**音乐与数学实验报告——Machine Composing**

第四题-第十四组

组长：赵陆森；组员：潘聿阳、梁科、闫志昂、吉喆、于宏平

**一、项目概述**

本小组使用遗传算法进行旋律创作与机器作曲。本项目全部代码、参考文献、生成音乐示例以及本报告均已同步至<https://github.com/ForestOnTheLand/MachineComposing>。

**（一）小组成员及分工**

* 赵陆森（组长）：搭建项目代码遗传算法框架，参与设计适应性函数。
* 潘聿阳：设计与试验适应性函数。
* 梁科：设计与试验适应性函数。
* 闫志昂：设计与试验适应性函数。
* 吉喆：试验适应性函数，参与小组实验报告的撰写。
* 于宏平：参与试验适应性函数，完成小组项目实验报告的撰写。

**（二）项目框架概述**

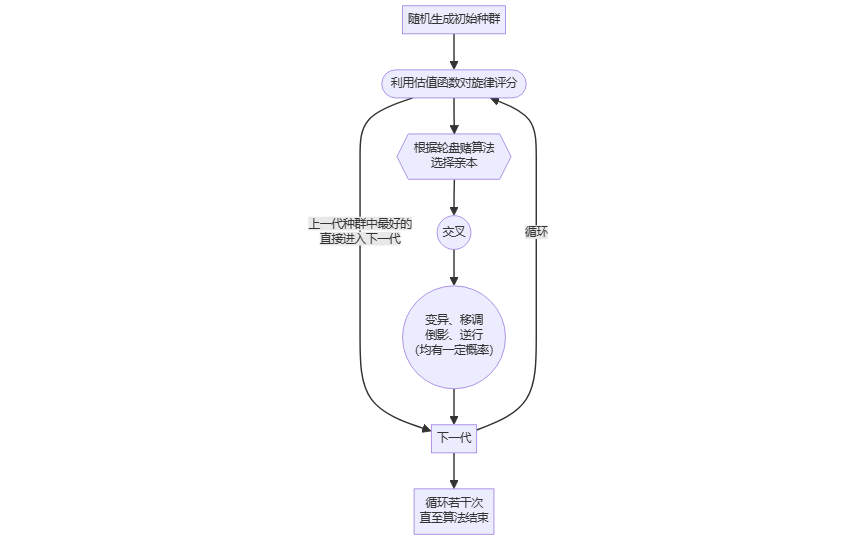
本项目编程部分全部采用Python完成。项目中引入mido库进行音乐导出（midi格式），pygame库进行音乐播放；实现了遗传算法（包括交叉、变异、移调、倒影、逆行等操作），采用从已有音乐导入、随机生成两种方式产生初始种群，编写和试验了多种估值函数用以指导旋律进化的方向。

在源代码目录下(./src)，各文件夹实现了机器作曲的不同部分。melody文件夹下实现了音符与旋律的抽象类型，实现了音乐导入与导出，并且包括多个乐曲片段；algorithm文件夹下实现了遗传算法的部分，包括交叉变异等操作与各类估值函数；util文件夹下包含一些与项目主体无关的工具。可以运行main.py文件保存并播放音乐，可以更改其中估值函数权值与初始种群等参数以试验效果。

具体细节不做赘述[[1]](#footnote-0)，但我们将对其中的遗传算法（尤其是估值函数）在后文作详细讨论。

**二、遗传算法的实现**

遗传算法的过程与课上所讲类似。首先向遗传算法中输入等长的若干旋律作为初始种群，然后进行若干代进化：每一次进化时，一方面从上一代种群中选择评分最高的一个旋律直接进入下一代；另一方面采用轮盘赌算法（即按被选择概率正比于评分的方式）选择两个亲本进行交叉产生子代，同时子代有一定概率发生突变。突变的形式包括：随机选择音符单点突变，随机选择片段移调、倒影、逆行。

 在algorithm文件夹下，genetic.py包含了遗传算法框架，fitness.py包括了多种估值函数，init.py包含了随机初始种群生成器，operation.py包含了交叉(one\_point\_cross,two\_points\_cross)，突变(one\_point\_mutate)，移调(transpose)，倒影(inverse)，逆行(retrograde)操作。下面会对估值函数做详细的讨论。

**三、对估值函数的探索**

在一段时间探索后，我们编写的多个估值函数大概可以划分为两种：一种是“分数”(xxx\_score)，对每一个音乐返回一个数值（0到1之间），值越大表示音乐越好，用于指导旋律的进化方向；另一种是“惩罚”(xxx\_penalty)，对于大部分音乐都是无惩罚，只有少部分具有不希望出现的特征的音乐有惩罚。

