令和元年度　卒業研究

遺伝的アルゴリズムを利用した音楽生成の研究

函館工業高等専門学校
生産システム工学科　情報コース　5年
22番　髙橋宏太
指導教員　東海林　智也

目次

第１章　序論
　　第１節　英文アブストラクト
　　第２節　研究背景
　　第３節　研究目的
　　第４節　開発環境

第２章　関連技術
　　第１節　Mido
　　第２節　遺伝的アルゴリズム

第３章　プログラムの開発
　　第１節　プログラム構成
　　第２節　プログラム概要

第４章　結果

第５章　課題と考察

参考文献

付録　ソースコード

第１章　序論

第１節　英文アブストラクト

It is a purpose to generate music using genetic algorithms which is one of the algorithmic composition methods.

In this research, the program was developed using Python on Visual Studio Code.
Mido was used to generate MIDI files. There are several libraries that can make midi files in Python, but I chose Mido because it can easily handle many midi messages.

In conclude, we could make music, but it wasn't very good. There are mainly two causes, the first is a lack of evaluation functions. Second, there were so many random elements that we couldn't incorporate music theory much.

Key words: Algorithmic composition, Genetic algorithms, Evaluation function.

第 2 節　研究背景

　プログラムによる音楽の自動生成、いわゆる自動作曲は歴史が長く、これまで様々な手法を用いて作曲が行われてきた。近年は特に AI による自動作曲が注目を集め、多くのアプリケーションが開発されている。しかし AI を用いることは初学者にとって容易ではなく、学習を行う際にも多くの教師データやかなりのマシンパワーが要求される。

　そこで本研究では、アルゴリズム作曲法の一つである遺伝的アルゴリズムを用いた作曲法に注目した。遺伝的アルゴリズムを用いることで、入力データや強力なマシンを必要とせずに作曲ができると考えたからである。

第3節　研究目的

　本研究の目的は遺伝的アルゴリズムを用いて音楽知識のない人でも作曲を行えるようなプログラムを開発することである。そのため、どのような場面でも使える、多様な音楽を生成できるプログラムの開発を目的とした。

第４節　開発環境

研究室 PC　　OS：Windows10
　　　　　　　CPU：Intel Core i5 760 @2.80GHz
　　　　　　　RAM：4.00GB

開発言語　Python

開発環境　Visual Studio Code

使用ライブラリ　NumPy
　　　　　　　　Mido

第2章　関連技術

第1節　Mido

　Mido は Python で MIDI ファイルの作成を行うために使用したライブラリである。Python 上で MIDI ファイルを扱うためのライブラリは他にも存在するが、それらと比べて多くの MIDI メッセージを簡潔に表現できるため、本研究ではこのライブラリを用いて開発を行った。

　このプログラムは音の再生と停止を行うもので、note で音の高さ、channel で出力チャンネル、velocity で音量、time で音の長さを指定している（図1）。

```
track.append(Message('note_on',note=60, channel = 0, velocity=127, time=0))
track.append(Message('note_off',note=60, channel = 0, time=480))
```

図1　音の再生と停止を行うプログラムの例

第２節　遺伝的アルゴリズム

遺伝的アルゴリズム(Genetic Algorithm)とは生物が世代を重ねるごとに進化を行う仕組みをもとに考案された最適解探索を行うためのアルゴリズムである。データで表現された「遺伝子」の集合である「個体」に対し評価、選択、交叉、突然変異などの操作により最適解を求める。

初めに第一世代にあたる初期個体群を生成し、それらの個体に対し「評価」を行い個体それぞれに点数をつける。次に点数を基に次の世代の個体の親となる個体を決定する「選択」を行う。そして、親となる個体の遺伝子を組み合わせて新しい個体を生成する「交叉」や、遺伝子の一部がランダムに変化する「突然変異」を行い、次の世代の個体群を生成(世代交代)する。

