基于路况信息实时采集系统的红绿灯设计

摘 要

本文首先介绍了路况信息实时采集系统，通过该系统能够获取车辆位置信息，这是通过仿真模拟设计红绿灯的现实基础。接着采用数学建模的方法，用逐步分析的方法分析出各个方向的左直右车道的车辆在经过十字路口时的交错关系并据此选择了一种红绿灯轮换规则作为基础建立了道路模型。

接着通过程序设计，实现了仿真模拟传统红绿灯道路模型下的汽车行驶过程，记录下对道路交通的 100 次模拟时长，得出 720 辆车通过道路模型的平均用时为 459.2s。然后采用控制变量法，在保持车辆相同的初始数量和行车规则下，建立参数控制的红绿灯道路模型，通过改变统计车辆距离停止线范围得到不同的模拟时长，结果发现搜寻到的极小值点对应的平均模拟时长总大于 600s，也大于 459.2s，这表明轮换式的变灯规则本身就有较强合理性，通行效率优于打分式绿灯选择。

在后文中采用控制变量法，通过对 10s 到 34s 不同绿灯单次持续时长的 100 次模拟求平均值的方式求出模拟用时最短的绿灯单次持续时长。结论为该模型最佳绿灯持续时长为 11s，其对应的平均模拟时长为 416.89s，比同车辆初始条件下的 27s 绿灯持续时长的平均模拟结果 434.51s 节省 4.4% 的时间，这表明绿灯单次持续时长是影响通行效率的因素之一。

此外还通过加装倒计时器的的方法，分析出车辆的刹车起始点距离从 100m 降为 85m，同时将绿灯单次持续时长设置为 11s，经过 100 次仿真模拟得到的平均模拟时长为 410.31s，比不加装倒计时器，绿灯单次持续时长为 27s 的道路仿真模拟的通车时间减少了 10.65% 的时间，表明加装倒计时器可以有效提高车辆通行效率。

在正文的最后一部分，总结了建模的思路，过程以及结果，分析了建模的优点和缺点，并且提出了一些改进措施。

关键词：红绿灯设计，数学建模，仿真模拟，控制变量法，定步长搜索法

Abstract

This paper firstly introduces the real-time gathering system of traffic information, through which the position of vehicles can be collected. This is the realistic basis of traffic light design through analogue simulation. Then the method of mathematical modeling is used to analyze the staggered relationship of the vehicles in the left and right lanes in each direction when passing through the intersection with the method of step by step analysis. Based on this, a traffic light rotation rule is set up as the basis to establish the road model.

Then, through the program design, the vehicle driving process under the traditional traffic light road model was simulated, and 100 times of road traffic simulation time was recorded, and the average time of 720 vehicles passing through the road model was 459.2s. Then, control variable method was adopted to establish a traffic light road model controlled by parameters under the condition of maintaining the same initial number of vehicles and driving rules. Different simulation duration was obtained by changing the range of statistical vehicles' distance from the stop line. The results showed that the average simulation duration corresponding to the minimum point searched was greater than 600s and 459.2s respectively. This shows that the rule of rotating change light itself has strong rationality whose traffic efficiency is better than the fractional green choice rule's.

In the following paper, the control variable method is adopted to calculate the single duration of green light with the shortest simulation time by averaging 100 times of the single duration of different green lights from 10s to 34s. The conclusion is that the optimal green duration of this model is 11s, and the corresponding average simulation time is 416.89s, which saves 4.4% time compared with the average simulation result of green duration of 27s under the same vehicle initial condition of 434.51s, which indicates that the single green duration is one of the factors affecting the traffic efficiency.

In addition, by adding a rewind timer, it is analyzed that the distance of the starting point of the vehicle brake is reduced from 100m to 85m, and the single duration of the green light is set to 11s. After 100 simulations, the average simulation time is 410.31s, which is better than that without a rewind timer. The traffic time of the road simulation with a single green light lasting for 27s is reduced by 10.65%, which indicates that adding a countdown timer can effectively improve the traffic efficiency.

In the last part of the text, it summarizes the idea, process and result of modeling, analyzes the advantages and disadvantages of modeling, and puts forward some improvement measures.

KEY WORDS: Traffic light design;Mathematical Modeling;analogue simulation;control variable method;Fixed-step search method

目 录

1 引言

现代的互联网能够将信息以光速传播到世界各地，它对于方便全球人类沟通起到了重要的作用；而现代交通系统能够使人们短时间内到达世界各地，它是物质实体全球移动的重要基础。互联网与现代交通运输系统共同使地球村成为了现实。

虽然现代交通运输系统的运输速度永远达不到信息的光速传播那么快，但是它的运输效率还有巨大的可优化空间，道路交通作为交通运输系统中的一环对于交通运输系统的效率起着举足轻重的作用，如果能够有效提高道路交通运输效率，那么这对于整个交通运输系统的效率提升也能起到促进作用。

本文在市区经常发生堵车的背景下以及路况信息实时采集系统的理论依据下，采用数学建模的方法建立了一个正方形的数字化市区，并且用控制变量法和定步长搜索法使用 Python 对不同红绿灯设计的道路的通行效率进行仿真模拟，目的是通过改变红绿灯设计然后仿真模拟得到不同的模拟数据，并据此得出最优的智能红绿灯设计。

预期结果是使用智能红绿灯道路模型的通车效率优于传统红绿灯道路模型的通车效率。作用是通过仿真模拟证明了通过改变红绿灯设计提高通车效率的可行性并且给出具体方案，此外模拟程序还可以对其他红绿灯研究提供重要参考。

建模设想与突破点：作为智慧道路不应该仅在两个时期改变红绿灯策略，每个时刻都会根据实时道路信息采用最优变灯策略；同时应该安装倒计时器用来防止由于采用不同策略导致司机无法评估绿灯时长而出现意外闯红灯。

理论和实践意义：通过使用优化设计的红绿灯能够提升运输效率，有利于打造一个富有活力的，运输高效的交通大国；使用优化设计的红绿灯能够减少车辆单位距离的碳排放量，有助于打造绿色的生态环境，也有助于实现碳中和目标。

文献综述：论文基于移动终端的交通路况信息实时采集与显示系统和期刊基于移动终端的路况信息实时采集与显示系统都介绍了一种基于移动终端的实时路况信息采集和显示系统，该系统采集移动终端的速度和方向信息，并将其转换为道路上的浮动车流量，据此划分交通拥堵状况。服务器端以 MySQL 为数据库，采用 Servlet 技术开发，客户端采用 jQuery Mobile + Html5 技术设计为 Web App，以适应不同的移动设备。交通路况的显示借助于百度地图，数据更新采用 Ajax 技术。该系统具有成本低、易维护的优点，可以提供实时路

况数据。文章详细介绍了 Ajax 技术、jQuery Mobile + Html5 设计、百度地图 API 等关键技术的应用，以及系统的组成和工作原理，数据采集和处理，系统的设计和实现等方面。测试结果表明，在 4G 网络及移动智能终端普及的今天，该系统获取实时路况信息成本低、准确度高，可缓解交通拥堵压力，同时也方便人们出行[1, 2]。

基于有限差分法与变步长搜索法的炉温曲线设计主要研究了回流焊工作过程中的热量传递问题和炉温曲线的优化问题。首先对回焊炉内的热量进行分析，得出了炉内环境空气温度变化规律，并建立了 PCB 板中的热量传递模型。文章结合参数随温度变化的影响将整个 PCB 板运动过程分为 5 个阶段，并根据实验数据对多个传导影响因子、对流辐射叠加影响因子进行搜索求解，最终确定最大过炉速度以及设计理想的炉温曲线。文章详细介绍了模型建立过程，包括建立各个温区的数学模型、运用有限差分法求解焊接区域中心的温度变化情况和运用变步长搜索算法确定最优的炉温曲线。这项研究可为集成电路板等电子产品的生产提供参考依据，提高产品的质量。[3]

基于“卡口”测速数据的城市道路限速值合理性论证研究一文结合道路交通治安卡口监控系统的测速数据进行限速值合理性论证。最终得出以车辆 85% 速度调整后的速度值作为限速值的结论，为重庆交通道路限速设计提供了理论依据。[4]

基于强化学习和计算机仿真的交通信号调度一文主要研究如何利用强化学习和计算机模拟技术控制交通信号，缓解城市交通拥堵问题。该研究使用了 A3C 算法来控制不同场景下的交通信号，包括单个和多个路口的恒定和变化的车流率。该研究还重新定义了状态空间，行动空间，奖励函数和评估指标，以优化 A3C 算法。通过模拟结果可以发现，该算法性能优于其他方法，可以减少车辆等待时间，从而缓解城市交通繁忙的情况。文章还提供了强化学习，Q-Learning 和策略梯度算法的综述及其在交通信号控制方面的应用。该研究最终得出结论，A3C 算法具有高效性和卓越性能，可以减轻交通信号安排的压力，缓解城市道路交通拥堵问题。[5]

2 路况信息实时采集系统简介

路况信息实时采集系统是获取道路信息的基础也是仿真模拟的基础。路况信息实时采集系统使用了 Ajax，jQuery Mobile 和 HTML5 技术，被设计成 Web App 的形式，它能够调用 HTML5 里的 Geolocation API 获取使用者的地

理位置信息。路况信息实时采集系统基于 Tomcat 服务器，MySQL 数据库，无线通信等服务把位置信息存储，加工，然后将其与从其他用户收集到的地理信息进行汇总分析，得出道路的实时情况，最终再通过调用百度地图的 API，把分析出的路况信息在地图上进行展示。

其中 Ajax 指的是一种网页开发技术，它用于创建交互式应用；jQuery Mobile 是一款 JavaScript 类的库框架，它是开源的，能够给不同平台提供统一的 UI 框架；Web App 指的是使用网页技术开发出的在浏览器上运行的 App。[1, 2]

