高级搜索

1.遗传算法解决TSP问题

为了方便批改作业, 我们统一用类 `GeneticAlgTSP` 来编写遗传算法的各个模块, 并分析算法性能. 该类需包含以下方法:

- 构造函数 `__init__()`, 输入为TSP数据集文件名 `filename`, 数据类型 `str`. 例如 `"dj38.tsp"` 是Djibouti的38个城市坐标数据文件; `"ch71009.tsp"` 是China的71009个城市坐标数据文件. 我们需要在构造函数中读取该文件中的数据, 存储到类成员 `self.cities` 中(数据类型自定, 建议存储为 `numpy` 数组). 同时在构造函数中初始化种群, 存储到类成员 `self.population` 中(数据类型自定).
- 求解方法 `iterate()`, 输入为算法迭代的轮数 `num_iterations`, 数据类型 `int`. 该方法是基于当前种群 `self.population` 进行迭代(不是从头开始), 返回迭代后种群中的一个较优解, 数据类型 `list`, 格式为1-n个城市编号的排列. 例如, 对于n=5的TSP问题, 迭代后返回的较优解形如 `[1,3,4,5,2]`, 表示当前较好的游览城市次序为1-3-4-5-2-1.

可以在类中编写其他方法以方便编写并分析遗传算法的性能. 请在代码注释或实验报告中说明每个方法/模块的功能.

提示

1. 数据集来源于网站National Traveling Salesman Problems (uwaterloo.ca). 可浏览该网站参考相关国家的TSP问题的解. 可以自选1-2个数据集来测试算法性能, 并在实验报告中说明.
2. TSP问题上遗传算法的具体实现(解的表示, 染色体交叉操作等)不一定局限于课件上的方式, 也许存在比课件效果更好的具体实现方法.
3. 由于遗传算法是基于随机搜索的算法, 只运行一次算法的结果并不能反映算法的性能. 为了更好地分析遗传算法的性能, 应该以不同的初始随机种子或用不同的参数(例如种群数量, 变异概率等)多次运行算法, 这些需要在实验报告中呈现.
4. 最后提交的代码只需包含性能最好的实现方法和参数设置.
5. 对于规模较大的TSP问题, 遗传算法可能需要运行几分钟甚至几个小时的时间才能得到一个比较好的结果. 因此建议先用城市数较小的数据集测试算法正确与否, 再用城市数较大的数据集来评估算法性能.
6. 本次作业可以使用 `numpy` 库以及python标准库. 有余力的同学可用 `matplotlib` 库对遗传算法的结果进行可视化处理与分析, 并在实验报告中呈现.