第六届湖南省研究生数学建模竞赛

题 目 基于概率分布遗传算法与仿真的交通灯配时策略研究

摘　　要：

随着我国经济的迅速发展以及人口总数的不断增加，汽车的数量也呈现不断增长的趋势，但汽车给人们带来便利的同时也带来了如交通拥堵、交通事故频发等问题。交通拥堵问题逐渐成为不少城市的心头之患，交通信号灯是解决交通堵塞问题的关键一环。交通拥堵现象通常发生在车流量较大的路口，如果路口处交通灯的调度方案不合理，导致无效绿灯时间，使得路口有效通行时间变短，则拥堵现象可能会更加严重。因此本文根据实际的道路交通信息，设计了基于概率分布遗传算法与仿真的交通灯配时算法。

首先，我们构建了有限状态机的车辆道路交通仿真系统，可以仿真计算任意调度方案下的车辆总得分。汽车在道路中行驶一共有 3 种状态，分别为车辆行驶在道路上、车辆行驶到非行程终点的道路终点和车辆行驶到行程终点，五种输入动作，分别为车辆未到达路口、车辆没通过路口、车辆到达非终点路口、车辆到达终点路口和车辆通过路口。根据这三种状态，五种输入动作，设计状态转移函数，实现了车辆在道路中行驶，在路口行驶或者等待的仿真模型。

其次，我们设计了基于概率分布的遗传算法，完成对调度方案的优化求解。通过仿真系统计算适应度函数，设计基因的初始化、选择、交叉、变异算子。为了加速计算，在基因初始化与变异的过程中，不采用均匀分布策略，而采用概率分布的策略，使得每个路口的信号灯变化周期较小的基因出现的频率更高。对比概率分布遗传算法与传统遗传算法的运行时间，概率分布遗传算法的收敛的运行时间比传统遗传算法的运行时间少了 7 倍，大大的提高了算法的快速响应能力。

再次，我们设计了小规模、中规模和大规模的交通道路网络结构。为了更加贴近真实道路，我们通过随机网络算法得到在节点平均度与平均路径长度与真实道路相近的不同规模的交通道路连通情况。根据交通道路网络结构，计算交通道路仿真系统的运行时间，并据此判断算法的可行性。根据计算，在大规模交通道路中，计算 1 次所需时间为 30s 左右，不适合用于遗传算法的计算中。而中小规模均可以较为快速的计算得到优化调度方案。

最后，我们分析道路故障对车辆路径的影响，采用基于 Dijkstra 的车辆路径规划算法，并应用概率分布遗传算法，求解得到新的优化调度方案，可以及时解决道路故障造成的拥堵现象，进而提高交通系统的鲁棒性。

关键词：遗传算法，有限状态机，仿真系统，概率分布，Dijkstra

1 问题重述

1.1 问题背景

近年来，随着人民生活水平的不断提高，我国车辆销量总体呈现大幅度上升趋势。车辆数目的急剧增长，导致了交通拥堵、交通事故、尾气排放和能耗增多等一系列问题。其中，交通拥堵已经成为了当下面临的严重问题。尤其在城市多向交通路口，更容易发生拥堵现象。随着物联网技术的快速发展。交通信号灯可以通过远程控制交通信号灯的调度方案。当街道存在交通拥堵或者即将发生拥堵现象时，可以通过及时更换交通信号灯的调度方案，来及时减少街道拥堵时间。在城市路网交通灯连通可控的情况下，在发生拥堵时，往往需要城市交通系统在较短的时间内给出相对合理的交通信号灯调度方案。

基于此，本文设计了遗传算法与有限自动机仿真模型相结合的算法，可以在较短时间内给出相对合理的调度方案。

1.2 问题提出

城市交通系统的相关信息包含全市的路网结构、车辆的行程计划和车辆行驶速度等信息，在题目中，城市交通系统的相关信息都是不变且已知的。

（1） 全市路网结构

数据文件 “streets.txt” 给出了全市的路网结构，包含了每条道路的“道路名称”、“起点路口”、“终点路口”和“通行时间”。其中，“通行时间”表示畅通状态下，车辆从该道路起点到终点所需要的时间，因为车辆行驶速度是固定的，故该通行时间也代表了道路的长度。总共有 8000 个路口，63968 条道路。

（2） 车辆行程计划

数据文件 “cars.txt”给出了所有车辆的行程路径信息，包含了每辆车“车辆ID”、“该车行程路径中的道路总数”、“该车的行驶路径”。总共有 200 辆车。

（3） 其他信息

调度的总时间 D 为 858 秒。车辆到达终点的基本得分F为 250，若某辆车j，在T_j时刻到达终点，则该车得分为

$$M_j = \begin{cases} F + (D - T_j), T_j \le D \\ 0, \qquad\qquad T_j > 0 \end{cases}$$

在已知城市交通系统的相关信息的情况下，调度方案的定义是：每个路口的道路交通灯在一个周期内亮起绿灯的顺序以及持续时长。其中，1）每个路口的交通信号灯群都是独立控制的；2）路口的交通信号灯群中的每个红绿灯都在固定周期内变换信号； 3）每个交通灯在一个周期内至多亮起绿灯一次；4）所有交通灯的初始状态为红灯。

基于上述条件，本文具体解决以下两个问题：

问题 1：研究交通灯全局优化调控策略，分析算法适用的城市路网规模，并针对附录 2 中指定的数据，给出具体的调度方案，按照附录 3 的要求将结果保存在 文件 “result1.txt” 中。

问题 2：在 30s 时，道路 ebf-ffef 的起点处发生设施故障，需要立即维修，导致该道路无法继续通行，在问题 1 的基础上，讨论对交通信号灯调度方案的影响，设计局部调整调度方案的方法，给出调整后的调度方案，按照附录 3 的格式

要求将结果保存在“result2.txt”文件中。同时按照附录 2“cars.txt”的格式，将调整路线的车辆完整路径信息（从起点到终点）保存在“newcars.txt”中，并将车辆路径信息结果写在正文中。

2 模型假设与符号说明

2.1 基本假设

根据附录 1 所列的条件可知，本文对真实的交通系统进行了简化，将本题中给出的假设条件分为道路、路口、车辆、发生故障四个方面，具体如下：

（1）道路假设条件

1. 每条道路由起点路口与终点路口组成，为单向通行道路。两个路口之间方向相反的两条道路对应现实中的双向同行。
2. 任意两个路口之间的同向道路至多有一条。
3. 每条道路的终点处都对应设置一个交通灯。交通信号灯仅有红灯和绿灯两种状态。当交通灯为红色时，表示“停止”，即到达道路终点处的车辆停在道路上，不能穿过路口；当交通灯为绿色时，表示“通行”，即到达道路终点的车辆可以穿过路口，开往以该路口为起点的其他路口。
4. 道路交通灯为红灯时，所有到达该道路终点的车辆排队等待绿灯。

（2）路口假设条件

1. 每个路口至少是一条道路的终点，另外一条道路的起点。
2. 路口的交通灯是指所有**以该路口为终点的道路的交通灯**。在任何时刻，每个路口最多有一个交通灯为绿灯。只有绿灯道路终点上的车辆可以通过路口，其他红灯道路终点上的车辆必须排队等待。
3. 当交通灯变为绿灯后，队列中的第一辆车可以立即通过路口，没有延迟，每辆车通过路口的时间均为$1s$。如果绿灯持续n秒，则队列中的前n辆车可以恰好通过路口，其余车辆只能等待下一个绿灯。
4. 每个路口中的道路红绿灯的绿灯亮起时长之和小于总时间 D。

（3）车辆假设条件

1. 已知全部车辆的行程计划，即由一系列道路组成的行驶路径，并且不能改变。
2. 车辆的行驶速度都相同且不受其他车辆的影响。
3. 忽略车辆的几何尺寸，不考虑车辆排队时对空间的要求。
4. 每辆车的路径经过任一个路口至多一次。
5. 车辆的初始状态为，所有的车辆都在各自路径上第一条道路的终点处，根据该道路交通灯的状态等待或准备移动。如果有多辆车从同一条道路的终点出发，则 ID 号较小的车辆先行。
6. 车辆到达路径上最后一条道路终点时行程结束。某车辆行程结束后，将不再考虑该车辆。即该车辆不需在最后一条道路终点处排队也不需要通过路口。

（4）发生故障

1. 故障发生后，该道路立即永久封闭，其他道路正常通行。
2. 故障发生时，已经进入并正好位于该条道路上的车辆按原计划行驶，不受影响。

3. 所有计划经过该道路但尚未经过的车辆，只有到达该道路起点路口时，才能知道该道路已被封闭，立即按照里程最短原则重新选择后续行驶路径。其他车辆不得改变行程路径。

其中路口假设条件 4 是本文另设的假设条件，其余假设条件均为题目给出。

2.2 符号说明

编号	符号	意义
1	D	总时长，单位秒
2	I	路口总数
3	S	道路总数
4	V	车辆总数
5	F	车辆按时到达终点的基本得分
6	P_j	车j的行程路径的道路总数
7	T_j	车j的到达行程终点的时刻
8	M_j	车j的得分
9	T_j^r	车辆j在道路上行驶的总时间
10	T_j^w	车辆j在道路终点等待的总时间
11	T_j^p	车辆j通过路口的总时间
12	PA_j	车辆j路径上所有道路组成的集合
13	PA_j^1	车辆j路径上的第一条道路
14	K	车辆状态集合，$K=\{q_1,q_2,q_3\}$
15	$\sum$	状态机输入集合，$\sum=\{I_1,I_2,I_3,I_4,I_5\}$
16	δ	状态转移函数
17	q	车辆状态

3 问题一模型建立与求解

3.1 问题分析

本题是在全市的路网结构、车辆行程计划、车辆的行驶速度等信息不变且已知的情况下，合理配置每个路口交通灯在一个周期内亮起绿灯的顺序以及绿灯持续时长，使得在规定时间内所有车辆总得分最高。

单辆车的得分为：

$$M_j=\begin{cases}F+(D-T_j), T_j\le D\\ 0, \qquad\qquad T_j>0\end{cases} \tag{1}$$

目标函数为使所有车辆的总得分，则优化目标为：

$$\max\sum_{j=1}^{V}M_j \tag{2}$$

其中，V为总的车辆数，F为车辆到达终点的基本得分，D为总时长，T_j为车辆j到达路径终点的时刻。

公式(1)(2)为计算所有车辆的总得分的目标函数，计算该函数，需要知道每

辆车到达终点的时间。对于单个车辆j，其到达终点的时间为$T_j = T_j^r + T_j^w + T_j^p$。即车辆到达终点的时间可分为三部分：车辆在道路上行驶的总时间T_j^r、车辆在道路终点等待的总时间T_j^w和车辆通过路口的总时间T_j^p。