百度 API 是一套基于百度地图服务的应用程序接口，包括 Web 服务 API，Android SDK、iOS SDK、定位 SDK、JavaScript API 等多种开发工具和服务。使用百度地图 API 开发地图服务功能时，一般要包括 Web 服务器、百度地图服务器、数据库、客户端等。JSON 是轻量级的数据交换格式，它能够与 Java 基本类型相互解析。XML 是另一种主流数据交换语言，可以标记电子文档。JSON 与 XML 相比，具有节约手机计算资源，减小网络数据传输时长，提高传输速度的优点。

此外该系统有低成本、维护方便的优点。经过测试，在 4G 网络和移动终端普及的基础上，该系统获取实时路况信息的不仅成本低而且准确度高。使用该系统一方面能够缓解交通拥堵压力，另一方面也能方便人们选择出行路径。

通过路况信息实时采集系统就可以得到当前道路车辆的位置，速度信息，在此基础上仿真模拟的现实依据已经充分。

3 数学建模

3.1 传统红绿灯道路模型

非智能红绿灯道路模型的红绿灯变化只跟时间有关，变化规律是固定的。具体变化规律在基本假设的基础上由逐步分析得到，接着进行程序模拟得到相关数据。

3.1.1 模型假设

1. 假设在距离路口停止线超过 100 米时，车辆在保证适当车距的前提下尽可能以最高限速行驶。

2. 假设区域道路限速为 $17m/s$。

3. 车辆近似为质点。

4. 假设车辆在距离停止线 $100m$ 时开始减速。

5. 假设所有车辆不会超车。

6. 假设行人不对车辆右转产生影响。

7. 假设车辆可以左转或直行时，斑马线上已经没有行人。

8. 假设每个路口的红绿灯按照同一种固定程序运作不受车辆影响。

9. 假设汽车变道的时间可以忽略。

10. 假设车辆在进行转弯时的路径为标准的四分之一圆弧。

11. 假设每个路口在相同方向的红绿灯是同步的且按照顺时针方向变换出新的绿灯。

12. 假设车辆经过路口时，按照原速无法通过路口，它会降速等红灯而不是提速通过绿灯。

13. 假设距离路口停止线 $20m$ 以内时，$6m/s$ 为理想速度。

14. 假设加速度绝对值上限为 $2m/s^2$。

3.1.2 符号说明

符号	含义	单位
r_1	左转半径	m
r_2	右转半径	m
w_r	路口宽度	m
t_1	绿灯单次持续时长	s
t_2	黄灯单次持续时长	s
T	红绿灯变化周期	s
v	车速	$m \cdot s^{-1}$
a	车加速度	$m \cdot s^{-2}$
Δt	数据更新的单位时间	m
d_c	与前车距离差	m

d_v	与前车速度差	$m \cdot s^{-1}$
d_s	车辆距离停止线距离	m
d_l	车辆经过路口时需要行进的路程	m
a_0	加速度绝对值上限	$m \cdot s^{-2}$
v_l	道路最高限速	$m \cdot s^{-1}$
v_c	停止线 20m 内限速	$m \cdot s^{-1}$

3.1.3 道路模型

首先建立一个道路模型用来模拟市区道路，这条虚拟道路的构想如下：它在一个 $4944m \times 4944m$ 的大正方形区域中，在这片大正方形区域中包含了正方形的住宅区，它们有 4×4 共计 16 个，每一个住宅区的大小为 $1200m \times 1200m$，它们均匀地分布在大正方形区域中，除去这些正方形的住宅区域，在大正方形区域中剩下的部分就是汽车通行道路，将位于同一条直线上的汽车通行道路视作一个整体，那么一共有 6 条汽车通行道路，其中有三条东西方向的汽车通行道路，还有三条南北方向的汽车通行道路，这些道路一共有 3×3 共计 9 个交叉部分，这 9 个交叉部分就是十字路口。

由于每条汽车通行道路宽为 $48m$，所以这 9 个交叉路口的每一个都是一个 $48m \times 48m$ 的正方形区域。此外，每个汽车通行道路都有两个通行方向，每个通行方向的总宽为 $24m$，这两个通行方向所指向的单位向量夹角为 180°，如东西方向的汽车通行道路有两个行车方向，其中一个方向是由西向东，另一个方向则是由东向西。由于道路模型是基于中华人民共和国的道路规则所进行的模拟设计，所以在每个车辆以当前行驶方向为前方的视角下，与之行驶方向相反的车辆位于其左侧。

每个汽车通行道路都有两个通行方向，而每个通行方向又有三个车道，分别是直行车道，左转车道和右转车道，这些不同车道的车辆在遇到下一个十字路口时会驶入不同的方向，直行车道的车辆会在直行灯为绿灯时直行，左转车道的车辆会在左转灯为绿灯时左转，右转车道的车辆会在右转灯为绿灯时右转。

从在整体来看，大正方形区域有 9 个十字路口，在大正方形区域的最外侧，每条边上都有三个连接外部的道路进出口，即在每个边上都有 3 个进口和 3 个出口，这三个进口和三个出口是一一对应的，一条汽车通行道路在一个边就会产生一对进出口，这对进出口的行驶方向是相反的，驶入正方形区域的

是进口，驶出正方形区域的是出口。可以通过 4×3 得出整个大正方形区域共有 12 个进口和 12 个出口。

图 1 为该道路模型的图示，黑色区域为住宅区，白色区域为道路，用黄色边线围起来的正方形区域是十字路口。

接下来是对十字路口的模型建立，由于十字路口的通行时机受到红绿灯的调控，所以应当首先分析出在模型假设下的红绿灯的基本特点。可以通过逐步分析和分类讨论的数学方法进行分析。

首先在一个十字路口有来自四个方向，每个方向有三个车道共 4×3 为 12 种行驶路线。将这些路线在十字路口的运动轨迹用直线或曲线记录下来就可以得到一个十字路口车辆路线图，通过这些线路的图像以及行驶路线发生交错的车辆不能同时发车以避免发生车辆碰撞的基本常识就可以逐步推断出红绿灯的基本规则, 如图 2 所示。

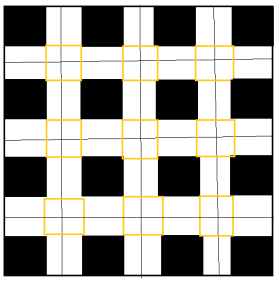

图 1: 大正方形交通区域

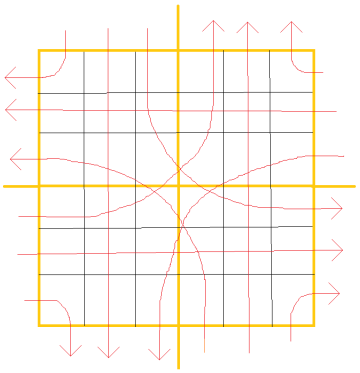

图 2: 十字路口交通路线图

接着从十字路口车辆路线图可以看出在路口进行右转的车辆不与其他任何路线产生交叉，所以可以首先规定在该模拟道路区域内的十字路口，右转方向的红绿灯始终保持为绿色可通行状态，接下来则只需分析直行灯和左转灯的关系。然后从直行方向的交通灯入手，直行灯总有状态为绿灯的时候，可以首先将直行灯确定为绿灯状态，在此基础上进行进一步分析。以从南向北的直行灯为绿灯为例，此时由东向西的左转路线，东西方向的直行路线，由北向南的左转路线都与从南向北的直行路线有交叉，所以可以确定出 $1 + 2 + 1$ 共 4 个交通灯为红灯。直行与左转的交通灯一共有 8 个，除去 1 个作为分析基础的绿灯和 4 个分析出的红灯，只剩下三个交通灯待确定，为了提高交通效率，剩下的交通灯是绿灯的状态越多越好。这三个交通灯分别是从北向南的直行灯，从南向北左转灯和由西向东的左转路线。这三个交通灯所对应的交通路线都与从南向北的直行灯对应的交通路线没有交叉，但是它们却相互交

叉，所以这三个交通灯只能有一个绿灯。

若从北向南的直行灯为绿灯，则从南向北左转灯和由西向东的左转灯为红灯，见图 3，这时由于结论中缺少左转灯为绿灯的情况，所以还需进行一次在左转灯为绿灯的基础上进行的交通灯状态分析。将南向北左转灯为绿灯作为基础，再次运用上述的逐步分析法可以得到结论，只有从南向北直行灯或由东向西的直行灯为绿灯或从北向南的左转灯为绿灯，其它左转和直行的交通灯都为红灯。由于从南向北直行灯为绿灯和由东向西的直行灯为绿灯时的情形与上一层讨论的第二种情形或第三种相同，所以选择从北向南的左转灯为绿灯，如图 4 所示。此时可得出第一种情况的交通灯规则为的直行灯和左转灯轮换变为绿灯。

