2023春 音乐与数学 遗传作曲 作业报告

一、概述

在众多的音乐创新尝试中，机器作曲的研究与应用自提出以来便在质疑与追捧中曲折前进——机器能否实现人类所引以为傲的创作。在认可其中的创新型的基础上，研究小组决定利用遗传算法探究机器作曲的可能性。其大致过程为：我们在若干歌曲中选取了长为4小节的片段，进行随机变异后生成了初始种群，在python上建立了遗传算法。每一次迭代对片段进行随机变换，随机杂交，从生成的种群中根据我们建立的适应度函数挑选出数值较高的进行下一次迭代。最后我们生成了若干长为4小节的音新的乐片段。

二、过程

1、片段选取

我们从流行乐、民谣、古典等多种音乐中类中截取了近70个音乐片段，每个音乐片段都满足最短时值为八分音符的要求，同时我们标注了每个音乐片段的调号，以便后续变换处理。最终，我们以片段间的相似性和正确性为条件筛选，从中挑选出52个音乐片段，包括了C,A,D,G,F等多种调号，作为我们遗传算法的初始种群。

2、初始化

对于每一个初始片段，我们将输入的音符编码以八分音符的时值转换，0代表休止，20代表延音，并用20到108的整数代表A0-C8的音高，将其转化为一个长度为32的向量进行储存。同时我们输入音符时，记录了每一个片段对应歌曲的调号，并在初始化时将所有片段平移至1=C。

3、随机变异

我们采用的随机化操作包括随机移调，逐小节的倒影变换，逆行变换，和对单个音节的随机改变（随机左移或右移一至二个半音），以及对两个片段的随机杂交。每次迭代时每个片段都以一定概率进行上述操作，确保了迭代若干次后有较为分散的结果分布。

4、适应函数

适应函数可以说是遗传算法的灵魂, 也可以说，适应函数决定了遗传算法的表现。我们不难发现，在智能作曲这个与个人感觉高度相关的领域, 对于一首歌曲是否好听的评判尤其难以程序化、数据化。所以如何给出一个行之有效而又不失高效的适应函数，将关系到整个算法体系的成败。在基于遗传算法的智能作曲模型中，我们需要手动对于音乐规则进行提取，即设置一些条件和规则去告诉适应函数一首歌曲是否“好听”。首先，我们考虑引入一段参考乐曲，该参考乐曲选取公认的“好听”的歌曲，接下来将通过遗传算法得到的乐曲片段与参考乐曲进行对比。第一步，我们分别计算参考乐曲与结果乐曲片段所有音符的音高的均值，计算其均值的差值，得到分数part_1，第二步，我们考虑参考乐曲与结果乐曲音高浮动的差距，计算参考乐曲方差与结果乐曲方差的差值，得到分数part_2，代码如下：

```
part_1 = abs(np.mean(reference)-np.mean(x))
#part_1表示与参考数组的均值差距
part_2 = abs(np.var(reference)-np.var(x))
#part_2表示与参考数组的方差差距
```

其次，我们暂时认为在一段乐曲片段中音高越趋于稳定，在一定范围内波动，这样的片段相对更“好听”，这里我们设置区间[65, 75]，然后统计结果乐曲片段中超出这一范围的音符个数，得到数量n并取对数，得到分数part_3; 最后，我们考虑和弦测试和跳跃音测试。第一步，利用和弦测试函数，用于评估和弦的质量：我们用evaluate_chord函数接收一个和弦序列 chord，并根据特定的音程规则来评估和弦的质量。在模型中我们采用了一个简单的评估逻辑，即判断和弦中的音程是否与目标音程（如大三度、纯四度、纯五度）相符，通过计算不符合音程的数量，将和弦的质量得分定义为不符合音程的比例，从而得到分数part_4。代码如下：

```
def evaluate_chord(chord):
    # chord是一个音符序列，假设为一个整数列表，表示和弦的音高
                # 示例评估逻辑：判断和弦是否符合某种特定的音程规则
    intervals = [chord[i+1] - chord[i] for i in range(len(chord)-1)]  # 计算音程
    # 定义一个目标音程列表，例如：大三度、纯四度、纯五度
    target_intervals = [4, 5, 7]
    # 计算和弦中与目标音程不符合的音程数量
    incorrect_intervals = sum([1 for interval in intervals if interval not in
target_intervals]
    # 根据不符合音程的数量计算和弦的质量得分
    chord_quality = 1 - (incorrect_intervals / len(intervals))
    return chord_quality
```

第二步，我们引入跳跃音过渡（即音高跳跃），来评估音符序列中的跳跃音过渡的质量。我们利用evaluate_leap_transitions函数接收一个音符序列 melody，并计算其中跳跃音过渡的数量。通过检查相邻音符之间的音高差是否大于预设的音程阈值（例如大于大三度），来判断是否存在跳跃音过渡。最后，根据跳跃音过渡的数量占总过渡数的比例，计算跳跃音过渡的质量得分，从而得到分数part_5。代码如下：

```
def evaluate_leap_transitions(melody):
    leap_transitions = 0  # 记录跳跃音过渡的数量
    for i in range(len(melody) - 2):
        # 计算相邻音符之间的音高差
        interval1 = abs(melody[i+1] - melody[i])
        interval2 = abs(melody[i+2] - melody[i+1])
        # 判断是否存在跳跃音过渡，例如大于一个预设的音程阈值
        leap_threshold = 4  # 音程阈值，例如4表示大三度
        if interval1 > leap_threshold and interval2 > leap_threshold:
            leap_transitions += 1
    leap_transition_fitness = 1 - (leap_transitions / (len(melody) - 2))  # 计算跳
跃音过渡的质量得分
    return leap_transition_fitness
```

在得到五个关键分数后，根据文献中适应性函数的定义，我们设置分数对应的权重，经过不断地调参并评估乐曲的悦耳程度，我们最终得到以下权重：`initial_parameter= [-0.1,-0.1,1,200,10]`

5、后期处理

对于一些不和谐的片段，我们将其所有音符向我们选定的“和谐”音符上靠拢。比如我们将所有音符转换为最近的五声音阶，或者转化为最近的C大调音阶等。

三、结果评估

四、总结