そして、生成された個体群に対しても再び評価、選択、交叉、突然変異の操作を行い、この一連の操作をあらかじめ指定した世代数だけ繰り返し操作を終了する。

これを簡単に図で表したものが次の図である（図 2）。

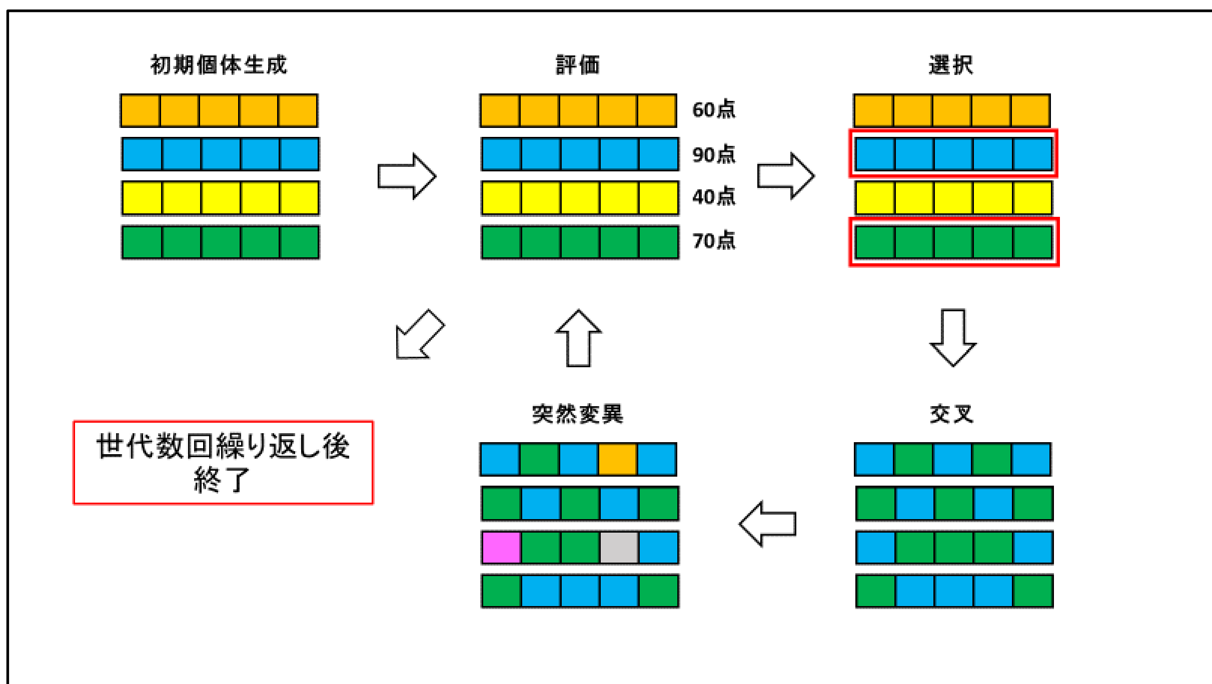

図 2　遺伝的アルゴリズムの流れ

第 3 章　プログラムの開発

第 1 節　プログラム構成

プログラムの流れを解説する前に、プログラムの大まかな構成について説明する。

最初の世代の個体群である初期個体は、全てランダムで決定する。

次の世代の個体を生成する動作、いわゆる「世代交代」は 10 回行い、最後の世代の最も点数の高かった個体と伴奏、ピアノ演奏を一つの MIDI ファイルにまとめたものを最終的な出力とする。

生成される曲は、C ダイアトニックコードの構成音のみ使用するものとする。(簡潔に言い換えると、ピアノの真ん中のドの音から一オクターブ高いドの音までのド・レ・ミ・ファ・ソ・ラ・シ・ドの八つの音のみ使用する、ということである。)

個体は配列で表現し、各個体には遺伝子が 16×小節数だけ存在するものとする。これを表したのが以下の図 3 である。各遺伝子の第一要素には 0 か 1 が格納されており、0 であったら休符を、1 であったら音符をそれぞれ表現している。第二要素には音の高さを表す MIDI ノート番号が格納されており、第一要素が 0 の場合には 0 が格納される。また、遺伝子 1 つは 16 分音符(休符)分の長さを表しているため、同じ遺伝子が 2 つ、4 つ連続で並んだ場合にはそれぞれ 8 分音符(休符)、4 分音符(休符)を表現している。そのため、本研究で生成される曲に同じ音の連符は出現しない。