（1） 车辆在道路上行驶的总时间

由城市路网信息可知车辆在任意一条道路s的通行时间t_s。由车辆假设条件1、2可知，车辆路径已知且不变、车速不变。由车辆假设5、6可知，车辆的初始状态为路径第一条道路的终点处，终止状态为路径最后一条道路的终点处。因此车辆在道路上行驶的时间为车辆通过路径上第二条道路至最后一条道路的时间之和，即：

$$T_j^r = \sum_{s \in PA_j - PA_j^1} t_s \tag{3}$$

其中，PA_j为车辆j路径上所有道路组成的集合，PA_j^1为路径上的第一条道路。

（2） 车辆通过路口的总时间

由车辆假设条件3知，每辆车通过路口的时间均为$1s$，因此车辆通过路口的总时间为通过路口数量之和，车辆假设条件6中指明车辆不需要通过最后一条道路终点处的路口，因此车辆通过路口的数量为路径中总的道路数减一，即：

$$T_j^p = P_j - 1 \tag{4}$$

（3） 车辆在道路终点等待的总时间

计算车辆在道路终点处等待的时间，需要对车辆到达路径上每条道路终点时情况进行分析，求出在路径上每条道路终点的等待时间，进而求得等待的总时间。由道路假设条件3、4知，车辆到达道路终点时，存在四种情况：1）道路终点处的交通灯为绿灯且前方无等待车辆，此时无需等待；2）道路终点处的交通信号灯为绿灯，但道路较为拥堵，前方有等待车辆，则该车需等待其他车辆通过后再通过；3）道路终点处的交通灯为红灯，且前方无其它等待车辆，即该车为道路队列的第一位，此时车辆需等到绿灯亮起时才能通过路口；4）道路终点处的交通灯为红灯，但前方有 n 辆等待车辆，此时车辆需要等待其他车辆通过后再通过。综上，在道路终点的等待时间需要根据具体的调度方案和实时路况求出，该时间难以通过解析的方法求出，因此本文通过构建仿真模型求解。

3.2 模型构建

通过对问题的进一步的分析，本文首先建立车辆位置和道路的仿真模型，进一步利用仿真模型求出在已知调度方案下的目标函数，再利用启发式算法对调度方案进行优化。本小节主要对仿真模型的构建过程进行描述。

3.2.1 基于有限状态机的车辆状态转移模型

车辆在道路上行驶可以看作是一个基于交通规则的位置状态转换过程。有限状态机（Finite State Machine，FSM）是从一种工作状态转移到另一种工作状态的动态系统的表示，它包含了若干个离散的系统行为状态，不同状态之间可以根据预定的规则进行转换。为了详细描述在本题设的交通规则下车辆的位置状态，本文利用有限状态机对车辆位置状态转移的过程进行建模，具体的建模过程如下：

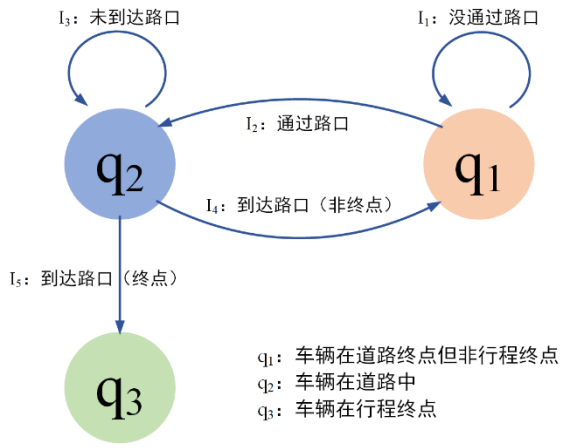

图 1 车辆位置状态转换示意图

车辆有限状态机模型（如图 1）可以描述为一个有序的五元组，记作$M=(K,\sum,\delta,q_0,F)$，具体含义为：

（1） K：车辆状态集合

$K=\{q_1,q_2,q_3\}$，我们将车辆的位置状态分为三种，分别为q_1,q_2,q_3。其中，q_1表示“车辆在道路终点但非行程终点”；q_2表示“车辆在道路中”；q_3表示“车辆在行程终点”。

（2） $\sum$：状态机输入集合

$\sum=\{I_1,I_2,I_3,I_4,I_5\}$，该集合表示使车辆位置状态发生改变的事件。其中，$I_1$表示“车辆没通过路口”；$I_2$表示“车辆通过路口”；$I_3$表示“未到达路口”；$I_4$表示“到达非终点的路口”；$I_5$表示“到达终点路口”。

（3） δ：状态转移函数

状态转移函数具体为，$\delta(q_1,I_1)=q_1\delta(q_1,I_2)=q_2,\delta(q_2,I_3)=q_2,\delta(q_2,I_4)=q_1,\delta(q_2,I_5)=q_3$，表 1 为状态转移的表格。

表 1 车辆状态机状态转移表

	I_1	I_2	I_3	I_4	I_5
q_1	q_1	q_2	$\emptyset$	$\emptyset$	$\emptyset$
q_2	$\emptyset$	$\emptyset$	q_2	q_1	q_3
q_3	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

（4） q_0：初始状态

根据题目中给出的条件知“初始状态（即 0 时刻），所有车辆都在各自路径上第一条道路的终点处”，故初始状态为q_1状态的一种特殊情况。

（5） F：终止状态

车辆到达行程终点时中终止，故终止状态与q_3一致。

在上述车辆位置状态转移模型的基础上，给出**车辆位置更新算法**如下：

Step1：根据车辆当前的位置信息得到当前车辆位置状态q_{car}；

注释：其中$q_{car}\in K$，车辆的位置信息是指车辆在道路上的具体位置。

Step2：判断车辆是否行驶到道路终点。

a. 如果没有到达终点，即$q_{car}=q_2$，转到 Step3；

注释：车辆在道路中行驶，下一时刻仍在道路中行驶。

b. 如果到达终点，即$q_{car} = q_1 \ or \ q_3$，转到 Step4；

注释：车辆到达了道路终点，需要判断是否能通过路口，才能确定下一时刻的车辆位置。

Step3：车辆状态不发生改变，直接返回 Step1；

Step4：判断车辆是否行驶到行程的终点；

a. 如果是，即$q_{car} = q_3$，转到 Step5；

注释：车辆移动结束，根据车辆假设 6，车辆移除队列。

b. 如果否，即$q_{car} = q_1$，转到 Step6；

注释：车辆到达了非行程终点的道路终点，需要进一步判读车辆是否能通过路口，才能确定下一时刻车辆的位置。

Step5：修改车辆的状态为终止态$\boldsymbol{q_3}$，结束；

Step6：判断车辆是否可以通过路口；

a. 如果是，转到 Step7；

注释：只有车辆所在道路的交通信号灯为绿灯且车辆排在道路终点处队列的第一位时才可通行，通过路口后，车辆进入下一条道路行驶。

b. 如果否，转到 Step8；

注释：在其他情形下，车辆都无法通过路口，需在道路终点处等待。

Step7：修改车辆状态为q_2，并回到 Step1;