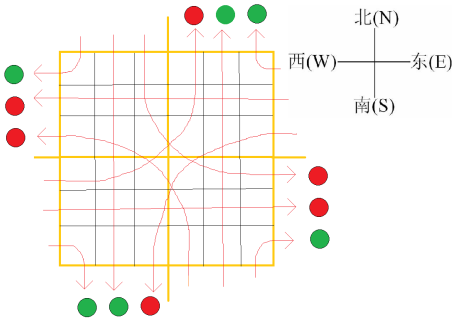

图 3: 南北直行绿灯

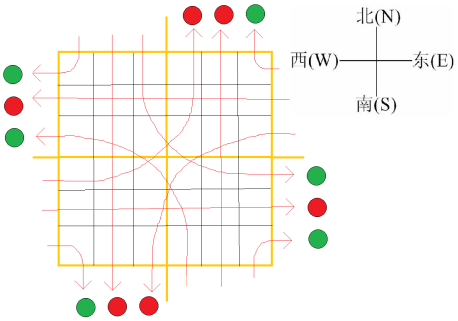

图 4: 南北左转绿灯

若从南向北的左转灯或由西向东的左转灯为绿灯，则同时出现了直行和左转为绿灯的情况，通过交通四个方向的交通灯的轮流变换就可以实现每条车道的车辆通行，见图 5、6。

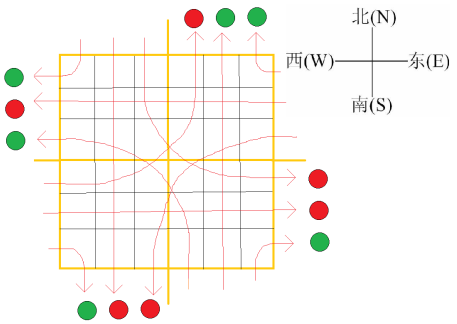

图 5: 南北直行与左转绿灯

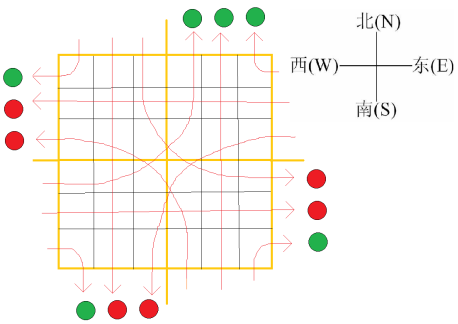

图 6: 南北直行与由西向东绿灯

对同一直行灯这三种变灯方式都需进行 4 次轮换变回绿灯，所以采取任何一种都是等效的，程序采用第二种情形即图 5 所示。

不排除直行灯和左转灯变灯时有时间差出现情形一、情形二与情形三交替产生的情况，事实上当红绿灯直行灯和左转灯变灯的时间差为 0 时，即为

情形二或情形三中的一种，而情形二与情形三在通行效率上等效，所以采用情形二的规则结论不失一般性。

3.1.4 车辆模型

整个模型运作分为四大步骤。

Step1: 定义车类，车类包含 11 个属性，它们分别是方向 $direction$，坐标 $coordinate$，车道 $serial$，距离停止线的距离 $disstopline$，经过十字路口的行驶路程 $dislimitation$，红绿灯颜色 $light$，方向集合 lis，速度 $speed$，加速度 $acceleration$，与前方车辆距离 d_c，与前方车辆速度差 d_v。d_c，d_v，$speed$，$acceleration$ 的默认值为 $T = 0$ 时刻的数值，分别为 50，0，17，0。

属性的作用与关系：其中 $direction$，$coordinate$，$serial$，$disstopline$ 属性起到了定位的作用，lis 属性起到确定行进路线的作用，$speed$ 主要起更新定位的作用，$acceleration$ 决定这 $speed$ 在下一时刻的大小，而 d_v，d_c，$disstopline$，$light$ 又影响着 $acceleration$ 的大小。

本文以大正方形区域的中心为原点，正东为 x 轴正方向，正北为 y 轴正方向建立平面直角坐标系，在此基础上 $direction$ 有四种方向分别是 $[1,0]$，$[-1,0]$，$[0,-1]$，$[0,1]$。

$coordinate$ 则是在 $direction$ 确定的基础上，分为东西和南北两套编号，由西向东道路和由东向西道路在坐标系下的位置相对一致，由南向北道路和由北向南道路在坐标系下的位置相对一致。由此确定对 $direction = [1,0]$ 或 $direction = [-1,0]$ 的道路，$coordinate$ 的横坐标取值范围是 $[-2,-1,1,2]$，纵坐标取值范围是 $[-1,0,1]$；对 $direction = [0,1]$ 或 $direction = [0,-1]$ 的道路，$coordinate$ 的横坐标取值范围是 $[-1,0,1]$，纵坐标取值范围是 $[-2,-1,1,2]$。图 8 中橙红色部分为 $T = 0$ 时刻车辆位置，图 9 到图 12 中标蓝的部分即为相应方向的道路。

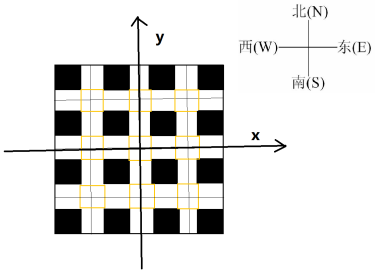

图 7: 建立直角坐标系

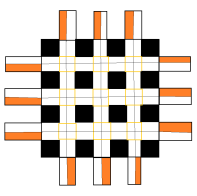

图 8: 车辆初始位置示意图

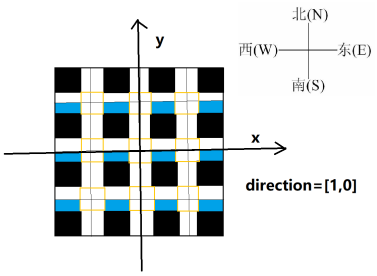

图 9: 由西向东道路

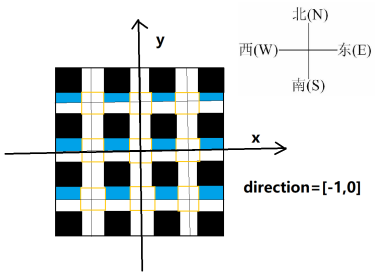

图 10: 由东向西道路

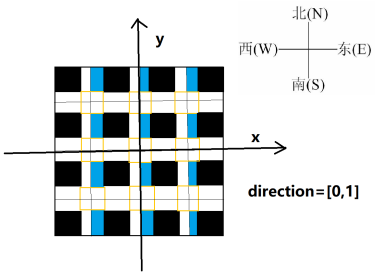

图 11: 由南向北道路

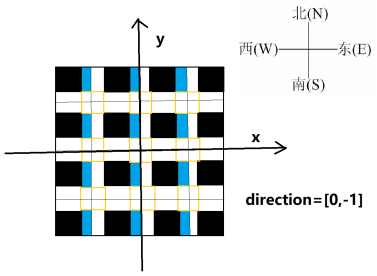

图 12: 由北向南道路

Step2: 每个路口产生 60 辆车辆，每个车辆有两个确定的属性：$direction$ 和 $coordinate$。

Step3: 编写初始化文件 $initialize.py$，文件包含主函数 $initialize$ 和其它在主函数中被调用的函数，主函数包含一个参数 $self$，对于传入参数的车辆，会对车的其它属性进行初始化，首先会在 $makelis$ 中调用 $random$ 函数，在与入口成对的出口之外的 11 个出口中选择一个作为目标出口；接着编写一个 $listmaker$ 函数，根据入口和出口确定 lis 属性，在 $makelis$ 中调用 $listmaker$ 即可确定车辆行驶路线；然后根据 lis 的第一个元素值确定 $serial$，根据 $serial$，确定 $dislimitation$，再根据 $direction$ 和 $serial$，调用 $makelight$ 函数确定 $T = 0$ 时的红绿灯颜色。最后在定义车类时，d_c，d_v，$speed$，$acceleration$ 的默认值为 $T = 0$ 时刻的数值，分别为 50，0，17，0，不需要在初始化文件的函数中对其定值。$disstopline$ 属性需要在运行初始化函数后结合 $serial$ 单独定值。

Step4: 编写更新文件 $update.py$，首先由上一时刻 $acceleration$ 更新 $speed$，然后更新 $disstopline$，如果 $disstopline > 0$，则只需要更新 $light$，否则先更新 lis，再判断 lis 是否为空，若 lis 为空则可计算时长，若 lis 不为空，再依次更新 $coordinate$，$direction$，$serial$，$disstopline$，$dislimitation$，$light$；然后判断 $cars$ 是否为空，若为空则结束；否则接着对首轮更新的 $cars$ 进行重组，将同一车道的车辆按照 $disstopline$ 的大小从小到大排序；最后更新 d_c,d_v,$acceleration$。

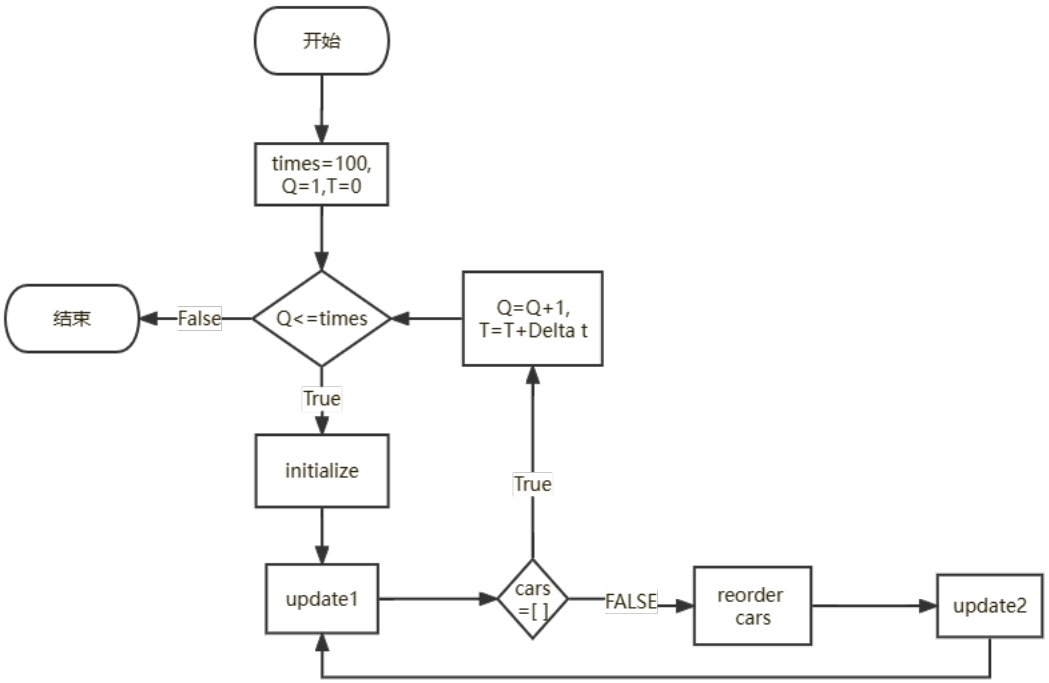

图 13: 传统红绿灯道路模拟

3.1.5 数值设定与计算

对于左转的车辆，$d_l = r_1 \cdot \frac{2\pi}{4} = 14\pi$
对于直行的车辆，$d_l = w_r = 48$
对于右转的车辆，$d_l = r_2 \cdot \frac{2\pi}{4} = 2\pi$
红绿灯一轮周期 $T = (t_1 + t_2) \cdot 4 = 120$
加速度绝对值上限 $a_0 = 2$
道路最高限速 $v_l = 17$[4, 6]
距离停止线 $20m$ 以内限速 $v_c = 6$