以下的示例中，为了保证结果的可复现性，均采用固定随机种子(3407)，并且只截取了有代表性的结果，而其中的结论是通过长期试验得来的。如无说明，初始种群均为随机生成。下面是我们对估值函数的探索历程，以及对估值函数效果的评价。

**（一）添加interval\_score函数**

在最初的版本中，我们只是对生成的旋律中相邻音符的音程进行判断，鼓励生成协和音程。以下是此时尝试生成的一段旋律，尚算不上好的音乐：



**（二）添加tonality\_score函数**

接下来我们添加了一个函数，用于判断音符是否尽可能多地处于同一大/小调内（可以指定某一具体的调式，使之尽可能生成处于该调式中的音符，如A major）。以下是此时尝试生成的一段旋律，比起开始时有了一些改进，但仍算不上好的音乐：



**（三）添加rhythm\_score函数**

我们继续添加了一个函数，用于判断相邻小节的节奏型的相似程度，尽量使相似程度更高。以下是此时尝试生成的一段旋律，在某种程度上有了小小的进步，但同时一个更大问题也显现出来，就是休止符的泛滥：



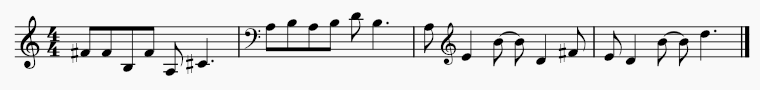
**（四）添加了四个惩罚函数**

针对可能出现的性质差的音乐，我们加入了惩罚机制，避免这些音乐的产生。为了不让音符过于集中或过于分散，添加了density\_penalty函数，对音符过疏、过密的情况扣分；为了不让音乐戛然而止，添加了stop\_penalty函数，避免了这种情况；为了不让休止符干扰生成的音乐，添加了rest\_penalty函数，用于阻止任何休止符的产生；为了不出现过多的连续八分音符，添加了consecutive\_penalty函数，避免了这种情况。以下是此时尝试生成的一段旋律，可以听出已经有了明显的改进，但仍有所欠缺：



**（五）添加了stable\_score、boundary\_score函数**

在现代旋律创作的"稳定-不稳定-稳定"理论中，将大调式中的主和弦的三个音视为稳定音，二、六音视为较不稳定音，四、七音视为极不稳定音。我们采用了这个理论，要求生成的旋律以稳定音开始，也以稳定音结束，对应的函数是boundary\_score。此外还要求旋律的变化满足如下的4种情况：稳定-较不稳定-稳定、稳定-极不稳定-稳定、稳定-较不稳定-极不稳定-稳定、稳定-极不稳定-较不稳定-稳定，对应stable\_score函数。以下是此时尝试生成的一段旋律，有了巨大的进步：



**（六）添加了三个惩罚函数**

尽管第五阶段大部分生成旋律的效果已经相当不错，但仍然有概率产生一些“难听”的音乐。由于以上评分函数的限制，有时会生成过于单调的音乐，甚至有时只有两种音高；有时会出现音符过于跳跃的情形，导致音乐的割裂感。

我们仍觉得有进步的空间，遂添加了三个新的惩罚函数：避免旋律中的音符音高相差过多的range\_penalty函数、避免旋律中的音符种类过少的variety\_penalty函数、避免出现孤立的（与其两端的音符音高相差过大的）音符的lonely\_penalty函数。以下是此时尝试生成的一段旋律，听感与第五阶段相差不大。但是多次试验表明，产生“难听”音乐的概率大大降低：



**（七）以乐曲片段构建初始种群**

以上这些尝试都是基于某个特定的随机数种子（在这里是3407）生成的初始种群。但是我们也可以基于一些固定的初始种群进行生成，下图中的例子使用了10段已有的旋律，进行了三次尝试，生成的三段旋律如下：







尽管是基于同样的初始种群进行的进化，由于没有固定随机数种子，遗传算法的不确定性导致了以上三个不同的进化结果。主观地说，这些旋律显然不能与生成它们的初始种群相提并论，但作为随机生成的旋律而言，已经是能让人接受的程度，不会使听者对这些旋律产生厌恶感或怪异感，尤其是第三段旋律，比第一、二段更让人听之愉悦。