Step8：保持车辆状态为q_1，并回到 Step1；

在车辆仿真模型中，每辆车辆位置会按照算法逐秒更新，算法流程图如图 2 所示。

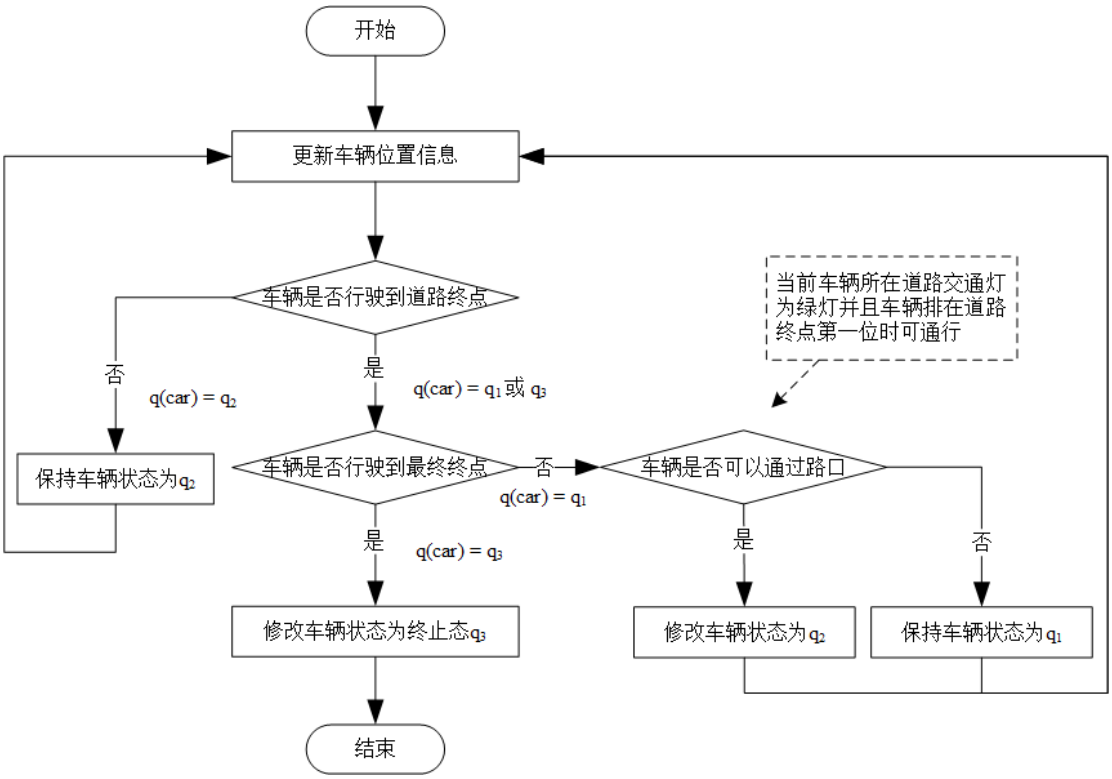

图 2 车辆位置更新流程图

道路信息主要是指每条道路终点处等待车辆的排队列表，而该道路信息会随着每辆车辆位置信息的变化而变化，下面给出当前车辆的**道路信息更新算法：**

Step1：输入当前车辆位置信息、运动状态$\boldsymbol{I_{car}}$与调度方案；

Step2：判断车辆状态是否发生变化；

a. 若车辆状态未发生变化（$\boldsymbol{I_{car} = I_1\ or\ I_3}$），则结束；

注释：车辆在道路上行驶或者在路口等待的过程中车辆状态不发生改变。

b. 若车辆状态为发生了变化（$\boldsymbol{I_{car} = I_2\ or\ I_4\ or\ I_5}$），转入 Step3；

注释：车辆从道路驶入路口或者从路口驶入道路的过程，车辆状态发生改变。

Step3：判断汽车状态是驶入路口还是驶出路口；

a. 若车辆驶入路口（$\boldsymbol{I_{car} = I_4\ or\ I_5}$），则进入 Step4；

注释：车辆驶入路口可能是最终路口，也可能是中间的路口

b. 若车辆驶出路口（$\boldsymbol{I_{car} = I_2}$），则进入 Step5；

Step4：判断车辆驶入的路口是否为行程终点；

a. 若为行程终点（$\boldsymbol{I_{car} = I_5}$），则结束；

b. 若不为终点（$\boldsymbol{I_{car} = I_4}$），进入 Step6；

Step5：道路终点队列弹出当前车辆；

Step6：车辆驶入中间路口（$\boldsymbol{I_{car} = I_4}$）。

根据该算法，可以仿真车辆位置的变化对道路信息的变化，具体流程如图 3 所示。

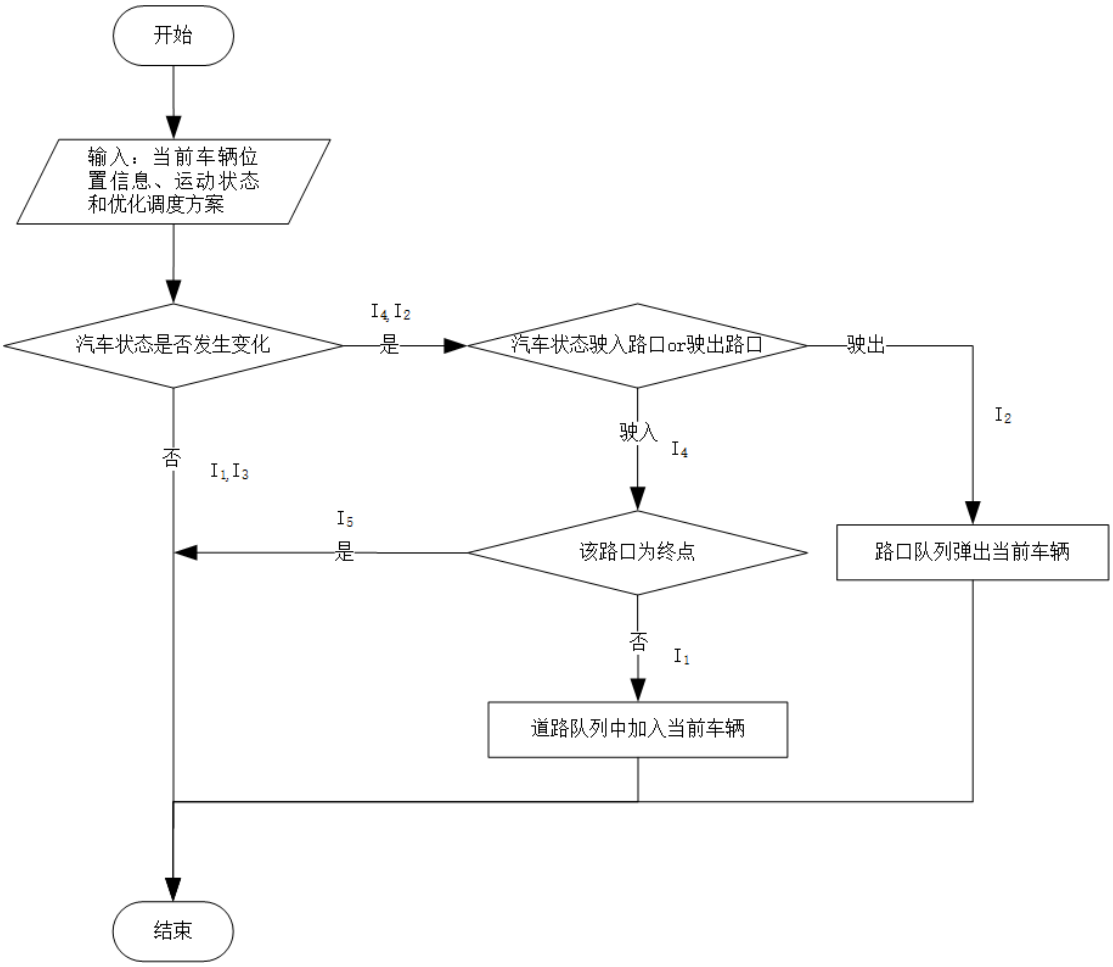

图 3 道路信息更新流程图

3.2.2 目标函数计算流程

基于上一小节的车辆位置更新算法和道路信息更新算法，对交通系统进行仿真，计算最终车辆总得分（目标函数），整个过程的仿真计算算法如下：

Step1：输入车辆行程信息“cars.txt”、路网信息“streets.txt”以及调度方案；

Step2：开始时，记时间$t = 0$；

Step3：获取第一辆车$C001$的位置信息（按照$\{C001, C002, \dots, C200\}$的顺序）；

Step4：将第一辆车的信息输入道路更新算法，更新道路信息；

Step5：获取下一辆车的位置信息；

Step6：判断所有车辆的道路信息是否更新完毕；

a. 若更新完毕，进入 Step7；

b. 若未更新完毕，返回 Step5；

Step7：所有车辆信息更新后，根据位置更新算法，更新第一辆车的位置信息，即车辆移动并状态转移；

Step8：根据位置更新算法，更新下一辆车的位置信息，即移动并状态转移；

Step9：判断车辆是否到达行程终点；

a. 若车辆到达行程终点，进入 Step10；

b. 若未到达行程终点，进入 Step11；

Step10：计算当前车辆的得分并加入总得分，进入 Step11；

Step11：判断所有车辆是否完成车辆位置信息更新；

a. 若已完成，进入 Step12；

b. 若未完成，返回 Step8；

Step12：时间更新$t = t + 1$；

Step13：判断当前仿真时间是否小于总的仿真时间；

a. 若小于，则回到 Step4；

b. 否则，算法结束。

根据该算法，可以求出在当前调度方案的总得分，具体的算法流程如图 4 所示。

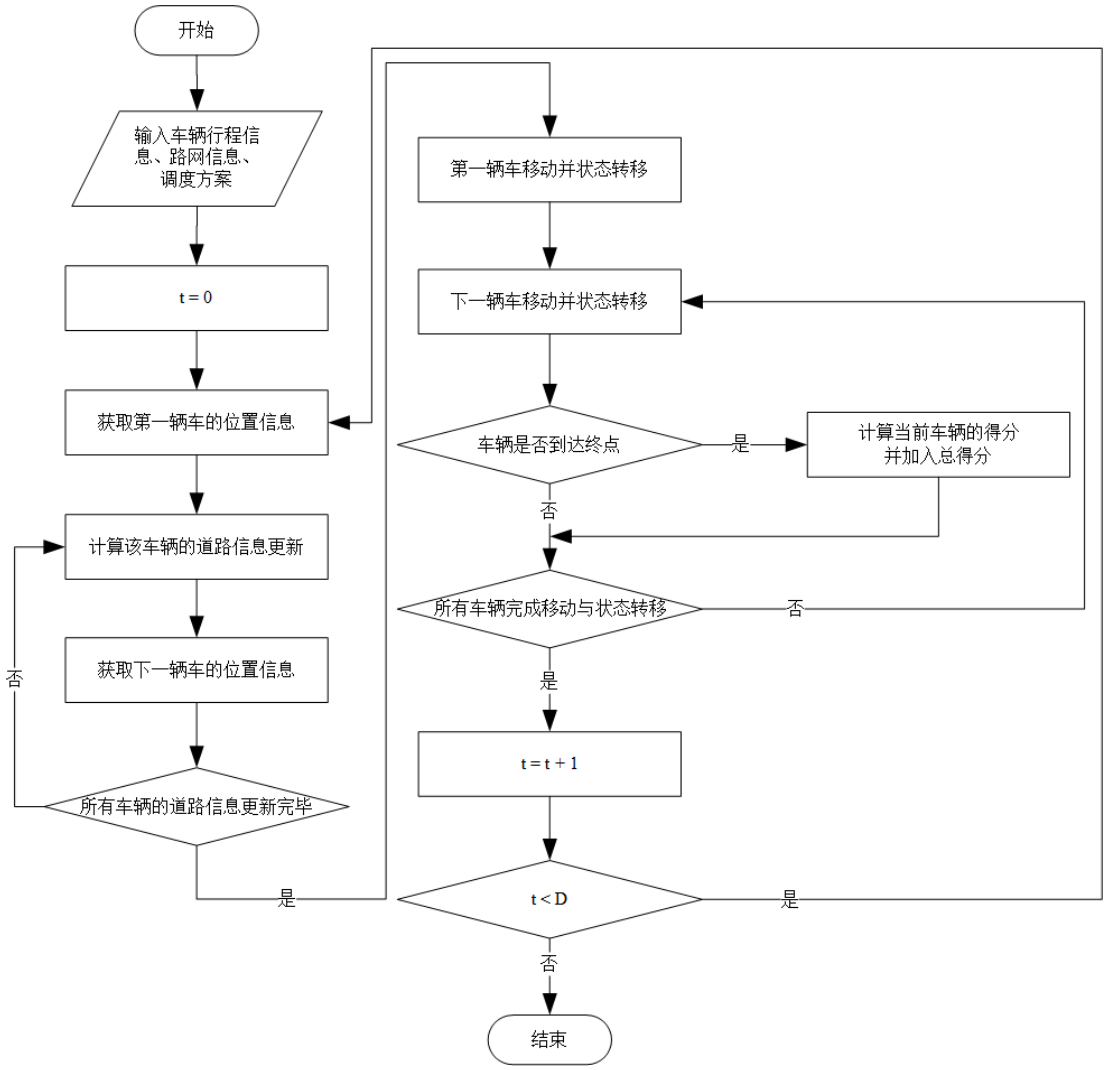

图 4 目标函数仿真计算流程图

3.3 优化模型求解

根据 3.2 构建的仿真模型，我们可以在已知调度方案的情况下，求解得到该调度方案的总得分。为了可以求出最佳的调度方案，我们设计遗传算法对该问题进行优化求解。

3.3.1 交通灯全局优化调控策略

由于本文的求解空间巨大，时间复杂度较高，在没有限制策略的情况下，很难以收敛。算法运行一天都无法得到较好数值。因此，我们设计了**概率分布遗传算法**，对遗传算法中的初始化种群和变异操作进行了改变。传统的遗传算法中，初始化解的分布是均匀分布，而改进的遗传算法中其分布是概率性的，路口的红绿灯周期越短，其出现的概率越大。**我们假定一个路口的红绿灯的周期越短，那么每个道路上等待的汽车的等待时间也就越短。**改进后的遗传算法如图 5，后续我们将对相关机制进行详细说明。

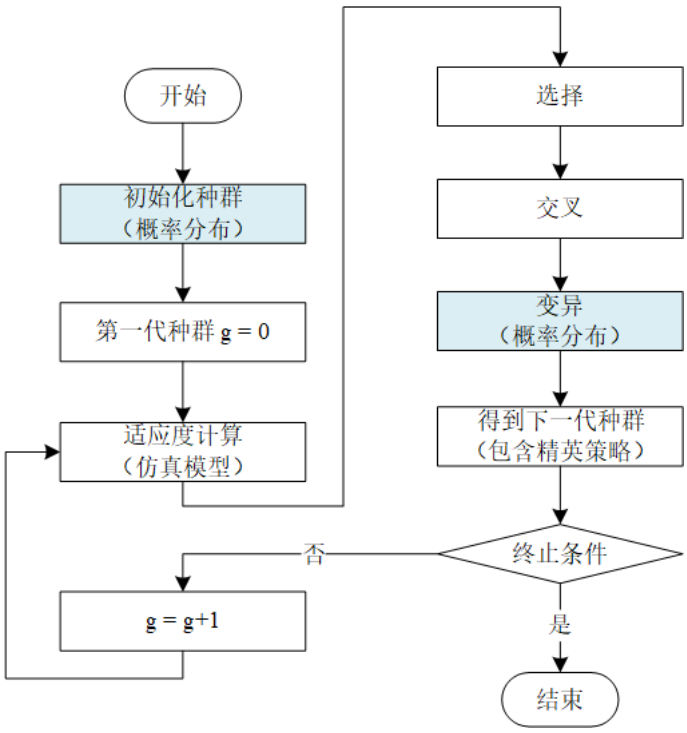

图 5 基于概率分布的遗传算法流程图

（1）编码解码规则

本文的基因设计即为调度方案的数据结构设计。根据题设条件，调度方案是每个路口的道路交通灯在一个周期内亮起绿灯的顺序以及持续时长。因此，我们设计基因为一个 python 的字典数据类型，根据路口名称索引得到该路口具有红绿灯的道路，这些道路按照一定顺序排列成为一个列表，同时这个列表中的每一项包含两个元素，一个是道路名称，另一个是绿灯持续时间。具体基因设计图如图 6。