3.1.6 加速度更新公式

一、对于当前车道第一辆车：
1. 当 $d_s > 100$ 时：

$$a = \begin{cases} \frac{v_l - v}{\Delta t} & , \quad v_l - v < a_0 \Delta t \\ a_0 & , \quad v_l - v \geq a_0 \Delta t \end{cases} \tag{3-1}$$

2. 当 $20 \leq d_s \leq 100$ 时：

$$a = \begin{cases} \frac{v_c - v}{\Delta t} & , \ |v_c - v| \leq a_0 \Delta t \\ a_0 \cdot sgn(v_c - v) & , \ |v_c - v| > a_0 \Delta t \end{cases} \tag{3-2}$$

3. 当 $0 < d_s < 20$ 且为绿灯时：

$$a = \begin{cases} \frac{v_c - v}{\Delta t} & , \ |v_c - v| \leq a_0 \Delta t \\ a_0 \cdot sgn(v_c - v) & , \ |v_c - v| > a_0 \Delta t \end{cases} \tag{3-3}$$

4. 当 $0 < d_s < 20$ 且为黄灯或红灯时：

$$a = \begin{cases} -\frac{v}{\Delta t} & , \ v \leq a_0 \Delta t \\ -a_0 & , \ v > a_0 \Delta t \end{cases} \tag{3-4}$$

公式内涵：在距离停止线超过 $100m$ 车辆尽可能加速到道路最大限速 v_l 行驶且加速度不超过加速度绝对值上限 a_0；距离停止线 $20m$ 到 $100m$ 范围内时，车辆逐渐减速到距离停止线 $20m$ 以内的限速 v_c，且减速加速度绝对值不超过加速度绝对值上限 a_0；距离停止线 $20m$ 以内时，观察红绿灯状态，如果是绿灯，就通过加减速以速度 v_c 通过，且加速度绝对值不超过加速度绝对值上限 a_0，如果是黄灯或红灯，就减速到 0，且加速度绝对值不超过加速度绝对值上限 a_0。[7, 8]

二、对于车道上的非首辆车：

1. 当 $(d_c - 3v) \cdot d_v < 0$ 时

$$a = \begin{cases} -\frac{d_v}{\Delta t} & , \ |d_v| < 2\Delta t \\ -a_0 \cdot sgn(d_v) & , \ |d_v| \geq 2\Delta t \end{cases} \tag{3-5}$$

2. 当 $(d_c - 3v) \cdot d_v \geq 0$ 时

$$a = 0 \tag{3-6}$$

公式内涵：将三倍车速定义为标准车距，如果速度差和距离差异号，则在加速度绝对值不超过加速度绝对值上限 a_0 的前提下通过加速或减速逐渐恢复标准车距。如果速度差和距离差同号或有一个为 0，则不需要改变速度，即加速度为 0。

3.1.7 程序模拟

表 2: 传统红绿灯道路模型的 100 次模拟时长记录表

483.0582353	419.3523529	460.0582353	461.5288235
494.5311765	462.4135294	382.4723529	494.0605882
464.4111765	530.5311765	437.1758824	483.1170588
457.3547059	303.3970588	418.1170588	464.0582353
483.1170588	513.0582353	543.0582353	573.0605882
557.1170588	459.1905882	488.5311765	340.4111765
453.0605882	573.0582353	483.1170588	453.1170588
423.1170588	460.2935294	513.0605882	506.5288235
464.0582353	506.0605882	391.3523529	459.5288235
480.3547059	464.0582353	513.0582353	573.0582353
370.3523529	418.0582353	470.4723529	431.8229412
433.1170588	513.0582353	513.0582353	461.5288235
514.1194118	430.0582353	467.1170588	460.4135294
432.4723529	462.0605882	458.6464706	492.5311765
453.1170588	453.0582353	391.1170588	461.0605882
421.1758824	372.0441176	460.0582353	483.1170588
431.9429412	386.0582353	393.0582353	543.0582353
381.3523529	461.5288235	363.0605882	397.4135294
453.1170588	420.1170588	483.1170588	454.0605882
513.1194118	461.0605882	464.0582353	453.1170588
413.3523529	543.0582353	520.3547059	457.1758824
466.1170588	418.1170588	453.1170588	461.5288235
463.5288235	380.4723529	466.6464706	443.3523529
483.0582353	462.0605882	423.1170588	462.5288235
462.4135294	460.3547059	423.1170588	483.0582353

表 2 为 100 次模拟的时长记录，模拟时长的平均值为 $459.20s$。

3.2 由参数控制的红绿灯道路模型

3.2.1 模型假设

1. 假设在距离路口停止线超过 100 米时，车辆在保证适当车距的前提下尽可能以最高限速行驶。
2. 假设区域道路限速为 $17m/s$。
3. 车辆近似为质点。
4. 假设车辆在距离停止线 $100m$ 时开始减速。

5. 假设所有车辆不会超车。
6. 假设行人不对车辆右转产生影响。
7. 假设车辆可以左转或直行时，斑马线上已经没有行人。
8. 假设汽车变道的时间可以忽略。
9. 假设车辆在进行转弯时的路径为标准的四分之一圆弧。
10. 假设每个路口在相同方向的红绿灯是同步的且按照顺时针方向变换出新的绿灯。
11. 假设车辆经过路口时，按照原速无法通过路口，它会降速等红灯而不是提速通过绿灯。
12. 假设距离路口停止线 $20m$ 以内时，$6m/s$ 为理想速度。
13. 假设加速度绝对值上限为 $2m/s^2$。

3.2.2 符号说明

符号	含义	单位
r_1	左转半径	m
r_2	右转半径	m
w_r	路口宽度	m
t_1	绿灯单次持续时长	s
t_2	黄灯单次持续时长	s
T	红绿灯变化周期	s
v	车速	$m \cdot s^{-1}$
a	车加速度	$m \cdot s^{-2}$
Δt	数据更新的单位时间	m
d_c	与前车距离差	m
d_v	与前车速度差	$m \cdot s^{-1}$
d_s	车辆距离停止线距离	m
d_l	车辆经过路口时需要行进的路程	m

a_0	加速度绝对值上限	$m \cdot s^{-2}$
v_l	道路最高限速	$m \cdot s^{-1}$
v_c	停止线 20m 内限速	$m \cdot s^{-1}$

3.2.3 模型简介

该模型的符号使用，模拟场景与传统红绿灯的道路模型相同，只有红绿灯轮换方式和绿灯持续时间不同。车辆模型的车类增加了两个属性分别是 $intersection$ 和 $time$，$intersection$ 属性表示车辆将要经过的十字路口，它一共有 9 个值，分别代表 9 个不同的路口。$time$ 属性表示当前路口的红绿灯还将保持的时长，$intersection$ 属性有利于对经过同一路口的车辆进行整合与同时改变 $time$ 属性；模拟的过程由于红绿灯的设置改变需要调用新的函数，所以稍有改变。

