Plat使用手册

目录

[各文件及其说明 1](#_Toc68701045)

[1. 问题类 2](#_Toc68701046)

[2. 算法类 4](#_Toc68701047)

[3. 个体类 6](#_Toc68701048)

[4、运行方法 7](#_Toc68701049)

# 各文件及其说明

1. Algorithms：用于存放算法
2. Problems：用于存放问题
3. config.py：用于传递程序运行过程中的相关参数
4. main.py：算法的主函数，用于运行程序

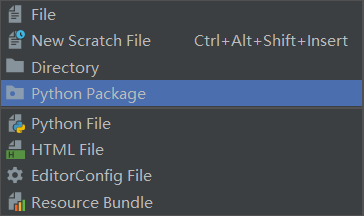
# 1. 问题类

定义问题时需要继承PROBLEM类，并且将问题保存在Problems文件夹中。问题类包含的属性以及函数如下所示：

|  |  |
| --- | --- |
| 属性 | |
| N | 种群规模 |
| D | 决策变量维度 |
| M | 目标个数 |
| encoding | 编码方式：’real‘，’permutation‘ |
| parameter | 问题相关参数 |
| FE | 当前评估次数 |
| maxFEs | 最大评估次数 |
| 函数 | |
| Initialization | 初始化初代种群 |
| CalDec | 修补决策变量 |
| CalObj | 计算适应度值 |

具体操作如下：

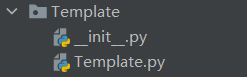
首先，新建Python Package



并将Package的名字修改成问题的名字，此时，Package中将自动生成一个\_\_init\_\_.py。



接着，在已生成的Package中新建python file，文件名为问题名。



每个问题需要继承PROBLEM类，并重新定义 CalDec，例如SOP\_F1.py：

1. from Problems.PROBLEM import PROBLEM
2. import numpy as np
3. class SOP\_F1(PROBLEM):
4. def \_\_init\_\_(self):
5. super().\_\_init\_\_()
6. def CalObj(self, PopDec):
7. PopObj = np.sum(PopDec \*\* 2, 1)
8. return PopObj

各行代码功能如下：

第1行：表示继承PROBLEM类

第2行：导入numpy类

第3行：定义问题SOP\_F1并继承PROBLEM

第4行：类的构造函数

第5行：继承父类\_\_init\_\_()函数

第6行：定义CalObj，用于计算适应度值

第7行：计算适应度值

第8行：返回适应度值

使用时，用户只需要在CalObj中定义问题的具体形式即可。

# 2. 算法类

定义算法时均需要继承ALGORITHM类，并将算法保存在Algorithms文件夹中，创建方法与问题类相同。算法类包含的属性以及函数如下所示：

|  |  |
| --- | --- |
| 属性 | |
| Problem | 算法当前所求解的问题 |
| parameter | 算法所包含的参数 |
| 函数 | |
| NotTerminated | 算法当前是否已经终止 |
| Optimization | 算法优化过程 |

每一个算法都需要继承ALGORITHM类，并重新定义Optimization函数，例如PSO.py：

1. from Algorithms.ALGORITHM import ALGORITHM
2. class PSO(ALGORITHM):
3. def \_\_init\_\_(self, \*parameter):
4. super().\_\_init\_\_()
5. if len(parameter) == 0:
6. self.c1 = 2
7. self.c2 = 2
8. elif len(parameter) == 1:
9. self.c1 = parameter[0]
10. self.c2 = 2
11. else:
12. self.c1 = parameter[0]
13. self.c2 = parameter[1]
14. def Optimization(self):
15. Population = self.Problem.Initialization()
16. while ALGORITHM().NotTerminated():
17. pass
18. return Population

各行代码功能如下：

第1行：表示继承ALGORITHM类

第2行：定义算法PSO并继承ALGORITHM

第3行：类的构造函数

第4-13行：算法的初始化

第14行：定义优化过程

第15行：初始化种群

第16行：判断算法是否终止

第17行：算法优化过程

第18行：返回整个种群

对于一个算法，其Optimization函数至少需要包含以下三个内容：

1. 初始化：

Population = self.Problem.Initialization()

2. 判断是否终止：

while ALGORITHM().NotTerminated():

3. 返回整个种群：

return Population

使用时用户只需要在Optimization中编写相应的代码，最后得到整个种群即可。

# 3. 个体类

SOLUTION对象用于表示一个个体，一组SOLUTION对象可以表示整个种群。个体类的属性以及函数如下所示：

|  |  |
| --- | --- |
| 属性 | |
| Problem | 解所求解的问题 |
| Population | 解所组成的种群 |
| PopDec | 解的决策变量 |
| PopObj | 解的适应度值 |
| decs | 种群中所有的决策变量 |
| objs | 种群中所有的适应度值 |

该类不需要用户定义，可以按照如下方式进行使用：

1. SOLUTION(np.ramdom.random((100,5)))

以上代码表示生成一个种群规模为100\*5的种群。

# 4、运行方法

文件main.py中定义了平台的运行方法，如下所示：、

1. import config
2. from Problems.Mesh\_Simplification.Mesh\_Simplification import Mesh\_Simplification
3. from Algorithms.PSO.PSO import PSO
4. if \_\_name\_\_ == '\_\_main\_\_':
5. # General parameter settings
6. config.M = 1
7. config.N = 10
8. config.D = 10
9. config.encoding = 'permutation'
10. Config.maxFEs = 50
11. # Problem initialization
12. Problem = Mesh\_Simplification('test1.obj', 0.9)
13. config.problem = Problem
14. # Algorithm initialization
15. PSO = PSO()
16. # Start the optimization process
17. Population = PSO.Optimization()

各行代码功能如下：

第1行：表示导入config

第2行：导入问题

第3行：导入算法

第7-11行：通用设置

第14-15行：初始化问题

第18行：初始化算法

第20行：优化过程

用户使用步骤如下：

1. 定义相应的问题及算法，并放入指定的文件夹下

2. 将第1、2、3行进行导入

3. 初始化通用设置

4. 初始化问题并将问题存放到config中

5. 初始化算法

6. 进行优化