城市认知智能挑战赛算法(DGT Arithmetic)介绍

队伍名: 零玖叁伍

成员:程凯越,谭泽龙

1 简介

在第一届"城市认知智能挑战赛"——智能交通治理专题中,我们提出了DGT算法(**D**ata generation, **G**enetic algorithms, manual **T**uning),通过三个阶段进行全局优化,达到"交通治堵"的目的。我们的工作主要有四个贡献:

- 采用了遗传算法进行全局优化;
- 构建了一个规模为2930的数据样本集;
- 制作了一个绘制地图和车辆热度图的工具;
- 提出了一种基于地图的手动调优方式。

我们在第一届"城市认知智能挑战赛"中,以699.09的分数排名第三。

2 主算法:遗传算法

在遗传算法中,我们主要分为四个步骤,分别是:

- 1. 排序筛选原始数据;
- 2. 遗传样本集基因融合产生子代;
- 3. 对子代进行评估测试;
- 4. 将子代及评估结果并入数据库;
- 5. 回到1, 再次排序筛选, 进行迭代。

2.1 排序筛选

排序筛选在现实中是一个选择优秀种群的过程,即所谓的"种马"、"种猪",在自然选择的基础上加入人工干预,从而尽快繁育出优秀的个体,由优秀的个体形成优秀种群。

条件:

- 每个 access.txt 和 result ——对应;
- result/time.txt 的第一行对应的是评估分数。

目标:

• 生成遗传算法的亲代

方法:

- 将已有的所有样本按分数排序;
- 取分数较小(排名较高)的40个样本;
- 将提取出的样本作为亲代。

2.2 基因融合

基因融合在现实中是一个配种的过程,即由双亲生成子代的过程。为了有足够的样本量,我们让每对双亲产生4个子代,并且让子代进行一定的变异,防止陷入局部最优,而且越到遗传后期,变异率就要越大。

条件:

● 亲代数据

目标:

• 子代数据

方法:

- 将亲代的40个数据点进行两次两两匹配,生成 $2 \times 20 = 40$ 个数据对;
- 每一个数据对 (p_A,P_B) 进行如下操作, 简称"基因结合":
 - 取 P A 的前一半和 P B 的后一半结合, 生成 S 1;
 - 取 P_A 的后一半和 P_B 的前一半结合, 生成 S_2;
 - 取 P A 的前一半和 P B 的前一半结合, 生成 S 3;
 - 取 P A 的后一半和 P B 的后一半结合, 生成 S 4 。
- 对生成的所有子代做如下操作,简称"基因变异":
 - 。 随机修改1%的数据;
 - 。 随机删除5%的数据;
 - 随机添加2%的数据。

2.3 测试评估

为了确定由上一步产生的子代怎么样,我们要用一个统一的标准来评估它们,比赛提供了虚拟引擎可供使用,所以 我们用虚拟引擎对每个子代进行评估。为了提高效率,我们同时运行了16个仿真,平均每个仿真耗时5分钟,评估 一次迭代后的所有子代公160个耗时不到1小时。

条件:

• 子代数据

目标:

• 子代评估结果

方法:

• 利用虚拟引擎对每个子代进行评估

2.4 数据入库

将上一步的所有子代和评估结果按照一定的格式放入数据库中,现在的数据格式是:

```
resultlist1/ //第一轮迭代的评估结果
    — out 1/
    — out_2/
    ├ . . .
  - epoch2/ // 第二轮迭代
    ─ access1.txt // 第二轮迭代的第一个样本
    ├─ access2.txt // 第二轮迭代的第二个样本
   - resultlist2/ //第二轮迭代的评估结果
    ___ out_1/
    - out_2/
    garaw/ // 保存了每一代的双亲
 —— epoch1p/ // 第一轮结束经过筛选后的双亲
 ├── epoch2p/ // 第二轮结束经过筛选后的双亲
 <u></u> ....
- raw_data/ // 摄像头控制的相关类
 ├─ 2021_09_28_19_35_11/ // 加入数据库的时间
    ─ accesslist/ // 规则文件
       - access1.txt
       - access2.txt
     — resultlist/ // 评估结果
       result1/
       - result2/
```

条件:

• 子代数据及评估结果

目标:

• 将子代的数据及评估结果并入数据库

方法:

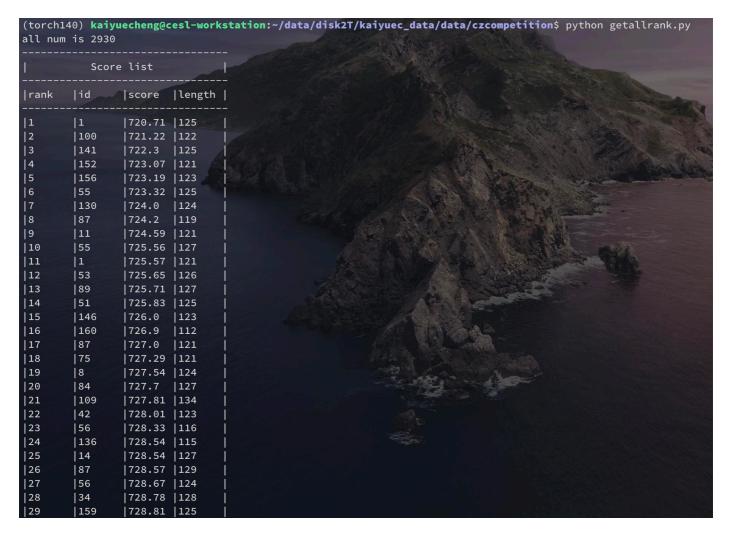
● 直接复制

2.5 迭代

重复2.1~2.4的步骤。

3 遗传算法数据

通过16轮迭代, 我们共得到了2930个样本, 并且将评分优化到了720.71分。

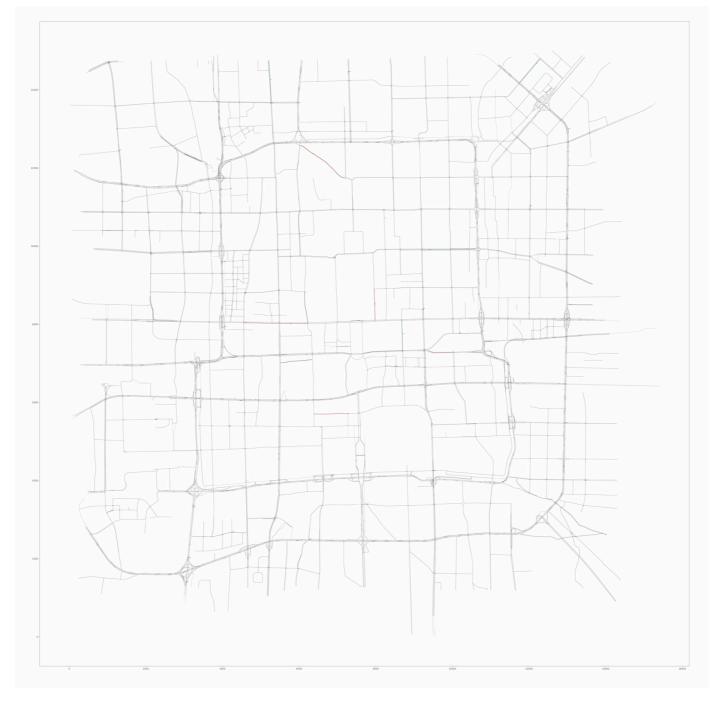


4 辅助工具: 绘制地图及热点图

因为我们采用的是"低配版"的遗传算法,几乎没有做任何避免陷入局部最优的工作,所以它还是不可避免地在达到 720后优化变缓。于是我们采用纯人工手动调优的方式,开发了一套能够绘制地图和热点图的工具,模仿"交通指挥中心"进行手动限行,对系统进行优化。

主要分为个步骤:

- 1. 获得限行规则 access.txt 和车辆轨迹数据 traj.txt;
- 2. 获得地图数据 map.json;
- 3. 解析以上文件, 绘制地图;
- 4. 在已有地图的基础上标出限行道路和拥堵路段。



4.1 绘制地图

- 1. 从 map. json 中读取一个车道;
- 2. 获取车道中心点的坐标序列;
- 3. 对坐标进行初始化,即减去地图范围的(x_min, y_min);
- 4. 采用matplotlib的根据坐标序列绘制折线的方式绘制地图;
- 5. 读取下一个车道,回2。

4.2 标注限行道路

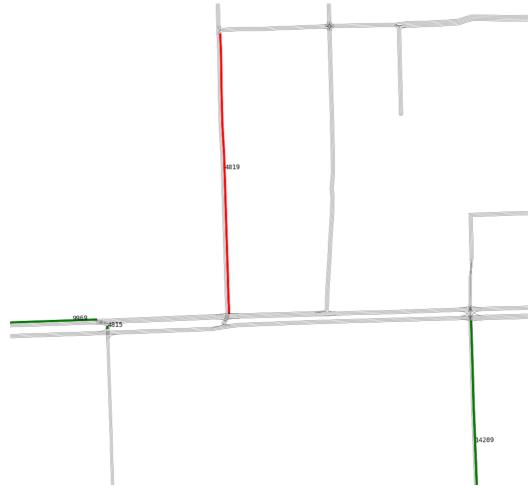
在4.1的基础上,读取车道之后进行判断,如果车道在 access.txt 中,则绘制为绿色,并用 plt.text 添加文字,否则绘制为黑色。

4.3 标注拥堵路段

- 1. 读取 traj.txt 文件中的一行数据;
- 2. 统计 2,5,8,...,3n+2,... 的字串(车道号),统计出每个时刻在每个车道上车辆的数量;
- 3. 按照规则筛选出拥堵路段,当前使用的是 车辆数>50 和 >100 的规则,还可以改为 车辆数/车道长度 、 通过时间/车道长度 等其他规则;
- 4. 将拥堵路段存储在文件或内存中;
- 5. 在4.1绘制地图时,读取车道之后进行判断,如果车道拥堵,则标注为红色,并用 plt.text 添加文字,否则 绘制为黑色。

5 手动调优

我们在对地图可视化的基础上进行手动调优,以下图为例



- 1. 发现拥堵车道4819;
- 2. 判断该车道为由南向北行驶, 南端和北端分别为两个岔路口;
- 3. 寻找分流路线, 由南向北的车辆可以通过4819东侧的道路进行分流行驶;
- 4. 限制4819北端右转车道和南端转入车道;
- 5. 在 map.json 中读取这两个车道的车道号;
- 6. 将其加入 access.txt, 重新仿真、绘制、手动调优。

最终,我们用此方法从720.71分优化到了699.09分,即最终分数。