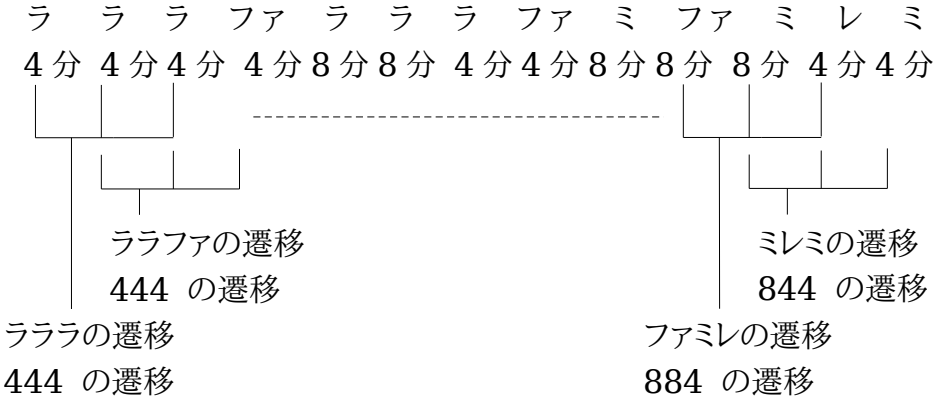

図 5． 音程と音符の遷移

1 音め	2 音め	3 音め	回数
ラ	ラ	ラ	2
ラ	ラ	ファ	2
ラ	ファ	ラ	1
ファ	ラ	ラ	1
ラ	ファ	ミ	1
ファ	ミ	ファ	1
ミ	ファ	ミ	1
ファ	ミ	レ	1
ミ	レ	ファ	1

表 1． 音程の度数分布

1 音め	2 音め	3 音め	回数
4	4	4	2
4	4	8	2
4	8	8	2
8	8	4	2
8	4	4	2
8	8	8	1

表 2． 音符の度数分布

このようにして度数分布表を作成する.ただ,曲は調ごとに根音と言うものが存在しているため,表 1 のようにラ,シ,ドなどの絶対的な音階を用いて処理を行う場合問題が起こる.そのため,曲の根音を基軸とした相対的音階を用いて度数分布表を作成する.表 1 をドを根音とした相対的音階で表すと表 3 の用になる.ただし,[ド = 0, レ = 1, ミ = 2, ファ= 3, ソ = 4, ラ = 5, シ = 6]のように音階に数字を割り当てる.

1音め	*2音め*	*3音め*	*回数*
ラ-ド(５)	ラ-ド(５)	ラ-ド(５)	2
ラ-ド(５)	ラ-ド(５)	ファ-ド(３)	2
ラ-ド(５)	ファ-ド(３)	ラ-ド(５)	1
ファ-ド(３)	ラ-ド(５)	ラ-ド(５)	1
ラ-ド(５)	ファ-ド(３)	ミ-ド(２)	1
ファ-ド(３)	ミ-ド(２)	ファ-ド(３)	1
ミ-ド(２)	ファ-ド(３)	ミ-ド(２)	1
ファ-ド(３)	ミ-ド(２)	レ-ド(１)	1
ミ-ド(２)	レ-ド(１)	ファ-ド(３)	1

表３.　相対的音階の度数分布表例

音符の度数分布には手を加えず,そのまま利用している.このような度数分布を用いることで,不自然な音の遷移をなくし,メロディの変化を滑らかなものに出来ると考えられる.

5. メロディの生成方法

メロディの生成方法に付いて記述する.メロディの生成は大まかにわけて以下の手順で行う.

(1)度数分布の作成
(2)Q 学習による音データの作成と長さの決定
(3)(1)から(2)までを複数回繰り返し音楽ファイルとして出力する.

今回出力に用いる音楽ファイルの形式としては MIDI ファイルを選択した.

5.1. 度数分布の作成

度数分布は楽譜を MML(Music Macro Language)に直し,MML を読み込んで度数分布を出力するプログラムを用いて作成する.図 6 に出力される度数分布の例を示す.

ID	1音め	2音め	3音め	度数
25	11	21	16	1
26	11	21	19	1
27	11	22	21	1
28	11	23	21	2
29	12	10	12	1
30	12	11	11	1
31	12	11	20	1
32	12	11	23	1
33	12	12	13	2
34	12	12	17	4
35	12	12	20	5
36	12	13	13	1
37	12	13	14	1
38	12	14	16	1
39	12	14	17	1

図 6． 度数分布例

各根音を 12 として相対的な高さを数値で表して記録している.例えば「12 が ド」であったとすると「13 は ド♯」,「11 は 低いシ」,と言うようになる.これを 0 から 23 までの範囲で記録している.休符は 24 として,度数が 0 のところはカットする.今回 29 曲の MML を作成し,度数分布の作成には 14 から 17 曲程度をランダムに選び度数分布を作成するたびに内容が変わるように作った.