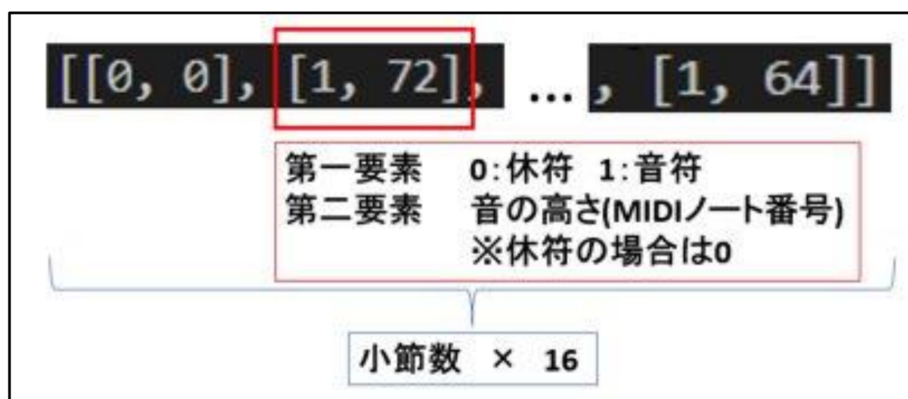

図 3　個体・遺伝子の概要

第2節　プログラム概要

ここからはプログラムの流れについて説明する。

初めに初期個体を生成する。音符か休符かをランダムで決定した後、音符であったら音の高さも使用音の範囲内でランダムに決定する。本研究では、音符が生成される確率を70%、休符が生成される確率を30%とした。

次に評価関数に従って、生成された個体に点数付け(評価)を行う。評価項目は主に①16分音符、休符の数、②付点音符の数、③長すぎる音符、休符の数、④8分音符、四分音符の数、⑤連続する二つの音符の音の高さの差の5つである。①～③については、メロディーに多く含まれると不自然になってしまうと考えたため、一定数以上含まれる場合には減点する処理を加えた。④についてはこれらの音符はメロディーに含まれるほうが曲にメリハリを感じられるため、個体の中にある場合は加点する処理を加えた。⑤は差が大きすぎると曲が慌ただしく、不自然になってしまうので減点する処理を加えた。個体の評価はこのように、主に音符や休符に注目して行った。

その後、各個体の点数をもとに、次の世代の親となる個体二つを決定する「選択」を行う。親①は最も点数の高いもの、親②は確率関数をもとに決定する。確率関数にはソフトマックス関数を採用しており、ソフトマックス関数を用いることで親②には高確率で二番目に点数が高い個体、ごくまれにその他の個体が選ばれる仕様となっている。

親が決定した後は、次の世代の個体を生成する操作を行う。次の世代の個体のうち二つは親①と親②をそのままコピーしたものとした。このようにすることで、次の世代の個体の最高点が前の世代の最高点より低くなってしまうことを防いでいる。その他の個体は交叉と突然変異を用いて作成しており、交叉は遺伝子をランダムで二か所選択し、親①と親②の遺伝子をその間の範囲で入れ替える二点交叉、突然変異はどちらかの親の遺伝子を一か所ランダムで反転させ、それを新しい個体とする動作をそれぞれ行っている。

以上の操作を世代数回繰り返した後は、ドラム演奏と伴奏を生成する。ドラム演奏はエイトビート、伴奏はあらかじめ決定している曲のコードに従って生成した。

最後に、最後の世代の一番点数の高い個体とドラム演奏、伴奏を一つのMIDIファイルにまとめて出力する。

第 4 章　結果

初めに、各世代の個体数を 100、500、1000 にしてプログラムを 5 回実行した際の、最終世代の個体の最高点をまとめたものを以下の図 4 に示す。表より、各世代の個体数が多ければ多いほど最高点の平均点が高くなることがわかる。しかし、同じ個体数でも最高点にはかなりばらつきが見られ、場合によっては個体数が少なくても個体数が多い場合より高得点の個体が生成されることもあった。