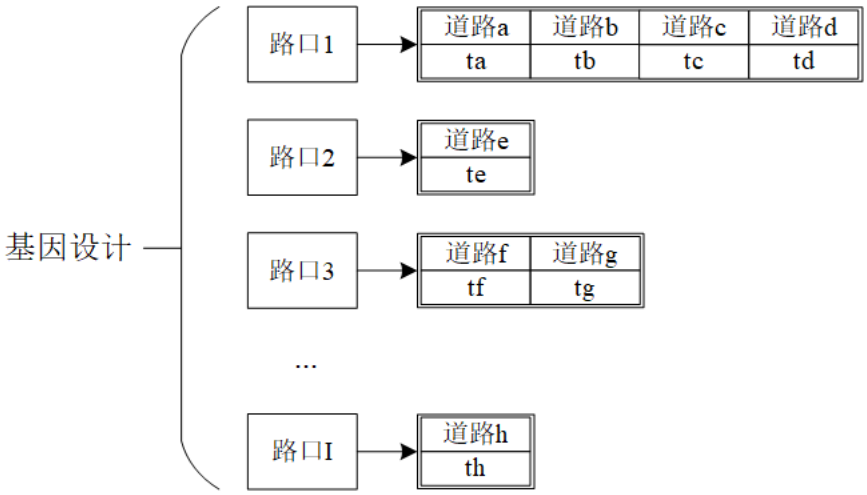

图 6 基因设计图

通过基因设计，我们可以解码得到在任意时刻，任意路口亮起绿灯的道路。

（2）遗传算子

遗传算法的核心在于基于编码规则通过遗传算子操作，迭代寻优。遗传算法中需要用到的几种遗传算子包括：初始化算子、选择算子、交叉算子和变异算子。其中初始化算子主要用于获得初代种群，选择算子则是挑选出种群中个体，交叉和变异算子用于产生下一代个体。本小节中，我们首先给出适应度函数，为选择算子提供依据，接着将依次介绍四种遗传算子。

a. 适应度函数

适应度函数主要用于评价一个种群中个体的优劣，即评价解的好坏。本文所采用的适应度函数时 3.2 所提及的仿真模型，种群中的一个个体基因输入到仿真模型中，通过计算得到的总得分就是该个体的适应度。

b. 初始化算子

初始化算子需要完成的工作就是生成若干初始个体（解）构成第一代种群。本章设计的改进遗传算法，初始化过程中主要考虑了：**路口假设条件 4**，每个路口中的道路红绿灯的绿灯亮起时长之和小于总时间 D。根据策略假定：一个路口的红绿灯的周期越短，那么每个道路上等待的汽车的等待时间也就越短。在随机生成道路的绿灯持续时间的过程中，更容易生成持续时间较短的周期。

通过上述初始化过程，可以得到一个初代个体，根据遗传算法设置的种群规模，将初始化算子运行多次，即可得到初代种群。

c. 选择算子

选择算子主要是挑选种群中优秀的个体进行交叉和变异操作。本文中采用的两两比较的方式选择候选个体：

1）随机从种群中选出两个个体 A 和 B；

2）计算个体适应度函数值，并进行比较，若 $Fitness(A) \leq Fitness(B)$，则保留 A 至候选个体集合；

3）若种群规模设置为PN， 则重复$PN-1$次步骤 1）和 2），得到包含$PN-1$个个体的候选个体集合；

为了获得更好的遗传效果，我们采用精英策略，即每代适应度函数值最优的个体（精英个体）直接挑选进入候选个体集合。因此，上述选择得到的$PN-1$个个体和精英个体组成了规模为PN的候选个体集。需要说明的是，其中可能存在重复的个体，但这对于求解过程不会造成影响，且适应度函数值越小的个体，出现重复的概率越高。

d. 交叉算子

通过选择算子，得到了候选个体集合，进一步需要采用遗传算子对候选个体集合中的个体进行遗传操作才能得到子代个体。交叉是遗传算法中最主要的操作算子，通过父代个体两两互换基因，形成新的子代个体。针对本文的题设条件，设计路口交叉算法。算法随机挑选一系列路口，将这些路口对应的调度列表进行交换。具体流程如图 7。

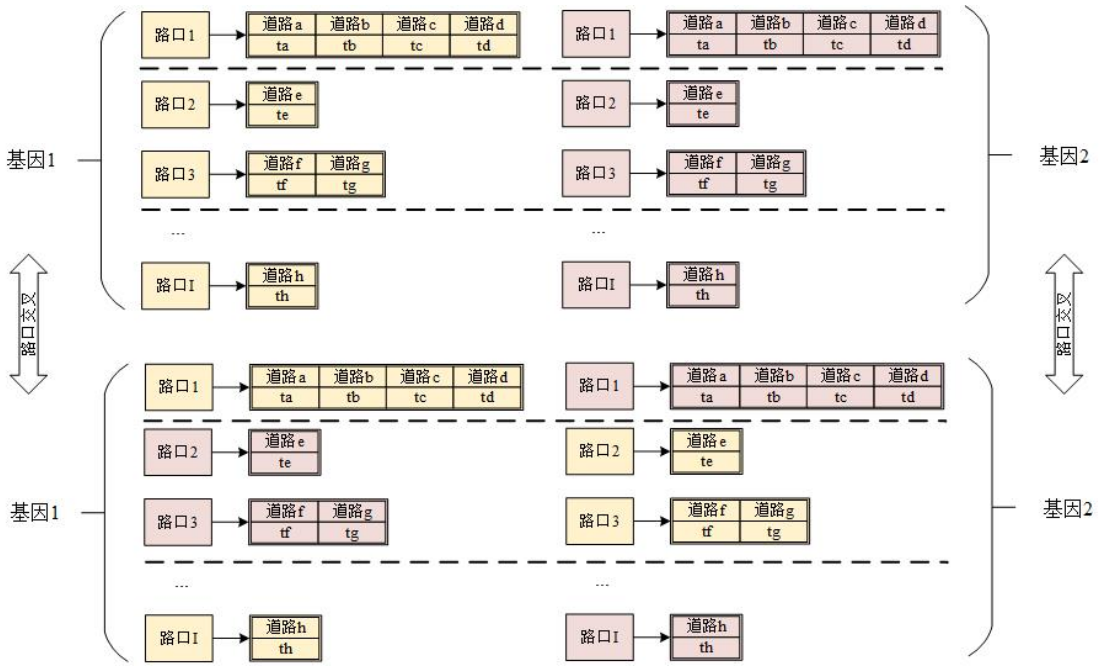

图 7 交叉算子示意图

完成交叉操作的父代个体将从选择算子得到的候选个体集合中移除，然后继续从集合剩余个体中挑选两个作为父代个体，继续交叉操作，候选个体集合为空时，可以得到PN个子代个体。

e. 变异算子

交叉算子是通过两个父代个体通过基因交换得到子代个体，这种方式有可能会出现收敛到局部极值的问题。为了尽量避免收敛至局部极值，我们还可以通过变异算子操作，来使种群保持多样化，保持搜索的广泛性，从而获得更好的整体效能。变异操作是针对单个个体，它随机确定个体的变异基因点位，通过直接改变该点位基因的方式获得新个体的过程。根据策略假定：一个路口的红绿灯的周期越短，那么每个道路上等待的汽车的等待时间也就越短。在变异操作时，路口更容易生成持续时间较短的周期。

变异主要是保持种群的多样化，从而尽最大可能避免收敛至局部极值。基于这一考虑，本章中主要考虑两种变异方式：**1）两点顺序变异**，该变异情况表示为同一路口的两段道路的顺序发生改变，但他们的绿灯持续时间不发生改变，两点顺序变异的示意图如 8；**2）单点时间变异**，该变异情况表示为某条道路的绿灯持续时间发生改变，单点顺序变异的示意图如图 9。

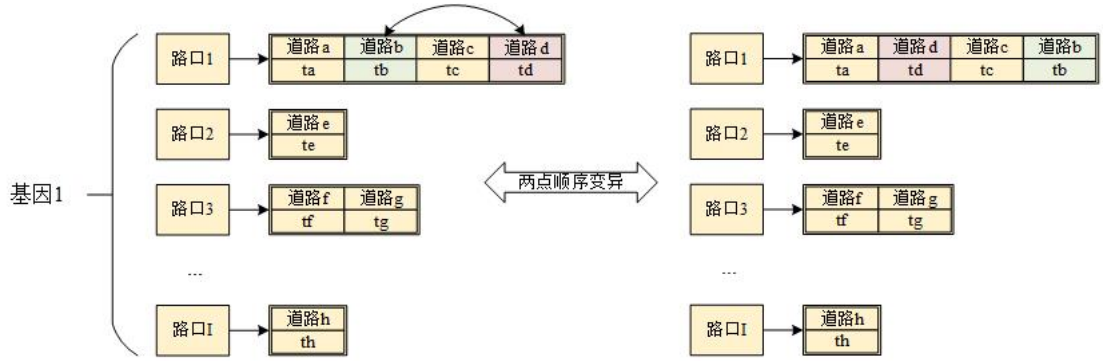

图 8 两点顺序变异

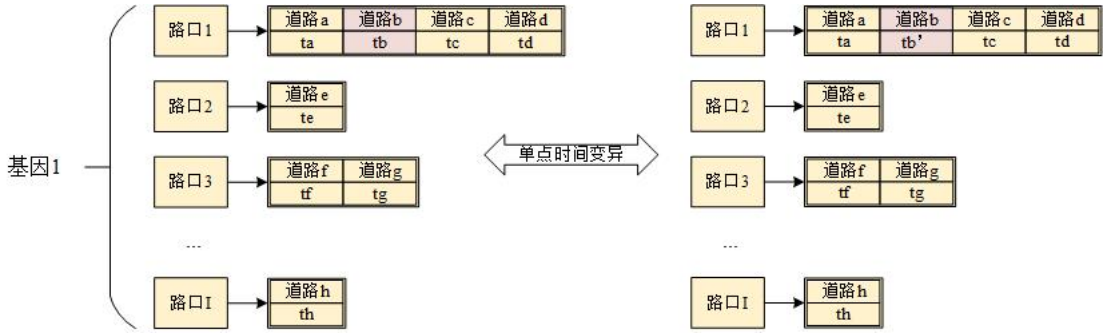

图 9 单点时间变异

3.3.2 算法适用的城市路网规模分析

为了验证算法适用的城市路网规模，我们根据表 2 举例，列举三种不同规模的城市路网，分别为小规模城市路网，中规模城市路网，以及大规模城市路网。

表 2 不同路网规模的路口、道路信息

	小规模城市路网	中规模城市路网	大规模城市路网
路口总数（个）	800	8000	10000
道路总数（条）	6000	60000	100000
车辆总数（辆）	20	200	400

为了更真实的生成网络，我们首先分析城市路网规模的特点，根据本题给出的道路信息（表 3），我们计算得到实际路网的节点平均度与平均路径长度。根据这两项指标，利用 python 的 networkx 包下 random_regular_graph 函数实现随机网络算法，构建不同规模城市路网的实际连通信息。