5.1.1. MML(Music Macro Language) について

MML(Music Macro Language)とは,コンピュータ上の楽譜となるような簡易言語である.MML では楽譜上の音楽記号を英字や数字で表しテキストファイルとして保存する.例として図7,図 8,図 9 を示す.図 7 に示した楽譜を MML に直すと図 8 のようになる.この時各記号の意味を以下に示す.

- D,C,E,G:　順にレ,ド,ミ,ソ,を表す
- D4:　4 分音符のレを表す
- D2:　2 分音符のレを表す

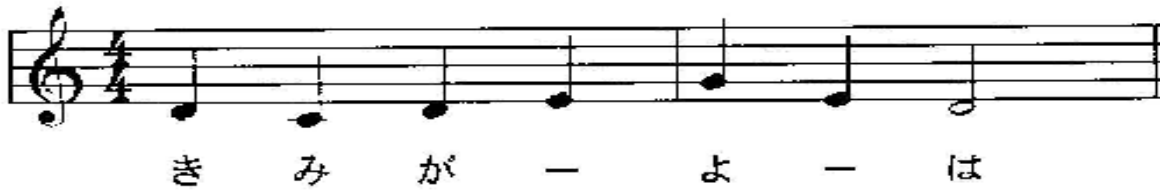

図 7.　楽譜例

D4 C4 D4 E4 G4 E4 D2

図 8.　MML に直した例

C = ド D = レ E = ミ F = ファ G = ソ A = ラ B = シ C+ (D-) = ド# (レ♭) D+ (E-) = レ# (ミ♭) F+ (G-) = ファ# (ソ♭) G+ (A-) = ソ# (ラ♭) A+ (B-) = ラ# (シ♭)	1 = 全音符 2 = 二分音符 4 = 四分音符 8 = 八分音符 16 = 十六部音符
Cn・Dn・En・Fn・Gn・An・Bn	音名　n :長さ　(例:D4)
+	音を半音上げる　(例:D+)
−	音を半音下げる　(例:B-)
Rn	休符　n:長さ　(例:R4)
.	付点(音の長さを 1.5 倍にする)(例:C4.)
Tn	テンポ　n: 0 から 240　(例:T120)
!n	ボリューム　n: 1 から 6　(例:!6)
>	オクターブを 1 つあげる
<	オクターブを 1 つあげる

図 9．　使用した MML の仕様

5.2. Q 学習による音データの作成と長さの決定

Q 学習を用いた音データの作成は以下の手順で行う.

(1)マップの作成
(2)エージェントの学習
(3) 音データの作成
(4) 音データの長さの決定

エージェントが移動するマップを作成する.マップ内には度数分布の ID を配置し,エージェントが学習を行うための初期位置と,目的位置を作る.エージェントはマップ内を移動しながら初期位置から目的位置までの最短経路を学習する.学習終了の後,エージェントが目的地に到達するまでにたどった道のりから ID を取り出して,ID に対応するデータを組み合わせ数字の列を作る.この数字の列を対応する音に変換し音データを作成する.音データを生成した後に,累積分布を用いて音データの長さを決定する.

5.2.1. マップについて

マップは 2 次元配列を用いて作る.2 次元配列の端にあたる部分には-99 を入れ,それ以外の部分にはエージェントが移動できるように度数分布の ID を配置する.エージェントが移動を開始する初期位置を決め,そこから順に ID を配置して,最後に辿り着くべき目的位置を決める.初期位置は配列内の上から 2 行めに,目的位置は下から 2 行めにそれぞれ配置する.配置には乱数を用い,それぞれランダムに配置を行う.図 7 にその例を示す.図 10 において(S)が初期位置,(G)が目的位置,-99 が移動不可能な位置を其々表している.

-99	-99	-99	-99	-99	-99	-99
-99	541	574(S)	252	218	500	-99
-99	194	195	248	500	445	-99
-99	26	27	209	286	281	-99
-99	244	229	378	307	294	-99
-99	191	191	270	308	525	-99
-99	233	132	313	307	305	-99
-99	133	46	308	317	309	-99
-99	46	344	25	320	415	-99
-99	148	118	154	511	452	-99
-99	137	134	306	189	296	-99
-99	51	51	512	302	296	-99
-99	32	32	517	455	517	-99
-99	85	167	514	519	528	-99
-99	166	167	521(G)	520	527	-99
-99	-99	-99	-99	-99	-99	-99

図 10 . 作成されるマップの例

5.2.2. エージェントの学習の行いかたについて

エージェントは初期位置から目的位置までの経路のうち報酬を最大にできる経路を最適経路として学習する.そのためにマップ内を動きまわって移動が可能な経路を探索する.エージェントは上下左右斜めへの移動が可能であるが,負の ID を持つ位置への移動は出来ない.また,各位置に割り当てられた ID に対応するデータを無視した移動も出来ない.具体的には現在位置の ID に対応するデータの「2 音め」,「3 音め」が移動の候補としてあがった位置の ID に対応する「1 音め」,「2 音め」と其々等しければ移動が起こり,その移動が正しければ報酬を,間違っていれば罰を受ける.どちらか一方でも等しくなければ移動は起こらず,エージェントは罰を受ける.移動可能な位置が複数ある場合「度数」が大きい方を選択して移動を行う.