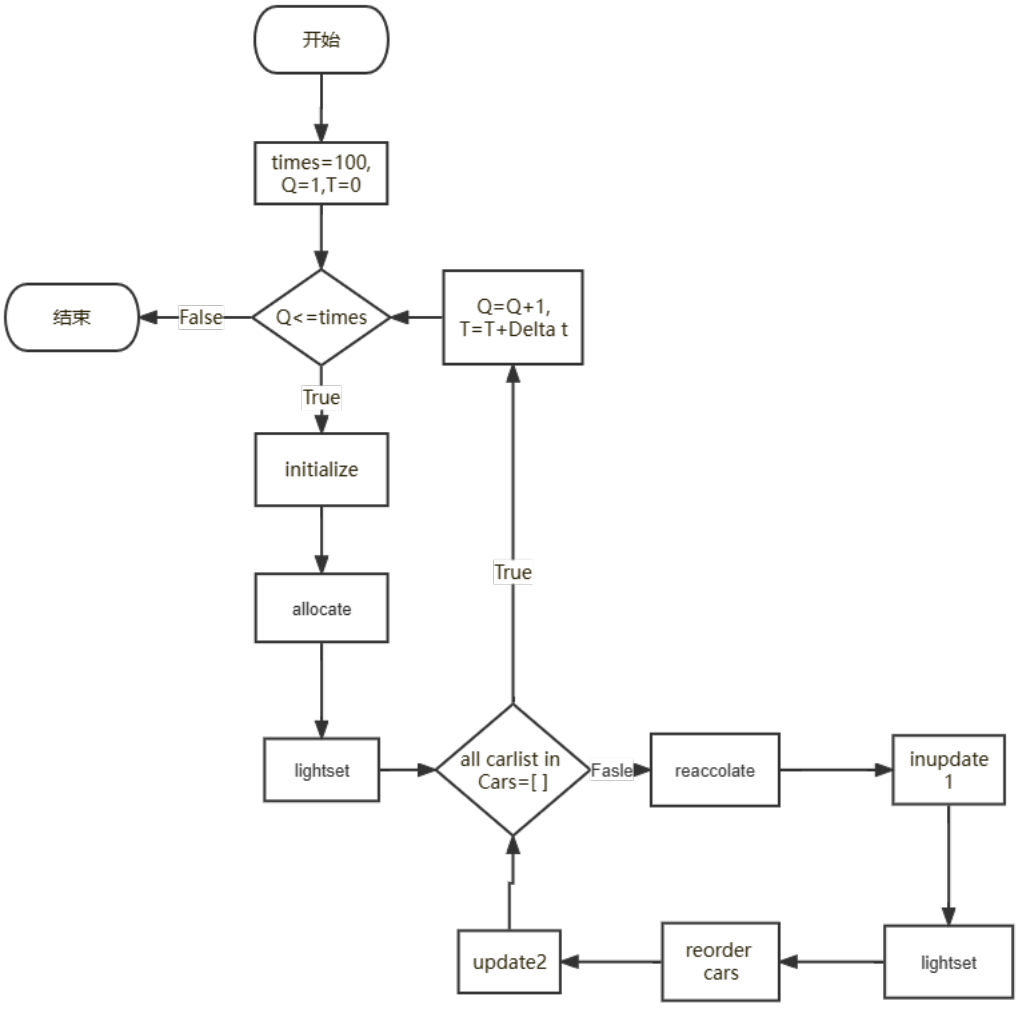

图 14: 参数控制的红绿灯道路模拟

3.2.4 红绿灯设计思路

对于每个路口，需要两个因素就可以确定该路口的红绿灯的状态，第一点是哪个方向的红绿灯为绿，第二点是绿灯持续时间应该多长。不妨先将第二点的绿灯持续时间定为 27s，接下来单独考虑将哪个方向的红绿灯设置为绿灯。直观上的想法是车多的一边应当设置为绿灯，但是存在一个问题，假如有一条道路车多但是距离十字路口还有很长的距离而另一条车道的车虽然少但是已经在十字路口停止线前等待通过，那么此时按照车的数量设置哪边为绿灯就不合理，所以应当只统计距离停止线某一段距离内的车数量，关于统计多少距离内的车并不确定，这是可以将其看作自变量，将模拟平均用时看作因变量。由于没有具体的函数表达两者的关系，所以无法采用求导的方式求极小值点。

但对于黑箱函数的极小值求法可以采用定步长搜索算法，即对参数初始化之后，向其所处空间的其它方向找一个距离一定的点，它代表一组新的参数，通过比较每个点对应函数值的大小，选出一个函数值最小的点作为新的起始点，接着继续向各个方向探寻更小函数值的点，直到达到指定迭代次数或者没有新的更小函数值的点[9]。

由于每次模拟时初始条件具有一定不同，即车辆的目的地是随机产生的，所以不同的模拟对应的黑箱函数 f 也不同，考察 f_1 与 f_2，在两个极端的随机初始条件下，f_1 函数的车辆全都以路程最短的出口作为目标点，而 f_2 函数的车辆全都以路程最远的出口作为目标点。这时通过定步长搜索算法求出来的极小值点不具有普遍性，只在特定初始条件下具有模拟时长短的特点，所以黑箱函数不能只用某个场景模拟，必须涵盖尽可能多的场景，令 $f(\omega)=\sum_{i=1}^{n} f_i(\omega)$,$n$ 的值越大，它包含的场景信息就越多，也越能消除由初始条件的随机性导致的结果随机性，使搜索出的结果具更加具有普遍性。

3.2.5 程序模拟

以统计的最远的车辆到停止线的距离为 x,x 的变化范围是 $[10,500]$，从 10m 开始每隔 10m 取一个点，将每个点对应的模拟平均用时作为 $f(x)$ 的值得到以下函数图像，具体数据见附录二。

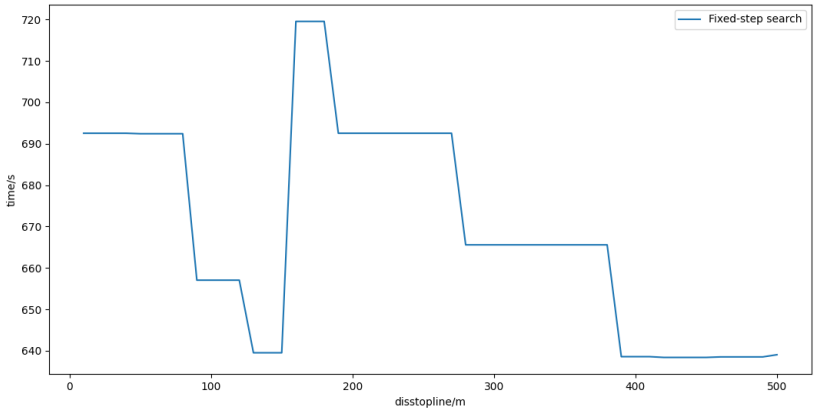

图 15: 定步长搜索

最小函数值为 638.41s，大于传统红绿灯的道路模型平均用时 459.20s[3, 9]。

3.3 传统红绿灯道路模型的改进

3.3.1 绿灯最优持续时长搜索

通过对参数控制红绿灯的道路模型结果的分析，可以得知轮流式换灯是一种本身就效率比较高的变灯方式，所以接下来只需要考虑绿灯持续时长的影响，在相同的道路模型，车辆模型基础上对不同的红绿灯持续时长进行模拟，可以得出最优单次绿灯持续时长。图 16是 100 次模拟的平均值折线图，由模拟结果可知，绿灯持续时长为 11s 时模拟平均值最小为 416.89s，比绿灯持续时长为 27s 的用时 434.51s 节省 4.4% 的时间，比绿灯持续时长为 34s 的用时 441.31s 节省 5.1% 的时间。

表 4: 不同绿灯持续时长对应的 100 次平均模拟时间

绿灯持续时长	100 次模拟平均用时（s）
10	417.6750471
11	416.8836647
12	420.0524294
13	425.0074235
14	427.1481059
15	431.4948412
16	427.7691647
17	426.2161294
18	426.2366471
19	429.5527471
20	429.7592706
21	435.0274412
22	436.0361588
23	427.7824647
24	428.3491529
25	431.5661824
26	431.3791353
27	434.5088588
28	435.6043706
29	432.6721647
30	436.9118588
31	439.8773176
32	437.1876235
33	439.6341824
34	441.3105882

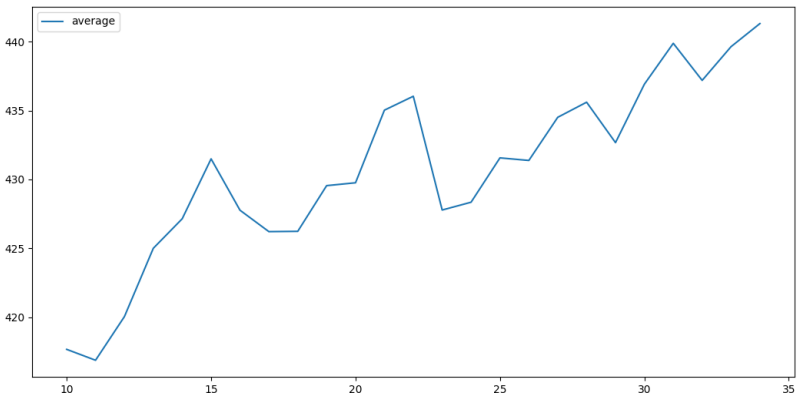

图 16: 100 次模拟平均值折线图

3.3.2 加装倒计时器

此外可以通过加装倒计时器的方法提高通行效率，符号说明与传统红绿灯道路模型相同, 模型假设除了“车辆在距离停止线 $100m$ 时开始减速”外其他假设与传统红绿灯道路模型相同。

程序模拟结果如下：

表 5: 加装倒计时器的 100 次模拟时间记录表

387.5294118	332.1176471	270.1176471	444.1317647
486.1317647	387.4847059	368.0141176	409.0729412
387.4847059	402.0752941	500.1170588	280.3670588
343.0729412	486.0729412	373.0752941	472.0164706
418.1341176	401.4847059	410.4847059	402.1317647
400.1317647	401.0729412	444.0729412	423.1317647
359.4847059	500.0141176	452.1170588	430.1341176
263.3923529	457.4847059	388.1176471	393.2958824
393.2935294	500.1317647	486.0164706	332.1341176
375.4847059	430.0729412	456.1317647	478.5288235
329.0141176	400.1317647	400.1170588	391.1170588
407.4870588	317.4111765	375.1317647	471.4847059
373.1170588	444.1341176	443.1317647	500.1317647
402.0729412	374.0588235	300.2182353	378.2494118
305.1782353	379.2935294	402.0729412	473.1194118
486.0164706	302.3335294	351.3382353	387.4847059
457.2517647	402.1317647	457.2494118	472.0164706
387.4870588	464.0729412	453.2494118	500.0164706
284.3670588	377.0605882	542.1194118	486.1341176
472.1317647	472.1317647	401.4847059	457.1905882
439.2517647	497.1317647	318.4282353	345.4870588
422.1341176	459.0141176	375.4111765	500.0164706
486.1317647	412.3105882	478.5288235	416.0141176
450.1317647	478.1929412	299.9388235	379.3547059
411.2494118	379.2935294	304.4705882	373.1170588

通过加装倒计时器提高通行效率的原理是，车辆不用担心由于信号灯突然变为红灯导致刹车不及而提前减速，若还有较长时间绿灯，则减速是没有必要的，所以通过加装倒计时器向车主传递绿灯剩余时长信息可以有效提高通行效率。以提前计时 5 秒计时为例，可以将刹车分界线划为 85m，因为 85m 恰好是以最大限速 $17m/s$ 在 5s 内行驶的距离，倒计时开始的时候，85m 以内的车辆不用减速能够通过，85m 之后的车辆无法通过，需要减速，同时采用上一小节的结论，使绿灯持续时长为 11s，由程序模拟 100 次的平均值为 410.31s，比不加倒计时器的 27s 单次绿灯时长下的模拟用时减少 10.65% 的时间。