	1	2	3	4	5	平均
100	20	21	6	15	32	18.8
500	32	33	43	46	44	39.6
1000	58	66	55	40	48	53.4

図 4 個体数を変化させたときの最高点

次に、各世代の個体数を 1000 にしたときに生成された曲の一例を示す。以下の図 5 は、Domino というアプリケーションで生成された曲を再生したときの様子である。

生成された曲を聴くと、ドラム演奏と伴奏はかなり綺麗に聞こえるが、メロディーはあまり良いものとは言えなかった。初期個体に比べ 8 分音符や 4 分音符といった長めの音符は多くみられるが、音の変化がかなり乱雑に感じられた。そのため、ドラム演奏と伴奏ありきで何とか曲が成り立っている印象を受ける。

また何度かプログラムを実行した結果、類似した曲が生成されることがあった。

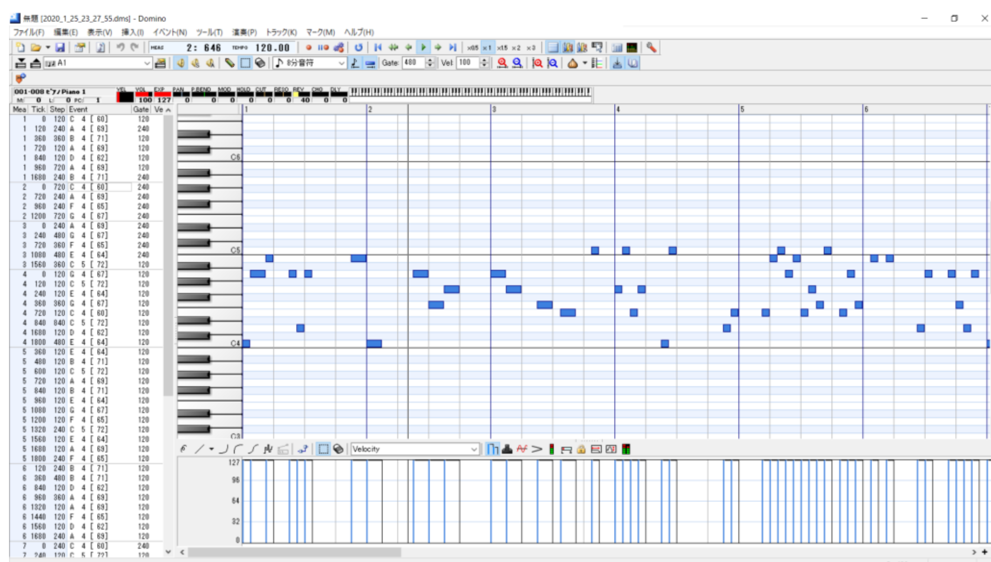

図 5 生成された曲の一部

第 5 章　考察と課題

結果より、現時点では実用的なプログラムではないといえる。そのため、今後より実用的なプログラムにするための考察と今後の課題を述べる。

まずあまりメロディーが美しくない理由として、評価関数の不足が挙げられる。本研究で作成したプログラムの評価関数は、できるだけ主観的な評価項目を入れるべきではないと考えたため、音符や休符に関する評価項目のみ取り入れている。そのため、音の流れやリズムは一切考慮していないためあまり美しい曲が生成されなかったのだと考えられる。よって、音やリズムに関する評価項目を追加することによってこの課題を解決できると考えられる。

次に類似した曲が多数生成されるという問題についてである。この問題は、曲を生成する際のコードや、使用する音に多くの制限を加えていることによって起こっていると考えられるので、今後はコードのパターンや使用できる音の種類を増やして生成される曲に多様性を持たせせたい。

また点数が安定せずに、出力結果が運任せになってしまっているという問題もある。これは初期個体の生成段階や交叉を行う際の入れ替えの位置、突然変異する場所などをすべてランダムで決定してしまっていることが原因であると考えられる。そのため、これらの操作を完全ランダムで行うのではなく、一定のルールを設けて行うことで安定した出力結果が得られるのではないかと考える。