エージェントは移動が行われた後,その移動が正しかったかどうかを報酬あるいは罰で知ることになる.新しいデータを得た結果が今回用いたメロディ作成のためのルール(順次進行中心に作る,跳躍進行の後は逆方向への順次進行を行う)を満たしているならば ID が持つ度数を報酬としてを与え,満たしていないならば負の値を罰として与える.こうすることによってエージェントは目的位置に辿り着くまでに得られる報酬を最大にしようとするので,学習を重ねるごとに負の値を得る方向を選択しなくなってゆき,また度数が大きい方向を選択するようになる.今回は割引率 0.9,学習率 0.1,負の値として-50 を与えた.割引率,学習率,負の値は適当に決定した.エージェントの移動例を示す.マップが図 11(A はエージェントを示す),度数分布が図 12 のようにあったとき図 13 のようにデータを比較し,現在のデータの 2 音めと候補となるデータの 1 音め,現在のデータの 3 音めと候補となるデータの 2 音めが其々等しければ,移動可能でありどちらか一方でも等しくなければ移動不可能である.移動可能な位置が複数ある場合度数が最も多きいものを選択する.図 13 の場合『364』,『127』が移動可能で,度数が大きい『364』の ID を持つ方向へ移動を行う.

-99	-99	-99
-99	580 (A)	490
-99	127	364

図 11． 移動例 (マップ)

ID	1 音め	2 音め	3 音め	度数
127	12	15	14	3
364	12	15	10	5
490	9	8	10	6
580	16	12	15	4

図 12． 移動例 (度数分布)

ID	1 音め	2 音め	3 音め	度数	
580	16	12	15	4	現在の データ
364	12	15	10	5	候補 1
127	12	15	14	3	候補 2
490	9	8	10	6	候補 3

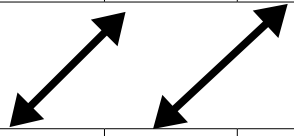

図 13． データの比較例

5.2.3. 音データの作成

学習が終了した後に音データの作成を行う.音データはエージェントが通った道のりから ID を取り出し,その ID に対応するデータを組み合わせて作成する.例を図 14 にエージェントの移動した道のりを示し,図 15 に対応する度数分布を示す.エージェントが図 14 において{1,2,3,4,5,6}の順で移動する.図 15 の度数分布を図 16 のように書き換える.図 16 の度数分布より数字を取り出し,図 17 のような数字の列を作る.この数字を{12:ド, 14:レ, 16:ミ, 17:ファ,}というように各音に対応させると図 18 のような音データが出来上がる.

-99	-99	-99	-99	-99
-99		1 (S)		-99
-99	2			-99
-99	3	4	5	-99
-99		6 (G)		-99
-99	-99	-99	-99	-99

図 14． 移動の軌跡

ID	1 音め	2 音め	3 音め	度数
1	12	14	16	10
2	14	16	17	9
3	16	17	12	6
4	17	12	12	5
5	12	12	12	11
6	12	12	17	8

図 15． 対応する度数分布

ID								
1	12	14	16					
2		14	16	17				
3			16	17	12			
4				17	12	12		
5					12	12	12	
6						12	12	17

図 16． 書き換えた度数分布

数字データ	12	14	16	17	12	12	12	17

図 17． 度数分布より得られる数字データ

音データ	ド	レ	ミ	ファ	ド	ド	ド	ファ

図 18． 生成された音データ

5.2.4. 音データの長さの決定

音データを作成した後累積分布とQ学習を用いて音データに当てはめる音符の長さを決定する.長さは長さの遷移を記録した度数分布の度数より累積分布を作成し,乱数を用いて累積分布よりデータを選択する.長さはこのデータの組み合わせが正しいかどうかによって報酬を受け取るか罰を受け取るかを決める.この際,音データの作成時に利用した方法と同様に,データが持つ「1 音めの音符」,「2 音めの音符」が其々前のデータの「2 音めの音符」,「3 音めの音符」と等しければエージェントは報酬を与えられ,どちらか一方でも異なっているば罰を受ける,というルールを用いる.ただし長さの場合には同じ長さの音符が 5 回以上続く様であれば,その場合似も罰を受けるようにしてある.ここで,与える報酬や罰,その他 Q 学習に用いたパラメータは音データ作成時と同じものを使用している.

5.2.5　累積分布について

度数分布表の度数より,各データが存在しうる確率を求める.累積分布はその数値の和を求めたもので,累積分布の 2 番目の確率値はと 1 番目の値と 2 番目の値の和になり,3 番目の確率値は 1 番めと,2 番めと,3 番目の値の和になる(表 4,表 5,図 19)[20].今回はこれを音符の度数分布を用いて作成する.