**（八）评分函数对于效率的影响**

在进化500次的前提下，分别对第一到六阶段测定进化所花的时间，测定的结果为0.16s、0.32s、0.34s、0.57s、1.16s、1.17s，可以看出第五阶段所做的优化费时最多，但相应的其优化效果也最好，其余的阶段的优化效果则几乎与多花费的时间成正相关。

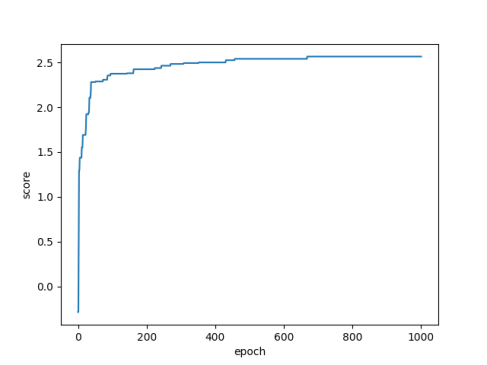


图1:乐曲分数与进化轮数的关系

另一方面，我们还研究了进化次数对生成的旋律在上述评分机制下所获分数的影响。固定随机种子为3407时，我们绘制了如图所示的图像。特别地，在进化10、50、100、200、500、1000次时，对应的分数分别为1.43、2.28、2.37、2.42、2.54、2.56。据此我们可以发现，前200次进化的效果较为明显，而在200次以后，进化对生成的旋律几乎没有改进。说明200次进化已经能近乎达到在这个评分机制下的最优状态。

**（九）评分函数小结**

评分函数汇总如下表：

|  |  |  |  |
| --- | --- | --- | --- |
| 类别 | | 函数 | 作用 |
| 旋律 | 评分 | interval\_score | 鼓励生成协和音程 |
| tonality\_score | 鼓励乐曲符合调性 |
| boundary\_score | 鼓励旋律以稳定音开始、以稳定音结束 |
| stable\_score | 鼓励旋律符合“稳定-不稳定-稳定”理论 |
| 惩罚 | range\_penalty | 避免最高、最低音差距过大 |
| variety\_penalty | 避免音符种类数过少 |
| lonely\_penalty | 避免某个音与相邻两个音差距大 |
| 节奏 | 评分 | rhythm\_score | 鼓励相邻小节的节奏型相近 |
| 惩罚 | density\_penalty | 避免起拍处密度过高过低 |
| stop\_penalty | 避免音乐戛然而止 |
| rest\_penalty | 避免不合理休止符 |
| consecutive\_penalty | 避免过多连续的起拍处 |

**四、结论**

在遗传算法的设计与完善过程中，我们不仅感受到了音乐与数学、计算机之间的学科交叉为音乐创作带来的创新性发展，也在估值函数的设计中对课上所讲的概念有了进一步的认识。在设计估值函数的过程中，我们从节奏、旋律两个角度设计音乐好坏的标准，这也就是音乐的三个要素中的两个（不涉及和声）；我们也对课上所讲的音乐中的旋律进行（稳定-不稳定-稳定）、移调/逆行/倒影变换、交叉变异操作等有了更多的了解。

**参考文献与工具**

在项目中，我们使用了mido库进行音乐导出（midi格式），pygame库进行音乐播放；在报告撰写过程中，我们使用了Musescore软件进行音频到五线谱的转换，利用matplotlib.plt库进行图像绘制。参考文献如下：

1. Dragan Matic, A genetic algorithm for composing music, *Yugoslav Journal of Operations Research,* **20** (2010), 157–177.
2. Towsey, M., Brown, A., Wright, S., and Diederich, J., “Towards Melodic Extension Using Genetic Algorithms”, *Educational Technology & Society*, 4 (2) 2001.
3. Marques, M., Oliveira, V., Vieira, S., and Rosa, A.C., “Music composition using genetic evolutionary algorithms”, *Proceedings of the IEEE Conference on Evolutionary Computation 2000*. IEEE Press, New York, NY, 2000.
4. Wiggins, G., Papadopoulos, G., Phon-amnuaisuk, S., and Tuson, A., “Evolutionary Methods for Musical Composition”, *Proc. of the CASYS98 Workshop on Anticipation, Music&Cognition*, 1998.
5. 刘艳梅.遗传算法在旋律创作中的应用研究[D].河南师范大学[2023-12-23].DOI:CNKI:CDMD:2.1014.381189.

1. 可以参考Github仓库中的README.md，其中有更详细的对代码框架的阐述。 [↑](#footnote-ref-0)