表 3 实际路网的网络特征信息

	节点平均度	平均路径长度
实际城市路网	7.996	3.837

表 4 是在已知三种规模的实际路网情况下，比较不同规模的路网运行 10 次仿真系统的程序执行时间。根据该表，可以看出，本算法并不适用于大规模的路网结构，运行 10 次就需要 336 秒，而运行中规模的城市路网可以在 73 秒左右完成十次计算，而小规模的路网结构可以快速完成计算。

表 4 不同规模路网仿真系统运行时间

	小规模城市路网	中规模城市路网	大规模城市路网
仿真系统运行 10 次时间（秒）	17.3	73.2	336.5

表 5 是在求解调度方案的过程中，计算调度方案收敛时的程序运行时间，这里对比了传统的遗传算法和概率分布遗传算法的效率对比。在迭代 100 次后，概率分布的遗传算法已经收敛，并且适应度函数远远大于传统的遗传算法，收敛时间和收敛迭代次数的远远小于传统的遗传算法，因此概率分布遗传算法更适用于调度方案的选取，在下一小节的调度方案计算中，采取概率分布遗传算法来进行计算。

表 5 传统遗传算法与概率分布遗传算法效率对比

	传统遗传算法	概率分布遗传算法
迭代 100 次后的总得分	64341	129281
收敛代数	1000	100
计算时间（秒）	87513	11816

3.3.1 最优调度方案

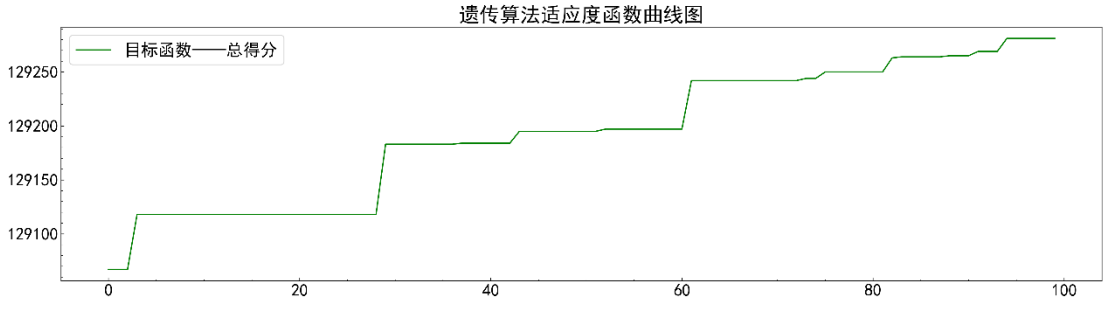

图 10 遗传算法适应度函数图

根据遗传算法迭代，迭代停止后的精英基因作为最优调度方案，具体结果保存在“result1.txt”文件中。在遗传算法的迭代过程如图 10 所示，根据图像可以看出，在迭代 100 次左右，算法已基本得到收敛，得到精英基因的得分为 129281，即我们求得的最优调度方案的得分为 129281。我们称在考虑其他车辆和路况的

情况下，任意一辆车都不需要等待的情况为理想情形，此时所有车辆的总得分为132222。计算可得此调度方案的得分为理想情形下得分的97.78%，说明优化效果较好。

4 问题二模型建立与求解

4.1 问题分析

问题二需要求出发生道路故障后，调整后的车辆路径和调度方案。

题目给出一个道路发生故障的情形，即 30s 时，道路 ebf-ffef 发生故障。此时交通系统的调度方案是第一问中求得的最优方案，发生故障后，受到影响的车需要改变行程路径，因此首先需要找出受到道路故障影响的车辆，再利用算法规划车辆新路径。最后，得到与第一问略有不同的路网结构（道路 ebf-ffef 发生故障）和车辆行程计划（受道路故障影响的车辆改变了行程计划），在此基础上调整第一问的调度方案，得到调整后的调度方案。因此本题的求解可以分为两步，第一步为车辆新路径求解，第二步为新的调度方案求解。

首先分析车辆新路径的求解，在 30s 道路 ebf-ffef 发生故障后，根据故障假设 2、3 知，受到影响的车辆要同时满足两个条件：1）原计划通过该道路；2）30s 时，车辆还未到达该道路起点处的路口 415（路口 ID）。通过计算可以得到，受到故障影响的车辆仅有三辆，具体信息如表 6 所示。

表 6 受到故障影响的车辆信息

车辆名称	到路口 415 前经过的道路数（条）	到路口 415 入口时间（s）	车辆的行程终点(路口 ID)
C016	7	220	1749
C035	9	281	2337
C050	4	134	1957

根据故障假设 4，受到故障影响的车辆只有到达路 415 时才能知道道路 ebf-ffef 已被封闭，然后才能按照历程最短原则重新选择后续路径，因此要求车辆调整后的路线，只需求路口 415 到车辆行程终点的最短路径。即路网中，节点 415 分别至节点 1749、2337、1957 的最短路径。本文利用经典的最短路径算法 Dijkstra 算法求解。

依据调整后的车辆信息，利用 1.2 节提出的概率分布遗传算法可优化得到新的调度方案。

4.2 模型建立

4.2.1 Dijkstra 算法

Dijkstra 算法是一个基于贪心、广度优先搜索、动态规划求网络中一个点到其他所有点的最短路径的算法，时间复杂度为$O(n^2)$。用 Dijkstra 算法求出路口节点 415 到其他所有点的最短路径，也就得到了节点 415 分别至节点 1749、2337、1957 的最短路径，即受到障碍影响的三辆车调整后的路径。下面简要介绍 Dijkstra

算法[1]。

我们利用具有权重的有向网络$G=(V,E,W)$表示路网结构，其中$V=\{v_0,\dots,v_{8000}\}$为路口节点集合，$E=\{e_1,e_2,\dots,e_{63967}\}$为路网中的所有道路，如$e_1$表示道路 a-e，也可以用该道路的起点路口和终点路口表示，即$e_1=(v_0,v_4)$，$W=\{w_1,w_2,\dots,w_{63967}\}$为路网中的所有道路的路径长度。当 ebf-ffef 发生故障时，永久封闭，因此将该条线段移出路网。

算法基本思想为：从起点v_s出发，逐步地向外探寻最短路。执行过程中，不断记录其他节点的P表与T表。P表记录从v_s到其他点的最短路径长度，T表记录从v_s到其他点的最短路径长度的上界。算法的每一步是去修改T表，并且把某一个T表的点改变为具有P表的点，从而使每一步都会求出起点v_s到某个节点的最短路径。具体算法流程为：

Step1：初始时，令$S_0=\{v_{415}\}$，$P(v_{415})=0$，$\lambda(v_{415})=0$，对于每一个$v\neq v_{415}$，令$T(v)=+\infty$，$\lambda(v)=M$，令$k=s$；

Step2：如果$\{v_{1749},v_{2337},v_{1957}\}\subset S_i$，算法终止；否则转入 Step3；

注释：此时，节点v_{415}到任意$v\in\{v_{1749},v_{2337},v_{1957}\}$的最短路径长度为$d(v_{415},v)=P(v)$，相应的最短路径也可求出。

Step3：考察每个使$(v_k,v_j)\in E$且$v_j\notin S_i$的节点v_j；

a. 如果$T(v_j)>P(v_k)+w_{(v_k,v_i)}$，则把$T(v_j)$修改为$P(v_k)+w_{(v_k,v_i)}$，把$\lambda(v_j)$修改为$k$；

b. 否则，转入 Step4；

Step4：令$T(v_{j_i})=\min\limits_{v_k\notin S_i}\{T(v_k)\}$

a. 如果$T(v_{j_i})<+\infty$，则把v_{j_i}的 T 标号变为P标号，$P(v_{j_i})=T(v_{j_i})$，$S_{i+1}=S_i\cup\{v_{j_i}\}$，$k=j_i$，把$i$换成$i+1$，转入 Step2；

b. 否则，算法终止；

4.3 模型求解

4.3.1 调整后车辆路径

根据 1.2 所提及的算法，计算表 6 中车辆调整后的车辆路径，得到表 7，并将该表格根据 cars.txt 的格式，保存成 newcars.txt 文件。

表 7 调整后车辆路径信息

车辆名称	车辆经过道路数（条）	车辆经过道路
C016	12	dbh-fjai,fjai-eggf,eggf-ebbe,ebbe-ebi,ebi-ghfc,ghfc-edaj,edaj-ebf,efb-cih,cih-bjc,bjc-iie,iie-f,f-bhej
C035	13	bch-djei,djei-ccii,ccii-ceag,ceag-eeg,eeg-bjif,bjif-dadh,dadh-ejje,ejje-egd,egd-ebf,ebf-dci,dci-ddac,ddac-ba,ba-cddh
C050	10	gg-djjc,djjc-cede,cede-dci,dci-ebf,ebf-dci,dci-ddac,ddac-ba,ba-ijh,ijh-bhi,bhi-bjfh

4.3.2 局部调整后的调度方案

根据 newcars.txt 的更改信息，通过遗传算法进行迭代，将迭代停止后的精英基因保存为 result2.txt 中。在遗传算法的迭代过程如图 11 所示，根据图像可以看出，在迭代 100 次左右，算法已基本得到收敛。得到精英基因的得分为 130097。对于调整路径后的车辆，理想情形下的总得分为 133083，本调度方案的得分占总得分的97.76%，说明本调度方案较为理想。

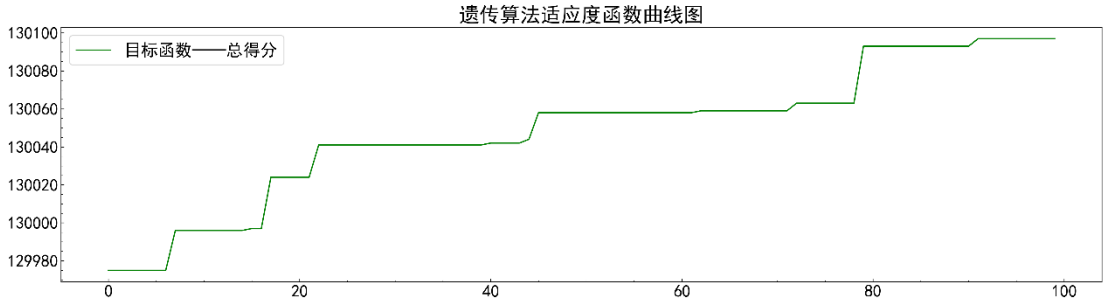

图 11 遗传算法适应度函数曲线

5 总结

本文首先构建了基于有限状态机的车辆道路交通仿真系统，可以仿真计算任意调度方案下的车辆总得分，算法简单，可行性高，计算方便。其次设计基于概率分布的遗传算法，完成对调度方案的优化求解。对比传统遗传算法，基于概率分布的遗传算法可以加速 7 倍以上的时间，可以更加快速计算出收敛的调度方案。再次设计了不同规模的，贴近真实道路的交通路网结构，分析传统遗传算法和基于概率分布的遗传算法的可行性。最后分析道路故障对车辆路径的影响，提出基于 Dijkstra 的车辆路径规划算法，并应用概率分布遗传算法，求解得到新的优化调度方案，可以及时解决道路故障造成的拥堵现象，进而提高交通系统的鲁棒性。本文提出的方法依旧存在计算时间慢的特点，在后续研究中，会采取更高算力的电脑进行加速计算，从而使得调度方案可以更加及时的应用到交通系统当中。