事象	確率
1	10%
2	0%
3	30%
4	10%
5	30%
6	15%
7	5%

表 4.　確率分布

事象	確率
1	10%
2	10%
3	40%
4	50%
5	80%
6	95%
7	100%

表 5.　累積分布

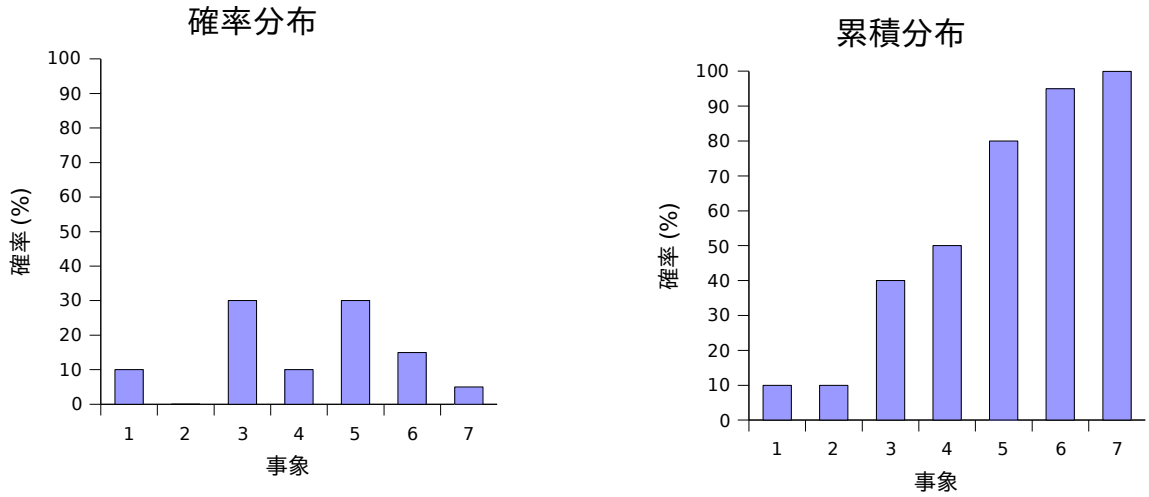

図 19　確率分布のグラフ (左) と累積分布のグラフ (右)

確率分布を累積分布にすることで,どんな乱数でも必ずどこかの間に入ることになる.例を図20,図21で示す.図20は確率分布の図であり,図21は累積分布の図である.分布から事象を選択するために乱数を用いたとし,その値が0.4(40%)であったとすると.図20では40%未満の事象しか存在しないため,どの事象も選択されないが図21では40%以上の事象が存在する.この時確率の低い事象の方から数えて,初めて確率40%以上の事象が選択される.この場合事象3が選択される.今回,音データの選択にはこの方法を用いている.