4 总结与分析

4.1 全文总结

文章首先介绍了路况信息实时采集的技术手段，系统组成工作原理为仿真模拟提供了现实依据和支撑。接着分别建立了传统红绿交通模型和参数控制红绿灯的交通模型，预期猜测是在参数控制红绿灯的交通模型中能够获得更短的模拟时长，通过对参数控制红绿灯的交通模型中的参数定步长搜索获得参数取不同值时的模拟时长，但从模拟结果看传统红绿交通模型的模拟时长更短。在模拟数据的基础上，文章继续对传统红绿灯道路模型进行优化，通过改变绿灯单次持续时长研究绿灯单次持续时长对传统红绿灯道路模型的通行时间的影响，最终得出改变绿灯单次持续时长对通行效率存在影响，并且存在一个最优时长使通行效率最高。最后为传统红绿灯道路模型增加了倒计时器，仿真模拟的结果表明加装倒计时器同样能够提高道路通行效率。

4.2 优点与不足

优点：通过程序模拟和的方法，成功设计出了能够提高通行效率的红绿灯，首先通过定步长搜索发现轮换式变灯是比较高效的道路模型，其次通过分别对不同绿灯持续时长进行模拟，发现绿灯持续时长为 11s 时，通车效率最高，减少了 17.62s，也就是 4.4% 的整体通过时间，最后再通过增加计时器进一步提高了车辆通行效率，比不加计时器的的模拟时长减少 10.65%，成功利用仿真模拟得到能够提高通行效率的红绿灯方案。

不足：本文的红绿灯只是针对当前模型的设计结果，并没有对其他类型的路口和道路模型进行建模分析，所以结论不具备迁移性，从推广的角度看具有局限性，其次，在假设中忽略了行人对交通的影响，与现实的贴近程度不够高。

4.3 改进建议

- 取消将车辆视作质点的假设，给车类增加车身长度属性。
- 增加车类的子类如救护车，消防车并分别设定不同属性与行车规则。
- 考虑行人对车辆的影响，改变右转车道恒为绿灯的设定。
- 在加速度的设置上采用更加精准的数学模型，更精巧的数学公式。

- 设计不同的交通路口的模型，在大量的道路模型基础上分析总结红绿灯通用规则。

- 改变多次模拟消除随机性的思路，当初始车辆增加时，需要更多的模拟次数，才能一定程度上消除随机性的影响。采用对随机产生的初始道路情况深拷贝的方法，将需要对比的另一种方法采用同一个初始条件，这样不受初始条件的随机性影响并且极大提高程序运行速度。

以上措施都将使模型更加贴近现实，从而增强模型的现实可参照性。此外还可以将模型整体设计成函数并将关键参数设计成函数的形参，在使用时可以方便从外部调用，不用修改内部代码。

对于参数控制的红绿灯道路模型，可以增加一个参数绿灯持续时长，对二元参数应用定步长搜索法；也可以考虑对不同方向的绿灯设置不同的持续时长，时长设置由一个或多个参数决定。

参考文献

[1] 许鹏辉. 基于移动终端的交通路况信息实时采集与显示系统 [D]. 南京理工大学,2017.

[2] 许鹏辉, 郭玲, 施盼. 基于移动终端的路况信息实时采集与显示系统 [J]. 计算机与现代化,2017,No.259(03):22-26.

[3] 林宇翔, 王方霄, 苏宇辰, 李明奇. 基于有限差分法与变步长搜索法的炉温曲线设计 [J]. 实验科学与技术,2022,20(05):45-51.

[4] 温健, 慕冰, 严秋实, 王岩. 基于“卡口”测速数据的城市道路限速值合理性论证研究 [J]. 道路交通管理,2023(02):56-58.

[5] 程留. 基于强化学习和计算机仿真的交通信号调度 [D]. 大连理工大学,2021.DOI:10.26991/d.cnki.gdllu.2021.001567.

[6] 白家豪. 道路限速区段划分方法的研究 [J]. 产业与科技论坛,2022,21(23):54-56.

[7] 王树凤, 孙文盛, 刘宗锋. 车辆稳定换道时的侧向加速度分析 [J]. 机械设计与制造,2020(07):17-20+24.DOI:10.19356/j.cnki.1001-3997.2020.07.005.

[8] 刘长运. 基于加速度信号的车辆行驶状态识别算法 [J]. 湖北汽车工业学院学报,2018,32(04):32-34.

[9] 张觉非，陈震. 用 Python 实现深度学习框架 [M]. 北京：人民邮电出版社，2020(10):18-23.

[10] 黄海波, 曹利坤, 王学军. 预动式交通路口的红绿灯优化设计思路———一种智能的红绿灯信号控制方法 [J]. 广东公安科技,2022,30(03):53-55.

[11] 张亚婉, 胡洽锋, 唐艳凤, 黄信维. 低峰期交通红绿灯减少候灯时间系统设计 [J]. 机电工程技术,2021,50(06):155-157.

[12] 于增亮. 基于仿真环境驾驶员临界反应能力的研究 [D]. 吉林大学,2005.

[13] 王满力, 张玉强. 有关防止闯红绿灯的智能化设计方案 [J]. 无线互联科技,2013(11):80.

[14] Aditi Agrawal,Rajeev Paulus. Intelligent traffic light design and control in smart cities: a survey on techniques and methodologies[J]. International Journal of Vehicle Information and Communication Systems,2020,5(4).

A 附录一

A.1 unintelligent.py

```
import initialize
import update
import reorder
import gc
times=100
Deltat=1
Q=1
set=[]
while Q<=times:
    cars=initialize.initialize1()
    initialize.initialize2(cars)
    initialize.initialize3(cars)
    n=0
    T=0
    timeset=[]
    while True:
        n+=1
        T+=Deltat
        speeds=[]
        for car in cars:
            speeds+=[car.speed]
        [cars,timeset]=update.update1(cars,timeset,n,speeds)
        if cars==[]:
            break
        else:
            cars=reorder.reorder(cars)
            update.update2(cars)
    cars.clear()
    del cars
    gc.collect()
    Q+=1
    set+=[max(timeset)]
    print("第",Q-1,"次非智能模拟完成,模拟时间为",max(timeset))
print("非智能模拟的时间集合为:",set,"平均值为",'%.2f' % float(sum(set)/len(set)))
```

A.2 initialize.py

```
import random
pi=3.14
eachnumber=60
class Car:
    def __init__(self,direction,coordinate):
        self.direction=direction
        self.coordinate = coordinate
        self.serial=0
        self.disstopline = 0
        self.dislimitation=0
        self.light = 1
        self.lis = []
        self.speed = 17
        self.d_c = 50
        self.acceleration=0
        self.d_v=0
        self.time=0
        self.intersection=0
def makedisstopline(self,n):
    self.disstopline=(n-1)*50+1200
def listmaker(p1, p2):
    p = [0, 0]
    p[0] = p2[0] - p1[0]
    p[1] = p2[1] - p1[1]
    if p1[1] == -2:
        pass
    elif p1[1] == 2:
        p[0] = -p[0]
        p[1] = -p[1]
    elif p1[0] == 2:
        t = p[0]
        p[0] = p[1]
        p[1] = -t
    elif p1[0] == -2:
        t = p[0]
        p[0] = -p[1]
        p[1] = t
    if p[1] == 4: # 由底到顶
```

```
        if p[0] == 0:
            return [0, 0, 0]
        elif p[0] == -1:
            return [-1, 1, 0, 0]
        elif p[0] == -2:
            return [-1, 0, 1, 0, 0]
        elif p[0] == 1:
            return [1, -1, 0, 0]
        elif p[0] == 2:
            return [1, 0, -1, 0, 0]
    elif p[1] == 0: # 由底到底
        if p[0] == -1:
            return [-1, -1]
        elif p[0] == 1:
            return [1, 1]
        elif p[0] == -2:
            return [-1, 0, -1]
        elif p[0] == 2:
            return [1, 0, 1]
    else: # 由底到边
        temporary = []
        if p[1] > 1:
            for i in range(p[1] - 1):
                temporary += [0]
            if p[0] > 0:
                temporary += [1]
                for j in range(p[0] - 1):
                    temporary += [0]
                return temporary
            elif p[0] < 0:
                temporary += [-1]
                for j in range(p[0] - 1):
                    temporary += [0]
                return temporary
        else:
            if p[0] > 0:
                temporary += [1]
                for j in range(p[0] - 1):
                    temporary += [0]
                return temporary
```

```
            elif p[0] < 0:
                temporary += [-1]
                for j in range(p[0] - 1):
                    temporary += [0]
                return temporary
def makelis(self):
    positions = [[-1, -2], [-1, 2], [0, -2], [0, 2], [1, -2], [1, 2], [-2, -1],
        [2, -1], [-2, 0], [2, 0], [-2, 1],[2, 1]]
    positions.remove(self.coordinate)
    outposition = random.choice(positions)
    self.lis = listmaker(self.coordinate, outposition)
def makeserial(self):
    if self.lis[0]==-1:
        self.serial=-1
    elif self.lis[0] ==1:
        self.serial=1
    else:
        self.serial = 0
def makelight(self):
    if self.serial==1:
        self.light=1
    else:
        if self.direction==[0,1]:
            self.light=1
        else:
            self.light=-1
def makedislimitation(self):
    if self.serial == -1:
        self.dislimitation=14*pi
    elif self.serial == 1:
        self.dislimitation=2*pi
    else:
        self.dislimitation =48
def initialize2(cars):
    for car in cars:
        makelis(car)
        makeserial(car)
        makelight(car)
        makedislimitation(car)
def initialize1():
```

```
    directions=[[1,0],[-1,0],[0,1],[0,-1]]
    coordinates = [[-2, -1], [-2, 0], [-2, 1]], [[2, -1], [2, 0], [2, 1]], [[-1,
        -2], [0, -2], [1, -2]], [[-1, 2], [0, 2], [1, 2]]
    cars=[]
    i=0
    for direction in directions:
        for coordinate in coordinates[i]:
            for j in range(eachnumber):
                cars+=[Car(direction,coordinate)]
        i+=1
    return cars
def initialize3(cars):
    zushu=int(len(cars)/eachnumber)
    for i in range(zushu):
        serialminus = 0
        serialzero = 0
        serialplus = 0
        for j in range(eachnumber):
            if cars[i * eachnumber + j].serial==-1:
                serialminus+=1
                makedisstopline(cars[i*eachnumber+j],serialminus)
            elif cars[i * eachnumber + j].serial == 0:
                serialzero+=1
                makedisstopline(cars[i * eachnumber + j], serialzero)
            elif cars[i * eachnumber + j].serial == 1:
                serialplus+=1
                makedisstopline(cars[i * eachnumber + j], serialplus)
```