今後の課題はユーザが欲しい曲の雰囲気を決定し、それに従って曲を生成する機能を追加することである。この機能により、よりプログラムの実用性が上がるほか、「生成される曲に多様性がない」という問題の解決にもつながると考える。

本研究を通じて、遺伝的アルゴリズムでは評価関数の設定が出力結果に大きな影響を及ぼすことが確認できた。世代交代を行うごとに点数は改善されていくが、評価関数の設定によってはそれが必ずしも曲の改善につながるとは限らないのである。しかし評価関数をうまく設定することができれば、自分の思い通りの曲が生成できるようになることが遺伝的アルゴリズムを用いた音楽生成の強みでもある。そのため、評価関数の改良を続けることでより実用的なプログラムに近づいていくと考えられる。

参考文献

[1]Python で遺伝的アルゴリズム
https://qiita.com/KeisukeToyota/items/0f527a72270430017d8d

[2]Mido 公式ドキュメント
https://mido.readthedocs.io/en/latest/index.html

[3]遺伝的アルゴリズム
https://www.sist.ac.jp/~kanakubo/research/evolutionary_computing/genetic_algorithms.html

[4]遺伝的アルゴリズムを用いたコード進行を考慮した自動作曲
https://ipsj.ixsq.nii.ac.jp/ej/?action=repository_action_common_download&item_id=104720&item_no=1&attribute_id=1&file_no=1