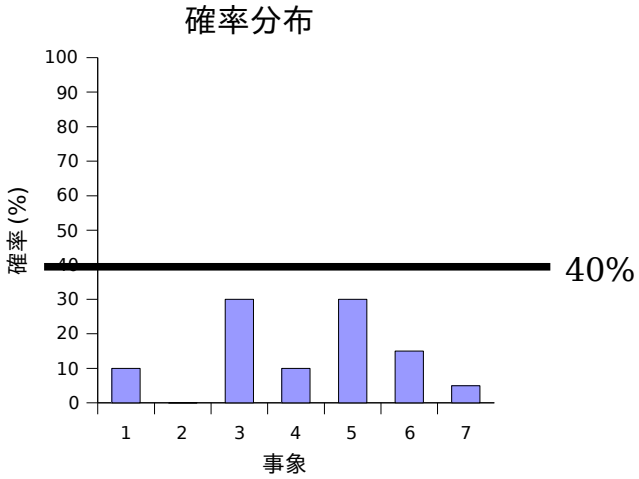

図20　確率分布での例

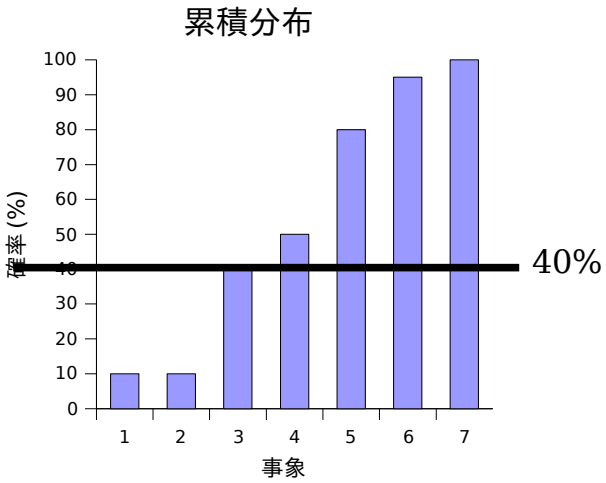

図21．　累積分布での例

6. シミュレーション

5 節で行った考察をもとにシミュレーション用のプログラムを作成する.今回のシミュレーションでは,度数分布の入力として童謡の楽譜をいくつか使用した[21].マップのサイズは見やすさを考慮して縦 16,横 15 とした.学習回数は音データの作成時,長さの決定時共に 400 回とした.これは,デバッグ時に音データ,長さ共にこの程度の学習回数で結果の収束が認められたことを理由とする.Q 学習におけるパラメータや,報酬などは 5 節に記述したものを使用する.シミュレーション用のプログラムには実行時に引数として,出力として得られる MML ファイルと,MIDI ファイルの名前をそれぞれ与える.このプログラムを10回起動し,その結果得られる MIDI ファイルを聞いてメロディと考えられるかどうかを調べ,MML ファイルから音符の遷移の状態を確認してルールが守られているかを調べる.

7. シミュレーションの結果

エージェントに初期位置から目的位置までの最適経路を探索させることは出来た.しかし,エージェントの移動ルールがシビアなようで,エージェントが目的位置に到達することよりもエージェントが全ての方向に移動することが出来なってしまったり,目的位置へ向かうルートが存在しなくなってしまうことの方が多くなり,学習が全く終了しなくなってしまう回数の方が多くなってしまった.これを回避するために,学習回数と学習ステップ数に制限を定めて制限回数以内に 1 度も目的位置に辿り着けない場合はゴールの位置を適宜変更することで学習を終了させ,最低限必要な音データの長さを定めて足りない場合にやり直しをさせることで,データ数が少なくなることを防ぐこととした.

出力された曲に関しては図 22 に示す例より長さのパターンが固定されてしまうと言う結果が出た.これは図 23 に示す音符決定に用いた度数分布に見られる[付点 8 分音符 , 16 分音符 , 付点 8 部音符]の組み合わせが最も度数が高く,それに続くことが出来るデータが[16 分音符 , 付点 8 分音符 , 16 分音符]しかないということが原因と思われる.Q 学習によって音符を決める際に度数を報酬として与えていることから,エージェントの目的である累積報酬を最大にすると言う意味で最適解が得られたと考えることはできる.ただ,メロディの自動作曲を目的とした場合,あるパターンしか出力できないと言うことは致命的であると思われる.よって,今回音符の決定に用いた方法は自動作曲を行うと言う目的にそぐわないものであることがわかった.改善を行うとすれば,音階を作成する際に幾つかのルールを用いたように音符を決める際にも幾つかのルールを用いる,エージェントに度数をそのまま報酬として与えるのではなく,何らかの形で手を修正を加えるなどが上げられる.

```
A8.g+16<c8.>A+16g8.g+16f8.f16
```

図 22　出力される MML の例

1音め	2音め	3音め	度数
4分音符	4分音符	4分音符	52
4分音符	4分音符	8分音符	11
4分音符	4分音符	符点8分	3
4分音符	8分音符	8分音符	26
4分音符	8分音符	16分音符	2
4分音符	符点8分	16分音符	18
8分音符	4分音符	8分音符	11
8分音符	8分音符	4分音符	23
8分音符	8分音符	8分音符	179
8分音符	8分音符	符点4分	12
8分音符	8分音符	符点8分	28
8分音符	16分音符	16分音符	10
8分音符	符点8分	16分音符	32
16分音符	8分音符	8分音符	34
16分音符	16分音符	8分音符	10
16分音符	符点8分	16分音符	92

1音め	2音め	3音め	度数
符点4分	4分音符	符点8分	9
符点8分	16分音符	4分音符	20
符点8分	16分音符	8分音符	35
符点8分	16分音符	符点4分	27
符点8分	16分音符	符点8分	93
休符	4分音符	4分音符	8
休符	4分音符	8分音符	5
休符	8分音符	8分音符	15
休符	符点4分	4分音符	5
休符	符点8分	16分音符	33
休符	休符	4分音符	14
休符	休符	8分音符	19
休符	休符	符点4分	6
休符	休符	符点8分	33

図 23　音符の度数分布

音の流れ方についても結果を見るために図 22 の MML を図 24 のように採譜を行う.図 24 より前の音と後の音の高さの差が 1 オクターブ以上高い音(長七度以上への進行)が存在しないことがわかる.また,図 25 のように曲線を描くと曲線とのずれが多少あるものの極端に変化が激しい箇所は見受けられない.このことから,度数分布を用いて不自然な音の遷移をなくすという当初の目的を果たすことが出来たと考えられる.

他は,図 24 に示したとおり,全体的に順次進行で構成され,跳躍進行の後に逆方向の順次進行も行なわれている.このように設定したルールを満足する結果が得られている.