参考文献

[1] 胡运权，运筹学，北京：清华大学出版社，263-265，2005。

附录：代码（python）
文件名称：GA.py
文件内容：遗传算法主函数，**全局 main 函数**

```
from gene import initPops
from fitness import fitness
from select import select
from mutation import mutate
from crossover import crossover
from plotFitness import plotFitness
import pandas as pd
import numpy as np

def GA(nIter, nPop, pc, pm):
    pops = initPops(nPop)
    fitList = []
    for i in range(nIter):
        fits = fitness(pops)

        # print('适应度函数计算完成')

        bestFit = min(fits)
        bestId = fits.tolist().index(bestFit)
        bestPop = pops[bestId]
        pops, fits = select(nPop, pops, fits)
        pops = crossover(pops, pc)
        pops = mutate(pops, pm)
        pops = np.hstack((pops, bestPop))

        print('第'+str(i)+'代已完成，最大值为'+str(-
bestFit))
        fitList.append(-bestFit)
    bestPop.savegene()
    plotFitness(fitList)


if __name__ == '__main__':
    GA(100, 20, 0.7, 0.3)
```

附录：代码（python）
文件名称：GlobalVar.py
文件内容：全局变量类，记录全局变量

```
import pandas as pd
```

```
    D = 858     # 总时长

    I = 8000    # 路口总数

    S = 63968   # 道路总数

    V = 200     # 车辆总数

    F = 250     # 基本得分


    streets = pd.read_csv('streets.txt', index_col='name')
    # streets = np.array(streets)

    streetsTable = 0

    carFile = open('cars.txt')
    carLine = carFile.readline()
    cars = []
    crossing = set()
    crossLine = {}
    line = set()

    while carLine:
        carLine = carLine.replace("\n", "")
        carData = carLine.split(',')
        for i in carData[2:len(carData)-1]:
            line.add(i)
            # crossing.add(streets.loc[i][0])
            crossingi = streets.loc[i][1]
            crossing.add(crossingi)
            if crossLine.__contains__(crossingi):
                crossLine[crossingi].append(i)
            else:
                crossLine[crossingi] = [i]

        cars.append([carData[0], carData[1],
carData[2:len(carData)]])
        carLine = carFile.readline()
    carFile.close()

    crossMutate = []

    for i in crossLine.keys():
```

```
        if len(crossLine[i]) != 1:
            crossMutate.append(i)

    infectCar = []
    for car in cars:
        index = 0
        for street in car[2]:
            if street == 'ebf-ffef':
                infectCar.append([car[0], index])
            index += 1
```

附录：代码（python）
文件名称：Car.py
文件内容：车辆类，记录车辆的状态转移函数

```
    from GlobalVar import streets
    import copy
    class Car:
        def __init__(self, infor):
            self.name = infor[0]
            self.streetNum = infor[1]
            self.streetList = infor[2:len(infor)][0]
            self.positionid = 0
            self.street = self.streetList[self.positionid]
            self.pastStreet = -1
            # print(self.streetList)
            self.cross = streets.loc[self.street][1]
            self.changeType = 1
            self.positionType = 2
            self.position = 0

        def move(self, streetsMessage, t):
            if self.positionType == 1:
                # 道路中

                self.position += 1
                if self.position >=
streets.loc[self.street][2]:
                    # 行驶到道路终点
                    # print(self.positionid)
                    if self.positionid ==
len(self.streetList)-1:
                        self.changeType = 0
```

```
                    # 行驶到最终的道路终点
                    return 3
                self.cross = streets.loc[self.street][1]
                self.positionType = 2
                self.changeType = 1
                self.position = 0
                # 行驶到路口
                return 2
            else:
                self.changeType = 0
                # 行驶在路上
                return 1

        elif self.positionType == 2:
            # 路口
            if streetsMessage.judgeMove(self, t):

                self.position = 0
                self.pastStreet =
copy.deepcopy(self.street)
                self.positionType = 1
                self.positionid += 1
                self.changeType = 2
                self.pastcross =
copy.deepcopy(self.cross)
                self.street =
self.streetList[self.positionid]
                self.cross = streets.loc[self.street][1]
                # print(self.name, self.positionType,
self.changeType)
                # 行驶出路口
                return 1
            else:
                self.changeType = 0
                # 停在路口
                return 2
        else:
            return 3
```

附录：代码（python）
文件名称：Streets.py
文件内容：道路类，记录道路的信息更新函数

```python
from randTime import passStreet
from GlobalVar import crossMutate
import pandas as pd
class Queue:
    def __init__(self):
        self.list = []

    def put(self, item):
        self.list.append(item)

    def get(self):
        temp = self.list[0]
        del(self.list[0])
        return temp

    def search(self, num):
        if num in self.list:
            return True
        else:
            return False

    def top(self):
        return self.list[0]


class Streets:
    def __init__(self, gene):
        self.gene = gene
        self.crossingDict = {}
        self.crossingMessage = []

    def changeState(self, cars, t):
        for car in cars:
            if car.changeType == 0:
                # 汽车的状态没变
                continue
            elif car.changeType == 1:
```

```
                # print(car.name + '驶入' +
str(car.cross))
                # 汽车驶入路口
                if car.cross in crossMutate:

self.crossingMessage.append([car.cross, car.street, t, '驶
入'])
                if
self.crossingDict.__contains__(car.street):

self.crossingDict[car.street].put(car.name)
                else:
                    q = Queue()
                    q.put(car.name)
                    self.crossingDict[car.street] = q
            elif car.changeType == 2:
                # 汽车驶出路口
                carName =
self.crossingDict[car.pastStreet].get()
                car.changeType = 0
                #
print(self.crossingDict[car.pastStreet].list)
                if car.name != carName:

print(self.crossingDict[car.pastStreet].list)
                    print('error')

    def judgeMove(self, car, t):
        # 判断车在 t 时刻是否可以行驶，当车所在的街道可以通行，并
且车排在路口第一位
        # return True
        passstreet =
passStreet(self.gene.chromesome[car.cross], t)
        # print(self.crossingDict)
        if car.pastStreet == -1:
            if car.street == passstreet:
                return True
            else:
```

```
                return False
            else:
                if car.street == passstreet and car.name ==
self.crossingDict[car.street].top():
                    return True
                else:
                    return False
        def saveMessage(self):
            saveData =
pd.DataFrame(data=self.crossingMessage)
            saveData.to_csv('message.csv', encoding='gbk')
```

附录：代码（python）

文件名称：gene.py

文件内容：调度方案基因类，初始化基因

```
    from GlobalVar import cars, crossing, line, crossLine,
D, crossMutate
    import matplotlib.pyplot as plt
    import numpy as np
    from randTime import randTime, randOneTime
    import random
    import pandas as pd
    from copy import deepcopy
    class gene:
        def __init__(self):
            # 构造函数 —— 基因初始化
            self.chromesome = {}
            for cross in crossLine:
                crossgene = []
                streetList = crossLine[cross]
                if len(streetList) == 1:
                    crossgene.append([streetList[0], D])
                else:
                    timeList = randTime(D, len(streetList))
                    for street, time in zip(streetList,
timeList):
                        crossgene.append([street, time])

                    random.shuffle(crossgene)
                self.chromesome[cross] = crossgene
        def mutate(self):
            # 单个基因的变异操作
```

```
            '''
            单个基因的变异操作
            1：单点时间变异
            2：全部时间变异
            3：两点顺序变异
            4：全部顺序变异
            :return:
            '''
            rand = random.random()

            randCross = random.sample(crossMutate, 1)[0]
            if rand < 0.4:
                # 单点时间变异
                randStreet =
random.sample(range(len(self.chromesome[randCross])),
1)[0]
                # print(randStreet)
                self.chromesome[randCross][randStreet][1] =
randOneTime(D, self.chromesome[randCross],
self.chromesome[randCross][randStreet][1])
            elif rand < 0.5:
                # 全部时间变异
                timeList = randTime(D,
len(self.chromesome[randCross]))
                for i, time in
zip(range(len(self.chromesome[randCross])), timeList):
                    self.chromesome[randCross][i][1] = time
            elif rand < 0.9:
                # 两点顺序变异
                if len(self.chromesome[randCross]) >= 2:
                    i, j =
random.sample(range(len(self.chromesome[randCross])), 2)
                    self.chromesome[randCross][i],
self.chromesome[randCross][j] =
self.chromesome[randCross][j],
self.chromesome[randCross][i]
            else:
```

```
                # 全部顺序变异
                random.shuffle(self.chromesome[randCross])

        def savegene(self):
            f1 = open('result.txt', 'w')
            f1.write(str(len(self.chromesome)) + '\n')
            geneList = []
            for i in self.chromesome.keys():
                f1.write(str(i)+'\n')
                f1.write(str(len(self.chromesome[i]))+'\n')
                for j in self.chromesome[i]:
                    if i ==
list(self.chromesome.keys())[len(self.chromesome.keys())
- 1] and j == self.chromesome[i][len(self.chromesome[i])-
1]:
                        f1.write(str(j[0]) + ',' + str(j[1]))
                    else:

f1.write(str(j[0])+','+str(j[1])+'\n')
            f1.close()


    class gene2:
        def __init__(self):
            # 构造函数 —— 基因初始化
            self.chromesome = {}
            for cross in crossLine:
                crossgene = []
                streetList = crossLine[cross]
                if len(streetList) == 1:
                    crossgene.append([streetList[0], D])
                else:
                    timeList = [random.randint(1, 5) for i in
range(len(streetList))]
                    for street, time in zip(streetList,
timeList):
                        crossgene.append([street, time])

                    random.shuffle(crossgene)
                self.chromesome[cross] = crossgene

        def savegene(self):
```

```
            f1 = open('result2.txt', 'w')
            f1.write(str(len(self.chromesome)) + '\n')
            geneList = []
            for i in self.chromesome.keys():
                f1.write(str(i)+'\n')
                f1.write(str(len(self.chromesome[i]))+'\n')
                for j in self.chromesome[i]:
                    if i ==
list(self.chromesome.keys())[len(self.chromesome.keys())
- 1] and j == self.chromesome[i][len(self.chromesome[i])-
1]:
                        f1.write(str(j[0]) + ',' + str(j[1]))
                    else:

f1.write(str(j[0])+','+str(j[1])+'\n')
            f1.close()

    def initPops(nPop):

        # 初始化种群

        pops = []
        for i in range(nPop):
            pops.append(gene())
        return np.array(pops)
```

附录：代码（python）
文件名称：randTime.py
文件内容：工具类，随机生成每个道路的绿灯通行时间，求解当前路口的绿灯的隧道

```
    import random


    def randTime(maxValue, num):

        '''生成总和小于某数的整数序列

        maxValue: 序列总和

        num: 要生成的整数个数


        return

        per_all_persons:list,指定 num 个接种点各自待接种的人数

        '''
```

```
        maxValue = random.randint(num, maxValue)
        suiji_ser = random.sample(range(1, maxValue), k=num
- 1)  # 在 1~maxValue 之间，采集 20 个数据

        suiji_ser.append(0)  # 加上数据开头
        suiji_ser.append(maxValue)
        suiji_ser = sorted(suiji_ser)
        per_all_persons = [random.randint(1,4) for i in
range(1, len(suiji_ser))]  # 列表推导式，计算列表中每两个数之间
的间隔
        return per_all_persons

    def randOneTime(maxValue, crossAssign, time):
        '''
        返回随机时间
        :param maxValue:
        :param timeList:
        :param time:
        :return:
        '''
        # print(crossAssign)
        timeList = [i[1] for i in crossAssign]
        maxValue -= (sum(timeList) - time)
        return random.randint(1, maxValue)


    def passStreet(cross, time):
        crossTimeList = [i[1] for i in cross]
        T = sum(crossTimeList)
        t = time % T
        streetTime = cross[0][1]
        for i in range(1, len(cross)):
            if t < streetTime:
                return cross[i-1][0]
            else:
                streetTime += cross[i][1]
        return cross[len(cross) - 1][0]
```