付録

ソースコード

```
import numpy as np
import random
import copy
import mido
import os
import datetime
from mido import Message, MidiFile, MidiTrack, MetaMessage

# 各パラメータを決定
gene_length = 16  # 遺伝子長
individual_length = 1000  # 個体数
generation = 10  # 世代数
mutate_rate = 0.1  # 突然変異の確率

measure_length = 8  #小節数

mid = [[MidiFile(type=1) for i in range(individual_length)]
       for g in range(generation + 1)]
track = [[MidiTrack() for i in range(individual_length)]
         for g in range(generation + 1)]
acc = [[MidiTrack() for i in range(individual_length)]
       for g in range(generation + 1)]
drums = [[MidiTrack() for i in range(individual_length)]
         for g in range(generation + 1)]

for g in range(generation + 1):
    for i in range(individual_length):
        mid[g][i].tracks.append(track[g][i])
        track[g][i].append(MetaMessage('set_tempo', tempo=mido.bpm2tempo(120)))
        track[g][i].append(
            Message('program_change', program=12, channel=0, time=0))

        mid[g][i].tracks.append(acc[g][i])
        acc[g][i].append(MetaMessage('set_tempo', tempo=mido.bpm2tempo(120)))
```

```
        acc[g][i].append(
            Message('program_change', program=0, channel=1, time=0))

        mid[g][i].tracks.append(drums[g][i])
        drums[g][i].append(MetaMessage('set_tempo', tempo=mido.bpm2tempo(120)))

#ディレクトリ作成
for g in range(generation + 1):
    if not os.path.exists('gene' + str(g) + ''):
        os.makedirs('gene' + str(g) + '')

if not os.path.exists('output'):
    os.makedirs('output')

#コード生成
C = [60, 64, 67]
Dm = [62, 65, 69]
Em = [64, 67, 71]
F = [65, 69, 72]
G = [67, 71, 74]
Am = [69, 72, 76]

tonic = [C, Em, Am]
dominant = [G]
sub_dominant = [F, Dm]

code_list = []
for m in range(int(measure_length / 4)):
    code_list.append(random.choice(tonic))
    code_list.append(random.choice(sub_dominant))
    code_list.append(random.choice(dominant))
    code_list.append(random.choice(tonic))

tone_list = [60, 62, 64, 65, 67, 69, 71, 72]


def note_func():  #音符か休符かを決定
    return np.random.choice([0, 1], p=[0.3, 0.7])
```

```
def tone_func(tonelist):  #音を決定
    return random.choice(tonelist)


def make_midi(population, g):  #MIDI ファイルを作成
    for i in range(individual_length):
        t_start = 0
        t_end = 120
        for j in range(measure_length):
            k = 0
            while k < gene_length:
                if population[i][j * 16 + k][0] == 1:
                    while k < gene_length - 1 and population[i][j * 16 +
                                                                k][1] == population[i][j
                                                                                       *
                                                                                       16
                                                                                       +
                                                                                       k
                                                                                       +
                                                                                       1][1
]:
                        t_end += 120
                        k += 1
                    track[g][i].append(
                        Message(
                            'note_on',
                            note=population[i][j * 16 + k][1],
                            channel=0,
                            velocity=127,
                            time=t_start))
                    track[g][i].append(
                        Message(
                            'note_off',
                            note=population[i][j * 16 + k][1],
                            channel=0,
                            time=t_end))
                    t_start = 0
                    t_end = 120
                else:
```

```
                t_start += 120
            k += 1


def make_sub(g):  #伴奏とドラム演奏を作成
    for i in range(individual_length):
        for j in range(measure_length):
            acc[g][i].append(
                Message(
                    'note_on',
                    note=code_list[j][0],
                    velocity=100,
                    channel=1,
                    time=0))
            acc[g][i].append(
                Message(
                    'note_on',
                    note=code_list[j][1],
                    velocity=100,
                    channel=1,
                    time=0))
            acc[g][i].append(
                Message(
                    'note_on',
                    note=code_list[j][2],
                    velocity=100,
                    channel=1,
                    time=0))

            acc[g][i].append(
                Message(
                    'note_off', note=code_list[j][0], channel=1, time=1920))
            acc[g][i].append(
                Message('note_off', note=code_list[j][1], channel=1, time=0))
            acc[g][i].append(
                Message('note_off', note=code_list[j][2], channel=1, time=0))

            for m in range(gene_length):
                if m % 2 == 0:  #ハイハット
```

```
                    drums[g][i].append(
                        Message(
                            'note_on', note=42, channel=9, velocity=80,
                            time=0))

                if m == 4 or m == 12:  #スネア
                    drums[g][i].append(
                        Message(
                            'note_on', note=40, channel=9, velocity=80,
                            time=0))

                if m % 8 == 0:  #バスドラ
                    drums[g][i].append(
                        Message(
                            'note_on', note=36, channel=9, velocity=80,
                            time=0))

                drums[g][i].append(
                    Message('note_off', channel=9, note=42, time=120))