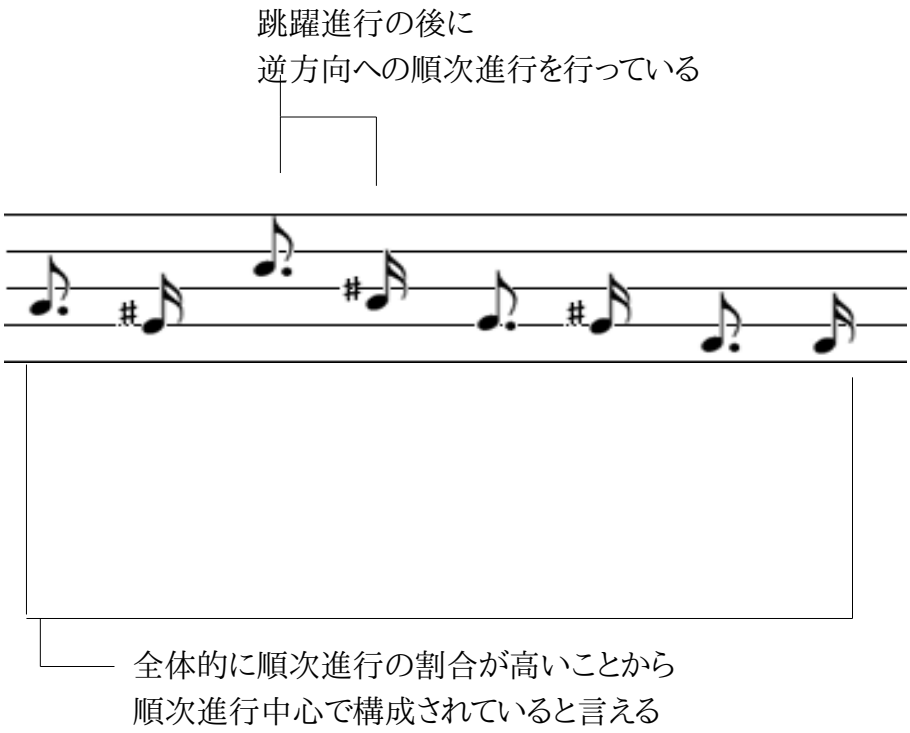

図 24　出力された結果を楽譜に採譜したもの

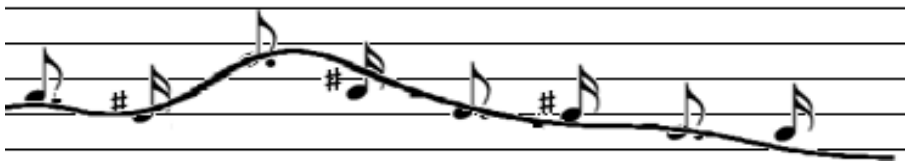

図 25　データの平滑化線

8. まとめ

今回自動作曲ツールを作成するにあたって,メロディのみの自動作曲を行った.その結果出力されたメロディは不自然な音階の遷移こそ取り除けたものの,自然な曲とは言えないものになってしまった.その理由の一つとして,音階の関係のみに固執してしまったことがあげられるだろう.このことより,音階のみに固執した方法では自然な流れを持つメロディを生成することは出来ないということが判った.

今回考慮しなかった要素のうち,リズムや拍子という要素を考慮することで自然なメロディを作成することが出来ると考えられる.リズムについては,今回別々に作成した音階と,その長さを同時に作成することで組み込むことが出来るのではないかと考えられる.具体的には,音階の関係とその長さの関係を同時に1つの度数分布に記録することで,もととなる曲のリズムを度数分布の中に記録することが出来ると思われる.その度数分布を用いることで多少はリズム感が改善され,自然なメロディに近づけるのではないかと思われる.ただ問題点として与えるMMLファイルの数を増やす必要があるということ,音階と音符の組み合わせが膨大になるため度数分布をメモリ内に保持するとしても,ファイルとして保存するとしてもそれに見合うだけの容量が必要になるという2点があげられる.

今後の課題としては,上記方法の実現と拍子の考慮によって自然なメロディを出力すること,ハーモニーなどその他の要素についても考慮してより音楽的な出力を得られるようにすること,さらに学習に必要な時間の短縮を行えるならばそれについても改善を行うこと,などが挙げられる.