附录：代码（python）
文件名称：fitness.py
文件内容：仿真模型类，计算得到适应度函数

```
from GlobalVar import D, cars, F
```

```python
from Streets import Streets
from Car import Car
import numpy as np
from gene import initPops

def simulation(g):
    # 基因的适应度函数计算
    Cars = []
    for car in cars:
        Cars.append(Car(car))
    streets = Streets(g)
    M = 0
    for time in range(D):
        # print(time)
        if Cars == []:
            break

        streets.changeState(Cars, time)
        for car in Cars:
            if car.move(streets, time) == 3:
                M += (F + D - time)
                # print(car.name, time)
                Cars.remove(car)
    # streets.saveMessage()
    return M

def fitness(pops):
    fit = []
    for genei in pops:
        # print(genei)
        fit.append(- simulation(genei))
return np.array(fit)
```

附录：代码（python）
文件名称：select.py
文件内容：遗传算法选择函数

```python
import numpy as np
import random


def select(pool, pops, fits):
    nPop = len(pops)
    newPops = [0] * pool
```

```
        newFits = [0] * pool
        indices = np.arange(nPop).tolist()

        i = 0
        while i < pool:
            idx1, idx2 = random.sample(indices, 2)
            idx = compare(idx1, idx2, fits)
            newPops[i] = pops[idx]
            newFits[i] = fits[idx]
            i += 1
        return np.array(newPops), np.array(newFits)


    def compare(idx1, idx2, fits):
        if fits[idx1] <= fits[idx2]:
            return idx1
        else:
            return idx2
```

附录：代码（python）
文件名称：crossover.py
文件内容：遗传算法交叉函数

```
    import copy
    import numpy as np
    import random
    from gene import gene
    def crossover(pops, pc):
        chrPops = copy.deepcopy(pops)
        nPop = len(chrPops)
        for i in range(0, nPop, 2):
            if np.random.rand() < pc:
                SBX(chrPops[i], chrPops[i+1])
        return chrPops

    def SBX(chr1, chr2):

        # 路口交叉

        pos1, pos2 = np.sort(np.random.randint(0,
len(chr1.chromesome), 2))
        pos2 += 1
        keys = list(chr1.chromesome.keys())
        for key in keys[pos1:pos2]:
            chr1.chromesome[key], chr2.chromesome[key] =
chr2.chromesome[key], chr1.chromesome[key]
```

附录：代码（python）
文件名称：mutation.py
文件内容：遗传算法变异函数

```
import numpy as np


def mutate(pops, pm):
    nPop = len(pops)
    for i in range(nPop):
        if np.random.rand() < pm:
            pops[i].mutate()
    return pops
```

附录：代码（python）
文件名称：Dijkstra.py
文件内容：计算发生拥堵后的车辆

```
import pandas as pd
def num2Street(num):
    str = ''
    while num != 0:

        temp = num % 10
        # print(temp)
        num = int(num/10)
        str = chr(temp + 97) + str
    return str


D = 858     # 总时长

I = 8000    # 路口总数

S = 63968   # 道路总数

V = 200     # 车辆总数

F = 250     # 基本得分

streets = pd.read_csv('streets.txt', index_col='name')
# streets = np.array(streets)
```

```
    streetsTable = 0

    carFile = open('cars.txt')
    carLine = carFile.readline()
    cars = []
    crossing = set()
    crossLine = {}
    line = set()

    while carLine:
        carLine = carLine.replace("\n", "")
        carData = carLine.split(',')
        for i in carData[2:len(carData)-1]:
            line.add(i)
            # crossing.add(streets.loc[i][0])
            crossingi = streets.loc[i][1]
            crossing.add(crossingi)
            if crossLine.__contains__(crossingi):
                crossLine[crossingi].append(i)
            else:
                crossLine[crossingi] = [i]

        cars.append([carData[0], carData[1],
carData[2:len(carData)]])
        carLine = carFile.readline()
    carFile.close()

    crossMutate = []

    for i in crossLine.keys():
        if len(crossLine[i]) != 1:
            crossMutate.append(i)

    infectCar = []
    for car in cars:
        index = 0
        time = 0
        for street in car[2]:
            time += (streets.loc[street][2]+1)
            if street == 'ebf-ffef':
                infectCar.append([car[0], index, time])
            index += 1

    import networkx as nx
```

```
    newStreets = streets.drop(['ebf-ffef'])
    G = nx.DiGraph()

    for index in range(newStreets.shape[0]):
        # print(index)

G.add_weighted_edges_from([(newStreets.iloc[index]['id1']
, newStreets.iloc[index]['id2'],
newStreets.iloc[index]['length'])])

    info = [[16, 7, 451, 1749], [35, 9, 415, 2337], [50,
4, 415, 1957]]

    f1 = open('newcars.txt', 'w')
    for i in info:
        cari = cars[i[0]-1]
        newTempi = nx.dijkstra_path(G, i[2], i[3])
        tempi = cari[2][0:i[1]]
        for j in range(len(newTempi) - 1):
            tempi.append(num2Street(newTempi[j]) + '-' +
num2Street(newTempi[j + 1]))
        f1.write(cari[0]+','+str(len(tempi))+',')
        for t in tempi:
            f1.write(t+',')
        f1.write('\n')
    f1.close()
```

附录：代码（python）
文件名称：GA.py
文件内容：遗传算法主函数，**全局 main 函数**

```
    from gene import initPops
    from fitness import fitness
    from select import select
    from mutation import mutate
    from crossover import crossover
    from plotFitness import plotFitness
    import pandas as pd
    import numpy as np

    def GA(nIter, nPop, pc, pm):
        pops = initPops(nPop)
        fitList = []
        for i in range(nIter):
            fits = fitness(pops)
```

```
            # print('适应度函数计算完成')
            bestFit = min(fits)
            bestId = fits.tolist().index(bestFit)
            bestPop = pops[bestId]
            pops, fits = select(nPop, pops, fits)
            pops = crossover(pops, pc)
            pops = mutate(pops, pm)
            pops = np.hstack((pops, bestPop))
            print('第'+str(i)+'代已完成，最大值为'+str(-
bestFit))
            fitList.append(-bestFit)
        bestPop.savegene()
        plotFitness(fitList)


    if __name__ == '__main__':
        GA(100, 20, 0.7, 0.3)
```

附录：代码（python）
文件名称：GlobalVar.py
文件内容：全局变量类，记录全局变量

```
    import pandas as pd


    D = 858     # 总时长

    I = 8000    # 路口总数

    S = 63968   # 道路总数

    V = 200     # 车辆总数

    F = 250     # 基本得分


    streets = pd.read_csv('streets.txt', index_col='name')
    # streets = np.array(streets)

    streetsTable = 0

    carFile = open('cars.txt')
    carLine = carFile.readline()
```

```
    cars = []
    crossing = set()
    crossLine = {}
    line = set()

    while carLine:
        carLine = carLine.replace("\n", "")
        carData = carLine.split(',')
        for i in carData[2:len(carData)-1]:
            line.add(i)
            # crossing.add(streets.loc[i][0])
            crossingi = streets.loc[i][1]
            crossing.add(crossingi)
            if crossLine.__contains__(crossingi):
                crossLine[crossingi].append(i)
            else:
                crossLine[crossingi] = [i]

        cars.append([carData[0], carData[1],
carData[2:len(carData)]])
        carLine = carFile.readline()
    carFile.close()

    crossMutate = []

    for i in crossLine.keys():
        if len(crossLine[i]) != 1:
            crossMutate.append(i)

    infectCar = []
    for car in cars:
        index = 0
        for street in car[2]:
            if street == 'ebf-ffef':
                infectCar.append([car[0], index])
            index += 1
```

附录：代码（python）
文件名称：Car.py
文件内容：车辆类，记录车辆的状态转移函数

```
from GlobalVar import streets
import copy
class Car:
    def __init__(self, infor):
```

```
            self.name = infor[0]
            self.streetNum = infor[1]
            self.streetList = infor[2:len(infor)][0]
            self.positionid = 0
            self.street = self.streetList[self.positionid]
            self.pastStreet = -1
            # print(self.streetList)
            self.cross = streets.loc[self.street][1]
            self.changeType = 1
            self.positionType = 2
            self.position = 0

        def move(self, streetsMessage, t):
            if self.positionType == 1:

                # 道路中


                self.position += 1
                if self.position >=
streets.loc[self.street][2]:

                    # 行驶到道路终点

                    # print(self.positionid)
                    if self.positionid ==
len(self.streetList)-1:
                        self.changeType = 0

                        # 行驶到最终的道路终点

                        return 3
                    self.cross = streets.loc[self.street][1]
                    self.positionType = 2
                    self.changeType = 1
                    self.position = 0

                    # 行驶到路口

                    return 2
                else:
                    self.changeType = 0

                    # 行驶在路上

                    return 1

            elif self.positionType == 2:
```

```
                # 路口
                if streetsMessage.judgeMove(self, t):

                    self.position = 0
                    self.pastStreet =
copy.deepcopy(self.street)
                    self.positionType = 1
                    self.positionid += 1
                    self.changeType = 2
                    self.pastcross =
copy.deepcopy(self.cross)
                    self.street =
self.streetList[self.positionid]
                    self.cross = streets.loc[self.street][1]
                    # print(self.name, self.positionType,
self.changeType)
                    # 行驶出路口
                    return 1
                else:
                    self.changeType = 0
                    # 停在路口
                    return 2
            else:
                return 3
```

附录：代码（python）
文件名称：Streets.py
文件内容：道路类，记录道路的信息更新函数

```
from randTime import passStreet
from GlobalVar import crossMutate
import pandas as pd
class Queue:
    def __init__(self):
        self.list = []

    def put(self, item):
        self.list.append(item)

    def get(self):
        temp = self.list[0]
        del(self.list[0])
```

```
            return temp

        def search(self, num):
            if num in self.list:
                return True
            else:
                return False

        def top(self):
            return self.list[0]


    class Streets:
        def __init__(self, gene):
            self.gene = gene
            self.crossingDict = {}
            self.crossingMessage = []

        def changeState(self, cars, t):
            for car in cars:
                if car.changeType == 0:
                    # 汽车的状态没变
                    continue
                elif car.changeType == 1:
                    # print(car.name + '驶入' +
str(car.cross))
                    # 汽车驶入路口
                    if car.cross in crossMutate:

self.crossingMessage.append([car.cross, car.street, t, '驶
入'])
                    if
self.crossingDict.__contains__(car.street):

self.crossingDict[car.street].put(car.name)
                    else:
                        q = Queue()
                        q.put(car.name)
                        self.crossingDict[car.street] = q
```

```
            elif car.changeType == 2:
                # 汽车驶出路口
                carName =
self.crossingDict[car.pastStreet].get()
                car.changeType = 0
                #
print(self.crossingDict[car.pastStreet].list)
                if car.name != carName:

print(self.crossingDict[car.pastStreet].list)
                    print('error')

    def judgeMove(self, car, t):
        # 判断车在 t 时刻是否可以行驶，当车所在的街道可以通行，并
且车排在路口第一位
        # return True
        passstreet =
passStreet(self.gene.chromesome[car.cross], t)
        # print(self.crossingDict)
        if car.pastStreet == -1:
            if car.street == passstreet:
                return True
            else:
                return False
        else:
            if car.street == passstreet and car.name ==
self.crossingDict[car.street].top():
                return True
            else:
                return False
    def saveMessage(self):
        saveData =
pd.DataFrame(data=self.crossingMessage)
        saveData.to_csv('message.csv', encoding='gbk')
```