                drums[g][i].append(
                    Message('note_off', channel=9, note=40, time=0))
                drums[g][i].append(
                    Message('note_off', channel=9, note=36, time=0))


def get_population():  #初期個体の生成
    population = [[[0 for k in range(2)]
                   for j in range(gene_length * measure_length)]
                  for i in range(individual_length)]

    for i in range(individual_length):
        for j in range(measure_length):
            for k in range(gene_length):
                population[i][j * 16 + k][0] = note_func()
                if population[i][j * 16 + k][0] == 1:
                    population[i][j * 16 + k][1] = tone_func(tone_list)

    make_midi(population, 0)
    return population
```

```python
def fitness(pop):  #評価関数
    score = [100 for i in range(individual_length)]
    for i in range(individual_length):
        rests = 0  #休符の数
        con_rests = 1  #連続する休符の数
        con_notes = 1  #連続する音符の数
        rests_long = 1  #長すぎる休符の数
        interval_w = 0  #離れすぎている音符の数
        interval_n = 0  #近い音符の数
        rests_16 = 0  #16 分休符の数
        note_16 = 0  #16 分音符の数
        note_8 = 0  #8 分音符の数
        note_4 = 0  #4 分音符の数
        note_long = 0  #4 分(より長い)音符の数
        dot_notes = 0  #付点音符の数

        for j in range(gene_length * measure_length):
            if pop[i][j][0] == 0:
                rests += 1
                if pop[i][j - 1][0] == 0:
                    con_rests += 1
                else:
                    if con_rests > 3:
                        rests_long += 1
                    con_rests = 1
                if j < 62 and pop[i][j - 1][0] == 1 and pop[i][j + 1][0] == 1:
                    rests_16 += 1

            if pop[i][j][0] == 1:
                if pop[i][j - 1][0] == 1 and pop[i][j][1] == pop[i][j - 1][1]:
                    con_notes += 1
                else:
                    if con_notes == 2:
                        note_8 += 1
                    elif con_notes == 4:
                        note_4 += 1
                    elif con_notes > 4:
```

```
                    note_long += 1
                if con_notes == 3 or con_notes == 6 or con_notes == 12:
                    dot_notes += 1
                con_notes = 1

            if j < 63:
                if pop[i][j + 1][0] == 1 and pop[i][j][0] == 1:
                    if abs(pop[i][j + 1][0] - pop[i][j][0]) >= 4:
                        interval_w += 1
                    if abs(pop[i][j + 1][0] - pop[i][j][0]) <= 2:
                        interval_n += 1

            if j < 62:
                if pop[i][j][0] == 1 and pop[i][j -
                                                1][1] != pop[i][j][1] and pop[i][j
                                                                                  +
                                                                                  1][1] != p
op[i][j][1]:
                    note_16 += 1

        if rests > 20:
            score[i] -= rests

        if dot_notes > 8:
            score[i] -= 2 * dot_notes

        score[i] -= (
            4 * rests_long + interval_w + note_long + rests_16 + 4 * note_16)
        score[i] += note_8 + 2 * note_4 + interval_n

    return score


def softmax(score):  #ソフトマックス関数
    c = np.max(score)
    exp_a = np.exp(score - c)
    sum_exp_a = np.sum(exp_a)
    y = exp_a / sum_exp_a
    return y
```

```python
def evaluate(s):  #エリート個体の要素番号をリターン
    elite = []
    elite.append(s.index(max(s)))
    s[s.index(max(s))] = 0
    pro = softmax(s)
    e = np.random.choice(
        range(individual_length), size=1, replace=False, p=pro)
    elite.append(e[0])
    return elite


def mutate(parent):  #突然変異
    r = random.randint(1, gene_length * measure_length - 1)
    child = copy.deepcopy(parent)
    child[r][0] = ~child[r][0]
    if child[r][0] == 1:
        child[r][1] = tone_func(tone_list)
    return child


def two_point_crossover(parent1, parent2):  #交叉
    r1 = random.randint(0, gene_length - 1)
    r2 = random.randint(r1, gene_length - 1)
    child = copy.deepcopy(parent1)
    child[r1:r2] = parent2[r1:r2]
    return child


def main():
    population = get_population()  #初期個体生成
    score = fitness(population)  #適応度取得
    print('Generation:0')
    print('Max:' + str(max(score)) + ' NO.' + str(score.index(max(score))))
    print('Min:' + str(min(score)) + ' NO.' + str(score.index(min(score))))
    print('-------------------------')
    p = population

    for g in range(generation):
```

```
    elite_num = evaluate(score)  #選択
    elites = [[[0 for k in range(2)]
               for j in range(gene_length * measure_length)]
              for i in range(2)]
    elites[0] = p[elite_num[0]]
    elites[1] = p[elite_num[1]]

    p = copy.deepcopy(elites)

    while len(p) < individual_length:
        if random.random() < mutate_rate:  #交叉
            m = random.randint(0, len(elites) - 1)
            child = mutate(elites[m])
        else:  #突然変異
            m1 = random.randint(0, len(elites) - 1)
            m2 = random.randint(0, len(elites) - 1)
            child = two_point_crossover(elites[m1], elites[m2])
        p.append(child)

    score = fitness(p)  #評価
    print('Generation:' + str(g + 1))
    print('Max:' + str(max(score)) + ' NO.' + str(score.index(max(score))))
    print('Min:' + str(min(score)) + ' NO.' + str(score.index(min(score))))
    print('--------------------------')

    make_midi(p, g + 1)

#出力ファイル生成
time = datetime.datetime.now()
y = time.year
mo = time.month
d = time.day
h = time.hour
mi = time.minute
s = time.second
make_sub(g + 1)
mid[g + 1][score.index(
```

```
        max(score))].save('output¥¥' + str(y) + '_' + str(mo) + '_' + str(d) +
                          '_' + str(h) + '_' + str(mi) + '_' + str(s) + '.mid')


if __name__ == '__main__':
    main()
```