9. 参考文献

[1] 勝田 哲司:月刊 DTM マガジン 10 月号 シリーズ 自動作曲研究所 乱数で音楽を作ろう!,p.56,(寺島情報企画,1999).
[2] 遺伝的アルゴリズムによる自動作曲 :http://www.nlab.sogo1.ynu.ac.jp/ynu/r_samples/no_10_3.html
[3] 音楽研究所:http://www.asahi-net.or.jp/~hb9t-ktd/music/musj.html
[4] 勝田 哲司:月刊 DTM マガジン 1 月号 シリーズ 自動作曲研究所 1/f ゆらぎでメロディを作ろう!,p.112,(寺島情報企画,2000).
[5] 高玉 圭樹 著:マルチエージェント学習-相互作用の謎に迫る-,(コロナ社,2005).
[6] 三上 貞芳, 皆川 雅章 著:強化学習,(森北出版,2004).
[7] 西田 豊明 著:人工知能の基礎,(丸善株式会社,2002).
[8] 強化学習とは:http://www.fe.dis.titech.ac.jp/~gen/edu/RL_intro.html
[9] 電気学会 GA･ニューロンを用いた学習とその応用調査専門委員会:学習とそのアルゴリズム,(森北出版,2005).
[10] メロディ :http://ja.wikipedia.org/wiki/%E3%83%A1%E3%83%AD%E3%83%87%E3%82%A3
[11] 水谷 友香, キッシ−岸田 著: Cd で覚えるやさしい楽譜の読み方 読み方･書き方から作曲･編曲までマスターしよう,(成美堂出版,2005).
[12] 関 和則 著:DTM のための全知識,(リットーミュージック,2000).
[13] デイブ･スチュワート 著,藤井 美保 訳:絶対わかる!曲作りのための音楽理論 必要最小限の理論知識だけで OK!,(リットーミュージック,2006).
[14] 御池 鮎樹 著:裏口からの MIDI 入門 理論不要の作曲道,(工学社,2004).
[15] 勝田 哲司:月刊 DTM マガジン 11 月号 シリーズ 自動作曲研究所 音階を組み込もう,p.64,(寺島情報企画,1999).
[16] 北川 祐 著:基本はドレミファソラシド 絶対わかる!コード理論,(リットーミュージック,2004).
[17] 意味音:http://imion.jp/index.asp?id=190&b_chkNum=4095
[18] 勝田 哲司:月刊 DTM マガジン 12 月号 シリーズ 自動作曲研究所 クラシックに学ぶメロディの作成技法,p.110,(寺島情報企画,2000).
[19]勝田 哲司:月刊 DTM マガジン 2 月号 シリーズ 自動作曲研究所 知識ベースでメロディを作ろう!,p.116(寺島情報企画,2000).
[20] Curtis Roads 著, 青柳 龍也 訳:コンピュータ音楽 歴史･テクノロジー･アート,(東京電機大学出版局,2005).
[21] 日本唱歌童謡集 唱歌･童謡･わらべ唄,(飯塚書店,1994).

10. ソースコード(Q 学習部分)