A.3 update.py

```
import math
pi=3.14
Deltat=1
def lasttime(s,v):#以初速度v，加速度为a，限速17m/s,行驶最后s所用时间的计算公式
    a=2
    if s<=(17**2-v**2)/2/a:
        if v**2+2*a*s>=0:
            t=-v/a+math.sqrt(v**2+2*a*s)/a
        else:
```

```
            t=-v/a
    else:
        t=s/17-(17**2-v**2)/34/a
    return t
def lightchange(self,T):
    if self.direction==[0,1]:
        if T%120>=0 and T%120<=27:
            self.light=1
        elif T%120>27 and T%120<=30:
            self.light=0
        elif T%120>30 and T%120<120:
            self.light=-1
    elif self.direction==[-1,0]:
        if (T-30)%120>=0 and (T-30)%120<=27:
            self.light=1
        elif (T-30)%120>27 and (T-30)%120<=30:
            self.light=0
        elif (T-30)%120>30 and (T-30)%120<120:
            self.light=-1
    elif self.direction==[0,-1]:
        if (T-60)%120>=0 and (T-60)%120<=27:
            self.light=1
        elif (T-60)%120>27 and (T-60)%120<=30:
            self.light=0
        elif (T-60)%120>30 and (T-60)%120<120:
            self.light=-1
    elif self.direction==[1,0]:
        if (T-90)%120>=0 and (T-90)%120<=27:
            self.light=1
        elif (T-90)%120>27 and (T-90)%120<=30:
            self.light=0
        elif (T-90)%120>30 and (T-90)%120<120:
            self.light=-1
def lightchange2(self,T,t):
    T2=4*(t+3)
    if self.direction==[0,1]:
        if T%T2>=0 and T%T2<=t:
            self.light=1
        elif T%T2>t and T%T2<=t+3:
            self.light=0
```

```
        elif T%T2>t+3 and T%T2<T2:
            self.light=-1
    elif self.direction==[-1,0]:
        if (T-t-3)%T2>=0 and (T-t-3)%T2<=t:
            self.light=1
        elif (T-t-3)%T2>t and (T-t-3)%T2<=t+3:
            self.light=0
        elif (T-t-3)%T2>t+3 and (T-t-3)%T2<T2:
            self.light=-1
    elif self.direction==[0,-1]:
        if (T-2*t-6)%T2>=0 and (T-2*t-6)%T2<=t:
            self.light=1
        elif (T-2*t-6)%T2>t and (T-2*t-6)%T2<=t+3:
            self.light=0
        elif (T-2*t-6)%T2>t+3 and (T-2*t-6)%T2<T2:
            self.light=-1
    elif self.direction==[1,0]:
        if (T-3*t-9)%T2>=0 and (T-3*t-9)%T2<=t:
            self.light=1
        elif (T-3*t-9)%T2>t and (T-3*t-9)%T2<=t+3:
            self.light=0
        elif (T-3*t-9)%T2>t+3 and (T-3*t-9)%T2<T2:
            self.light=-1
def update1s(cars,timeset,n,speeds,w):
    T=n*Deltat
    records=[]
    L=len(cars)
    for i in range(L):
        cars[i].disstopline = cars[i].disstopline - speeds[i] * Deltat
        cars[i].speed = cars[i].speed + cars[i].acceleration * Deltat
        if cars[i].disstopline>0:
            if cars[i].serial != 1:
                lightchange2(cars[i],T,w)
        else:
            if cars[i].lis==[]:
                cars[i].lis+=[0]
            cars[i].lis.pop(0)
            if cars[i].lis==[]:
                timeset +=
                    [T+lasttime(1200+cars[i].dislimitation+cars[i].disstopline,
```

```
                    cars[i].speed)]
89                records+=[i]
90            else:
91                updatecoordinate(cars[i])
92                directionchange(cars[i])
93                cars[i].serial=cars[i].lis[0]
94                cars[i].disstopline =
                      1200+cars[i].dislimitation+cars[i].disstopline
95                makedislimitation(cars[i])
96                if cars[i].serial != 1:
97                    lightchange2(cars[i],T,w)
98                else:
99                    cars[i].light=1
100    records.reverse()
101    for record in records:
102        cars.pop(record)
103    cars=revise(cars)
104    return [cars,timeset]
105def directionchange(self):
106    if self.serial==1:
107        temp=self.direction[0]
108        self.direction[0]=self.direction[1]
109        self.direction[1]=-temp
110    elif self.serial==-1:
111        temp = self.direction[0]
112        self.direction[0] = -self.direction[1]
113        self.direction[1] = temp
114def updatecoordinate(self):
115    if self.direction == [1,0] and self.serial==-1:#右上
116        if self.coordinate[0]<=-1:
117            self.coordinate[0]+=1
118        if self.coordinate[1]>=0:
119            self.coordinate[1]+=1
120    elif self.direction == [-1,0] and self.serial==1:#左上
121        if self.coordinate[0]>=1:
122            self.coordinate[0]-=1
123        if self.coordinate[1]>=0:
124            self.coordinate[1]+=1
125    elif self.direction == [1,0] and self.serial==1:#右下
126        if self.coordinate[0]<=-1:
```

```
            self.coordinate[0]+=1
        if self.coordinate[1]<=0:
            self.coordinate[1]-=1
    elif self.direction == [-1,0] and self.serial==-1:#左下
        if self.coordinate[0]>=1:
            self.coordinate[0]-=1
        if self.coordinate[1]<=0:
            self.coordinate[1]-=1
    elif self.direction == [0,1] and self.serial==-1:#上左
        if self.coordinate[1]<=-1:
            self.coordinate[1]+=1
        if self.coordinate[0]<=0:
            self.coordinate[0]-=1
    elif self.direction == [0,-1] and self.serial==1:#下左
        if self.coordinate[1]>=1:
            self.coordinate[1]-=1
        if self.coordinate[0]<=0:
            self.coordinate[0]-=1
    elif self.direction == [0,1] and self.serial==1:#上右
        if self.coordinate[1]<=-1:
            self.coordinate[1]+=1
        if self.coordinate[0]>=0:
            self.coordinate[0]+=1
    elif self.direction == [0,-1] and self.serial==-1:#下右
        if self.coordinate[1]>=1:
            self.coordinate[1]-=1
        if self.coordinate[0]>=0:
            self.coordinate[0]+=1
def makedislimitation(self):
    if self.serial == -1:
        self.dislimitation=14*pi
    elif self.serial == 1:
        self.dislimitation=2*pi
    else:
        self.dislimitation =48
def revise(cars):
    for car in cars:
        if car.lis==[]:
            cars.remove(car)
    for car in cars:
```

```
167         if car.speed>17:
168             car.speed=17
169         elif car.speed<0:
170             car.speed=0
171     return cars
172 def update1(cars,timeset,n,speeds):
173     T=n*Deltat
174     records=[]
175     L=len(cars)
176     for i in range(L):
177         cars[i].disstopline = cars[i].disstopline - speeds[i] * Deltat
178         cars[i].speed = cars[i].speed + cars[i].acceleration * Deltat
179         if cars[i].disstopline>0:
180             if cars[i].serial != 1:
181                 lightchange(cars[i],T)
182         else:
183             if cars[i].lis==[]:
184                 cars[i].lis+=[0]
185             cars[i].lis.pop(0)
186             if cars[i].lis==[]:
187                 timeset
                        +=[T+lasttime(1200+cars[i].dislimitation+cars[i].disstopline,
                        cars[i].speed)]
188                 records+=[i]
189             else:
190                 updatecoordinate(cars[i])
191                 directionchange(cars[i])
192                 cars[i].serial=cars[i].lis[0]
193                 cars[i].disstopline =
                        1200+cars[i].dislimitation+cars[i].disstopline
194                 makedislimitation(cars[i])
195                 if cars[i].serial != 1:
196                     lightchange(cars[i],T)
197                 else:
198                     cars[i].light=1
199     records.reverse()
200     for record in records:
201         cars.pop(record)
202     cars=revise(cars)
203     return [cars,timeset]
```

```
def updateacc1(self):
    if self.disstopline>100:
        if 17-self.speed<2*Deltat:
            self.acceleration=(17-self.speed)/Deltat
        else:
            self.acceleration=2
    elif self.disstopline>=20 and self.disstopline<=100:
        if self.speed>6:
            if self.speed-6>=2*Deltat:
                self.acceleration=-2
            else:
                self.acceleration=-(self.speed-6)/Deltat
        elif self.speed<6:
            if 6-self.speed>=2*Deltat:
                self.acceleration=2
            else:
                self.acceleration=(6-self.speed)/Deltat
        else:
            self.acceleration=0
    else:
        if self.light==1:
            if abs(self.speed-6)/Deltat<=2:
                self.acceleration=(6-self.speed)/Deltat
            else:
                if self.speed>6:
                    self.acceleration=-2
                else:
                    self.acceleration=2
        else:
            if self.speed/Deltat<=2:
                self.acceleration=-self.speed/Deltat
            else:
                a=-2
def updateacc2(self):
    if self.d_c>3*self.speed and self.d_v<0:
        if -self.d_v>=2*Deltat:
            self.acceleration=2
        else:
            self.acceleration=-self.d_v/Deltat
    elif self.d_c<3*self.speed and self.d_v>0:
```

```
244         if self.d_v>=2*Deltat:
245             self.acceleration=-2
246         else:
247             self.acceleration=-self.d_v/Deltat
248     else:
249         self.acceleration=0
250 def update2(cars):
251     i=0
252     for car in cars:
253         i+=1
254         if car.d_v==None:
255             updateacc1(car)
256         else:
257             updateacc2(car)
258 
259 def judge0(car):
260     if (car.direction==[1,0] and car.coordinate==[-2,1]) or
          (car.direction==[-1,0] and car.coordinate==[-1,1]) or
          (car.direction==[0,-1] and car.coordinate==[-1,2]) or
          (car.direction==[0,1] and car.coordinate==[-1,1]):
261         return True
262 def judge1(car):
263     if (car.direction==[1,0] and car.coordinate==[-1,1]) or
          (car.direction==[-1,0] and car.coordinate==[1,1]) or
          (car.direction==[0,-1] and car.coordinate==[0,2]) or