附录：代码（python）

文件名称：gene.py

文件内容：调度方案基因类，初始化基因

```
from GlobalVar import cars, crossing, line, crossLine,
D, crossMutate
import matplotlib.pyplot as plt
import numpy as np
```

```
    from randTime import randTime, randOneTime
    import random
    import pandas as pd
    from copy import deepcopy
    class gene:
        def __init__(self):
            # 构造函数 —— 基因初始化
            self.chromesome = {}
            for cross in crossLine:
                crossgene = []
                streetList = crossLine[cross]
                if len(streetList) == 1:
                    crossgene.append([streetList[0], D])
                else:
                    timeList = randTime(D, len(streetList))
                    for street, time in zip(streetList,
timeList):
                        crossgene.append([street, time])

                    random.shuffle(crossgene)
                self.chromesome[cross] = crossgene
        def mutate(self):
            # 单个基因的变异操作
            '''
            单个基因的变异操作
            1：单点时间变异
            2：全部时间变异
            3：两点顺序变异
            4：全部顺序变异
            :return:
            '''
            rand = random.random()

            randCross = random.sample(crossMutate, 1)[0]
            if rand < 0.4:
                # 单点时间变异
```

```
                randStreet =
random.sample(range(len(self.chromesome[randCross])),
1)[0]
                # print(randStreet)
                self.chromesome[randCross][randStreet][1] =
randOneTime(D, self.chromesome[randCross],
self.chromesome[randCross][randStreet][1])
            elif rand < 0.5:

                # 全部时间变异

                timeList = randTime(D,
len(self.chromesome[randCross]))
                for i, time in
zip(range(len(self.chromesome[randCross])), timeList):
                    self.chromesome[randCross][i][1] = time
            elif rand < 0.9:

                # 两点顺序变异

                if len(self.chromesome[randCross]) >= 2:
                    i, j =
random.sample(range(len(self.chromesome[randCross])), 2)
                    self.chromesome[randCross][i],
self.chromesome[randCross][j] =
self.chromesome[randCross][j],
self.chromesome[randCross][i]
            else:

                # 全部顺序变异

                random.shuffle(self.chromesome[randCross])


        def savegene(self):
            f1 = open('result.txt', 'w')
            f1.write(str(len(self.chromesome)) + '\n')
            geneList = []
            for i in self.chromesome.keys():
                f1.write(str(i)+'\n')
                f1.write(str(len(self.chromesome[i]))+'\n')
                for j in self.chromesome[i]:
                    if i ==
list(self.chromesome.keys())[len(self.chromesome.keys())
- 1] and j == self.chromesome[i][len(self.chromesome[i])-
1]:
                        f1.write(str(j[0]) + ',' + str(j[1]))
                    else:
```

```
f1.write(str(j[0])+','+str(j[1])+'\n')
            f1.close()


    class gene2:
        def __init__(self):
            # 构造函数 —— 基因初始化
            self.chromesome = {}
            for cross in crossLine:
                crossgene = []
                streetList = crossLine[cross]
                if len(streetList) == 1:
                    crossgene.append([streetList[0], D])
                else:
                    timeList = [random.randint(1, 5) for i in
range(len(streetList))]
                    for street, time in zip(streetList,
timeList):
                        crossgene.append([street, time])

                    random.shuffle(crossgene)
                self.chromesome[cross] = crossgene

        def savegene(self):
            f1 = open('result2.txt', 'w')
            f1.write(str(len(self.chromesome)) + '\n')
            geneList = []
            for i in self.chromesome.keys():
                f1.write(str(i)+'\n')
                f1.write(str(len(self.chromesome[i]))+'\n')
                for j in self.chromesome[i]:
                    if i ==
list(self.chromesome.keys())[len(self.chromesome.keys())
- 1] and j == self.chromesome[i][len(self.chromesome[i])-
1]:
                        f1.write(str(j[0]) + ',' + str(j[1]))
                    else:

f1.write(str(j[0])+','+str(j[1])+'\n')
            f1.close()

    def initPops(nPop):
```

```
        # 初始化种群
        pops = []
        for i in range(nPop):
            pops.append(gene())
        return np.array(pops)
```

附录：代码（python）
文件名称：randTime.py
文件内容：工具类，随机生成每个道路的绿灯通行时间，求解当前路口的绿灯的隧道

```
    import random


    def randTime(maxValue, num):
        '''生成总和小于某数的整数序列

        maxValue: 序列总和

        num: 要生成的整数个数

        return
        per_all_persons:list,指定 num 个接种点各自待接种的人数
        '''
        maxValue = random.randint(num, maxValue)
        suiji_ser = random.sample(range(1, maxValue), k=num
- 1)  # 在 1~maxValue 之间，采集 20 个数据

        suiji_ser.append(0)  # 加上数据开头
        suiji_ser.append(maxValue)
        suiji_ser = sorted(suiji_ser)
        per_all_persons = [random.randint(1,4) for i in
range(1, len(suiji_ser))]  # 列表推导式，计算列表中每两个数之间的间隔
        return per_all_persons

    def randOneTime(maxValue, crossAssign, time):
        '''
        返回随机时间
        :param maxValue:
```

```
    :param timeList:
    :param time:
    :return:
    '''
    # print(crossAssign)
    timeList = [i[1] for i in crossAssign]
    maxValue -= (sum(timeList) - time)
    return random.randint(1, maxValue)


def passStreet(cross, time):
    crossTimeList = [i[1] for i in cross]
    T = sum(crossTimeList)
    t = time % T
    streetTime = cross[0][1]
    for i in range(1, len(cross)):
        if t < streetTime:
            return cross[i-1][0]
        else:
            streetTime += cross[i][1]
    return cross[len(cross) - 1][0]
```

附录：代码（python）
文件名称：fitness.py
文件内容：仿真模型类，计算得到适应度函数

```
from GlobalVar import D, cars, F
from Streets import Streets
from Car import Car
import numpy as np
from gene import initPops

def simulation(g):
    # 基因的适应度函数计算
    Cars = []
    for car in cars:
        Cars.append(Car(car))
    streets = Streets(g)
    M = 0
    for time in range(D):
        # print(time)
        if Cars == []:
            break
```

```
            streets.changeState(Cars, time)
            for car in Cars:
                if car.move(streets, time) == 3:
                    M += (F + D - time)
                    # print(car.name, time)
                    Cars.remove(car)
        # streets.saveMessage()
        return M

def fitness(pops):
    fit = []
    for genei in pops:
        # print(genei)
        fit.append(- simulation(genei))
return np.array(fit)
```

附录：代码（python）
文件名称：select.py
文件内容：遗传算法选择函数

```
import numpy as np
import random


def select(pool, pops, fits):
    nPop = len(pops)
    newPops = [0] * pool
    newFits = [0] * pool
    indices = np.arange(nPop).tolist()

    i = 0
    while i < pool:
        idx1, idx2 = random.sample(indices, 2)
        idx = compare(idx1, idx2, fits)
        newPops[i] = pops[idx]
        newFits[i] = fits[idx]
        i += 1
    return np.array(newPops), np.array(newFits)


def compare(idx1, idx2, fits):
    if fits[idx1] <= fits[idx2]:
        return idx1
    else:
        return idx2
```

附录：代码（python）
文件名称：crossover.py
文件内容：遗传算法交叉函数

```
import copy
import numpy as np
import random
from gene import gene
def crossover(pops, pc):
    chrPops = copy.deepcopy(pops)
    nPop = len(chrPops)
    for i in range(0, nPop, 2):
        if np.random.rand() < pc:
            SBX(chrPops[i], chrPops[i+1])
    return chrPops

def SBX(chr1, chr2):

    # 路口交叉

    pos1, pos2 = np.sort(np.random.randint(0,
len(chr1.chromesome), 2))
    pos2 += 1
    keys = list(chr1.chromesome.keys())
    for key in keys[pos1:pos2]:
        chr1.chromesome[key], chr2.chromesome[key] =
chr2.chromesome[key], chr1.chromesome[key]
```

附录：代码（python）
文件名称：mutation.py
文件内容：遗传算法变异函数

```
import numpy as np


def mutate(pops, pm):
    nPop = len(pops)
    for i in range(nPop):
        if np.random.rand() < pm:
            pops[i].mutate()
    return pops
```

附录：代码（python）
文件名称：Dijkstra.py
文件内容：计算发生拥堵后的车辆

```python
import pandas as pd
def num2Street(num):
    str = ''
    while num != 0:

        temp = num % 10
        # print(temp)
        num = int(num/10)
        str = chr(temp + 97) + str
    return str


D = 858     # 总时长

I = 8000    # 路口总数

S = 63968   # 道路总数

V = 200     # 车辆总数

F = 250     # 基本得分

streets = pd.read_csv('streets.txt', index_col='name')
# streets = np.array(streets)

streetsTable = 0

carFile = open('cars.txt')
carLine = carFile.readline()
cars = []
crossing = set()
crossLine = {}
line = set()

while carLine:
    carLine = carLine.replace("\n", "")
    carData = carLine.split(',')
    for i in carData[2:len(carData)-1]:
        line.add(i)
        # crossing.add(streets.loc[i][0])
        crossingi = streets.loc[i][1]
        crossing.add(crossingi)
        if crossLine.__contains__(crossingi):
```

```
                crossLine[crossingi].append(i)
            else:
                crossLine[crossingi] = [i]

        cars.append([carData[0], carData[1],
carData[2:len(carData)]])
        carLine = carFile.readline()
    carFile.close()

    crossMutate = []

    for i in crossLine.keys():
        if len(crossLine[i]) != 1:
            crossMutate.append(i)

    infectCar = []
    for car in cars:
        index = 0
        time = 0
        for street in car[2]:
            time += (streets.loc[street][2]+1)
            if street == 'ebf-ffef':
                infectCar.append([car[0], index, time])
            index += 1

    import networkx as nx
    newStreets = streets.drop(['ebf-ffef'])
    G = nx.DiGraph()

    for index in range(newStreets.shape[0]):
        # print(index)

G.add_weighted_edges_from([(newStreets.iloc[index]['id1']
, newStreets.iloc[index]['id2'],
newStreets.iloc[index]['length'])])

    info = [[16, 7, 451, 1749], [35, 9, 415, 2337], [50,
4, 415, 1957]]

    f1 = open('newcars.txt', 'w')
    for i in info:
        cari = cars[i[0]-1]
        newTempi = nx.dijkstra_path(G, i[2], i[3])
        tempi = cari[2][0:i[1]]
```

```
        for j in range(len(newTempi) - 1):
            tempi.append(num2Street(newTempi[j]) + '-' +
num2Street(newTempi[j + 1]))
        f1.write(cari[0]+','+str(len(tempi))+',')
        for t in tempi:
            f1.write(t+',')
        f1.write('\n')
    f1.close()
```