Q 学習部分のソースコードを添付資料として添付する.

```
#define Xsize 16                    //マップ x 方向のサイズ
#define Ysize 16                    //マップ y 方向のサイズ
#define Note_def 5                  //跳躍進行を表す数字
#define Roop 400                    //学習回数
#define CONTN 200                   //最大学習ステップ数
#define Act 8                       //行動の種類
#define LENGTH 8                    //音データの長さ
int Mus = 12;

struct MADATA
{
   int sx;                          //初期位置 x 座標
   int sy;                          //初期位置 y 座標
   int gx;                          //目的位置 x 座標
   int gy;                          //目的位置 y 座標
   int le_loop;                         //学習回数
   int fail;                        //ゴールに辿り着けなかった回数
   bool lean;                           //学習中 F 音データ作成 T
};

//進行方向決定
//引数 学習回数　現在位置 x 座標　現在位置 y 座標　評価関数 q
//返り値　進む方向(0 から 8)
int soft_max( int le_loop, int x, int y, double ***q_map )
{
   int move = 0;
   double T;
   double tmp_Q[Act];
   double sum = 0;
   srand( time(NULL) );

   //ボルツマン分布を用いて進む方向を選択する
   for ( int i = 0; i < Act; i++ )
   {
      tmp_Q[i] = exp(q_map[y][x][i]/T);
      sum += tmp_Q[i];
   }
   tmp_Q[0] /= sum;
   for ( int i = 1; i < Act; i++ )
```

```
    {
        tmp_Q[i] = tmp_Q[i-1] + tmp_Q[i]/sum;
    }

    double f = (double)rand()/RAND_MAX;
    for ( move = 0; move < Act; move++ )
    {
        if ( f < tmp_Q[ move ])
        {
            break;
        }
    }
    return move;
}

//Q 値の更新
//引数　現在位置 x 座標　現在位置 y 座標　次の位置 x 座標　次の位置 y 座標　進む方向　報酬　評価関数
void update_Q( int x, int y, int x2, int y2, int dir, double r, double ***q_map )
{
    double max_q2,q;
    double a = 0.1, mm = 0.9;
    int i;

    //Q 値の最大値を求める
    max_q2 = -10;
    q = q_map[ y ][ x ][ dir ];
    for (i = 0; i < Act; i++)
    {
        if ( q_map[ y2 ][ x2 ][ i ] > max_q2)
        {
            max_q2 = q_map[ y2 ][ x2 ][ i ];
        }
    }
    //Q 値を更新する
    q_map[ y ][ x ][ dir ] = q + a * ( r + mm * max_q2 - q);
}

//報酬の計算を行う
//引数　現在位置 x 座標　現在位置 y 座標　次の位置 x 座標　次の位置 y 座標　目的位置 x 座標　目的位置 y 座標　マップ　度数分布
//返り値 報酬
double reward( int x, int y, int x2, int y2, int Gx, int Gy, int **Note_map, DB
```

```
*maps)
{
   double r;
   //次の位置がゴールであるか
   if (( y2 == Gy ) && ( x2 == Gx ))
   {
      r = 100;
   }
   else
   {
      int old_one = Note_map[ y ][ x ];          //現在位置の ID
      int new_one = Note_map[ y2 ][ x2 ];    //次の位置の ID
      //次の位置が負の ID を持っているか
      if ( new_one < 0 )
      {
         r = -50;
      }
      else
      {
         //データが移動のルールを守っているか
         if ( (maps[ old_one ].dataS == maps[ new_one ].dataF) &&
               (maps[ old_one ].dataT == maps[ new_one ].dataS) )
         {
            //順次進行であるか
            if ( abs(maps[ old_one ].dataT - maps[ new_one ].dataT) < Note_def )
            {
               r = maps[new_one].dataP/10;
            }
            else
            {
               //長 7 度以上であるか
               if ( (abs(maps[ old_one ].dataT - maps[ new_one ].dataT ) < 11) )
               {
                  //増 4 度であるか
                  if ( (abs(maps[old_one].dataT - maps[new_one].dataT) == 6) )
                  {
                     r = maps[new_one].dataP/10-2;
                  }
                  else
                  {
                     r = maps[new_one].dataP/10-1;
                  }
```

```
            }
            else
            {
               //長 7 度の飛躍しんこうである
               r = -50;
            }

         }
      }
      else
      {
         r = -50;
      }
   }
  }
  return r;
}

//Q 学習移動部分
//引数　エージェント位置データ　音データ保存配列　マップ　評価関数　データベース
//戻り値　成功　1　　失敗　-1
int move( MADATA *MA, int *stream, int **Note_map, double ***q_map, DB
*maps )
{
  int my_x,my_y,my_x2,my_y2;  //現在位置ｘ　現在位置ｙ　次の位置ｘ　次の位置ｙ
  double r;                   //報酬
  int dir;                    //方向
  int count;                  //ステップ数
  int G = -1;                 //ゴールフラグ
  static int icont = 0;       //音データ数
  //状態の初期化　ここから
  my_x = MA->sx;
  my_y = 1;
  my_x2 = my_x;
  my_y2 = my_y;
  count = 0;
  r = 0;
  //状態の初期化　ここまで
  //ゴールに辿り着くまで繰り返し
  while (G < 0)
  {
    my_x2 = my_x;
    my_y2 = my_y;
```

```
    //進行方向の取得　ここから
    dir = soft_max( MA->le_loop, my_x, my_y, q_map );
    switch ( dir )
    {
       case 0:                              //左
          my_x2 --;
          break;
       case 1:                              //右
          my_x2 ++;
          break;
       case 2:                              //上
          my_y2 --;
          break;
       case 3:
          my_y2 ++;                         //下
          break;
       case 4:                          //左下
          my_x2 --;
          my_y2 ++;
          break;
       case 5:                          //左上
          my_x2 --;
          my_y2 --;
          break;
       case 6:                          //右下
          my_x2 ++;
          my_y2 ++;
          break;
       case 7:                          //右上
          my_x2 ++;
          my_y2 --;
          break;
       default :
          break;
    }
    //進行方向の取得　ここまで
    //報酬の取得
    r = reward( my_x, my_y, my_x2, my_y2, MA->gx, MA->gy, Note_map,
maps);
```

```
    //CONTN カウントいないにゴールにたどり着けない場合は失敗とみなす。
    if ( count > CONTN )
    {
      r = -100;
      //Q 値の更新
      update_Q( my_x, my_y, my_x2, my_y2, dir, r, q_map );

      stream[ icont ] = -999;
      icont = 0;
      MA->gx = my_x;
      MA->gy = my_y;
      return -1;
    }
    //Q 値の更新
    update_Q( my_x, my_y, my_x2, my_y2, dir, r, q_map );

     //報酬が-10 より大きければ
    if( r > -10 )
    {

        //音データ保存
        int T = maps[ Note_map[ my_y2 ][ my_x2 ] ].dataT;
        stream[ icont ] = T - 11;
        icont++;
        stream[ icont ] = -999;
      //状態の遷移
      my_x = my_x2;
      my_y = my_y2;
    }
    //ステップ数カウント
    count++;

    //ゴールに到達したならゴールフラグを立てる
    if ((my_x == MA->gx) && (my_y == MA->gy))
    {
      G = 0;
      icont = 0;
    }
  }
  return 1;
}
```