          (car.direction==[0,1] and car.coordinate==[0,1]):
264         return True
265 def judge2(car):
266     if (car.direction==[1,0] and car.coordinate==[1,1]) or (car.direction==[-1,0]
          and car.coordinate==[2,1]) or (car.direction==[0,-1] and
          car.coordinate==[1,2]) or (car.direction==[0,1] and
          car.coordinate==[1,1]):
267         return True
268 def judge3(car):
269     if (car.direction==[1,0] and car.coordinate==[-2,0]) or
          (car.direction==[-1,0] and car.coordinate==[-1,0]) or
          (car.direction==[0,-1] and car.coordinate==[-1,1]) or
          (car.direction==[0,1] and car.coordinate==[-1,-1]):
270         return True
271 def judge4(car):
```

```
272     if (car.direction==[1,0] and car.coordinate==[-1,0]) or
            (car.direction==[-1,0] and car.coordinate==[1,0]) or
            (car.direction==[0,-1] and car.coordinate==[0,1]) or
            (car.direction==[0,1] and car.coordinate==[0,-1]):
273         return True
274 def judge5(car):
275     if (car.direction==[1,0] and car.coordinate==[1,0]) or (car.direction==[-1,0]
            and car.coordinate==[2,0]) or (car.direction==[0,-1] and
            car.coordinate==[1,1]) or (car.direction==[0,1] and
            car.coordinate==[1,-1]):
276         return True
277 def judge6(car):
278     if (car.direction==[1,0] and car.coordinate==[-2,-1]) or
            (car.direction==[-1,0] and car.coordinate==[-1,-1]) or
            (car.direction==[0,-1] and car.coordinate==[-1,-1]) or
            (car.direction==[0,1] and car.coordinate==[-1,-2]):
279         return True
280 def judge7(car):
281     if (car.direction==[1,0] and car.coordinate==[-1,-1]) or
            (car.direction==[-1,0] and car.coordinate==[1,-1]) or
            (car.direction==[0,-1] and car.coordinate==[0,-1]) or
            (car.direction==[0,1] and car.coordinate==[0,-2]):
282         return True
283 def judge8(car):
284     if (car.direction==[1,0] and car.coordinate==[1,-1]) or
            (car.direction==[-1,0] and car.coordinate==[2,-1]) or
            (car.direction==[0,-1] and car.coordinate==[1,-1]) or
            (car.direction==[0,1] and car.coordinate==[1,-2]):
285         return True
286 def allocate(cars):
287     Cars=[]
288     for i in range(9):
289         Cars+=[[]]
290     for car in cars:
291         if judge0(car):
292             car.intersection=0
293             Cars[0]+=[car]
294         elif judge1(car):
295             car.intersection=1
296             Cars[1]+=[car]
```

```
297         elif judge2(car):
298             car.intersection=2
299             Cars[2]+=[car]
300         elif judge3(car):
301             car.intersection=3
302             Cars[3]+=[car]
303         elif judge4(car):
304             car.intersection=4
305             Cars[4]+=[car]
306         elif judge5(car):
307             car.intersection=5
308             Cars[5]+=[car]
309         elif judge6(car):
310             car.intersection=6
311             Cars[6]+=[car]
312         elif judge7(car):
313             car.intersection=7
314             Cars[7]+=[car]
315         elif judge8(car):
316             car.intersection=8
317             Cars[8]+=[car]
318     return Cars
319 def renewintersection(car):
320     if judge0(car):
321         car.intersection=0
322     elif judge1(car):
323         car.intersection=1
324     elif judge2(car):
325         car.intersection=2
326     elif judge3(car):
327         car.intersection=3
328     elif judge4(car):
329         car.intersection=4
330     elif judge5(car):
331         car.intersection=5
332     elif judge6(car):
333         car.intersection=6
334     elif judge7(car):
335         car.intersection=7
336     elif judge8(car):
```

```
337         car.intersection=8
338 def inupdate1(cars,timeset,n,speeds):
339     T=n*Deltat
340     records=[]
341     L=len(cars)
342     for i in range(L):
343         cars[i].time-=Deltat
344         cars[i].disstopline = cars[i].disstopline - speeds[i] * Deltat
345         cars[i].speed = cars[i].speed + cars[i].acceleration * Deltat
346         if cars[i].disstopline>0:
347             pass
348         else:
349             if cars[i].lis==[]:
350                 cars[i].lis+=[0]
351             cars[i].lis.pop(0)
352             if cars[i].lis==[]:
353                 timeset+=[T+lasttime(1200+cars[i].dislimitation+cars[i].disstopline,
                        cars[i].speed)]
354                 records+=[i]
355             else:
356                 updatecoordinate(cars[i])
357                 directionchange(cars[i])
358                 renewintersection(cars[i])
359                 cars[i].serial=cars[i].lis[0]
360                 cars[i].disstopline =
                        1200+cars[i].dislimitation+cars[i].disstopline
361                 makedislimitation(cars[i])
362     records.reverse()
363     for record in records:
364         cars.pop(record)
365     revise(cars)
366     return [cars,timeset]
367 def reallocate(Cars):
368     for carlist in Cars:
369         for car in carlist:
370             if car.intersection!=Cars.index(carlist):
371                 Cars[car.intersection]+=[car]
372                 carlist.remove(car)
373     return Cars
374 def pri(cars):
```

```
375     i=0
376     for car in cars:
377         print("第",i+1,"辆车信息为")
378         print("self.lis =", cars[i].lis,
379             "self.speed=", cars[i].speed,
380             "self.acceleration=", cars[i].acceleration,
381             "self.disstopline =", cars[i].disstopline,
382             "self.intersection =", cars[i].intersection,
383             "self.dislimitation=", cars[i].dislimitation,
384             "self.d_v=", cars[i].d_v,
385             "self.d_c =", cars[i].d_c,
386             "self.direction=", cars[i].direction,
387             "self.coordinate =", cars[i].coordinate,
388             "self.serial=", cars[i].serial,
389             "self.light =", cars[i].light,
390             "self.time =", cars[i].time
391             )
392         i+=1
393 def lightset(carlist,w1,w2,w3):
394     if carlist!=[]:
395         if carlist[0].time>=Deltat:
396             pass
397         else:
398             numset=[0,0,0,0]
399             disstoplines=[0,0,0,0]
400             averds=[0,0,0,0]
401             for car in carlist:
402                 if car.serial!=1 and car.direction==[1,0] and car.disstopline<=w1:
403                     numset[0]+=1
404                     disstoplines[0]+=car.disstopline
405                 elif car.serial!=1 and car.direction==[-1,0] and
                        car.disstopline<=w1:
406                     numset[1]+=1
407                     disstoplines[1]+=car.disstopline
408                 elif car.serial!=1 and car.direction==[0,-1] and
                        car.disstopline<=w1:
409                     numset[2]+=1
410                     disstoplines[2]+=car.disstopline
411                 elif car.serial!=1 and car.direction==[0,1] and
                        car.disstopline<=w1:
```

```
                numset[3]+=1
                disstoplines[3]+=car.disstopline
        for i in range(4):
            if numset[i]!=0:
                averds[i]=disstoplines[i]/numset[i]
            else:
                averds[i]=10000000
        grades=[0,0,0,0]
        for i in range(4):
            grades[i]=numset[i]-w2*averds[i]
        mgindex=grades.index(max(grades))
        if mgindex==0:
            for car in carlist:
                if car.direction==[1,0]:
                    car.light=1
                elif car.serial==1:
                    car.light=1
                else:
                    car.light=-1
        elif mgindex==1:
            for car in carlist:
                if car.direction==[-1,0]:
                    car.light=1
                elif car.serial==1:
                    car.light=1
                else:
                    car.light=-1
        elif mgindex==2:
            for car in carlist:
                if car.direction==[0,-1]:
                    car.light=1
                elif car.serial==1:
                    car.light=1
                else:
                    car.light=-1
        elif mgindex==3:
            for car in carlist:
                if car.direction==[0,1]:
                    car.light=1
                elif car.serial==1:
```

```
                    car.light=1
                else:
                    car.light=-1
        for car in carlist:
            car.time=w3
```

A.4 reorder.py

```
def reorder(cars):
    L = len(cars)
    index = []
    for i in range(L):
        index += [i]
    newcars = []
    newcars2 = []
    while index != []:
        record = []
        for i in range(len(index)):
            if i == 0:
                newcars += [cars[index[i]]]
                record += [0]
            elif cars[index[i]].direction == newcars[-1].direction and
                cars[index[i]].coordinate == newcars[
                -1].coordinate and cars[index[i]].serial == newcars[-1].serial:
                newcars += [cars[index[i]]]
                record += [i]
        cardisstoplines = []
        for i in record:
            cardisstoplines += [cars[index[i]].disstopline]
        ordered_list = sorted(range(len(cardisstoplines)), key=lambda k:
            cardisstoplines[k]) # 获取索引排序
        cars[index[ordered_list[0]]].d_c=None
        cars[index[ordered_list[0]]].d_v = None
        for i in ordered_list:
            newcars2 += [cars[index[i]]]
        record.reverse()
        for number in record:
            index.pop(number)
    return newcars2
```

A.5 intelligent.py

```
import initialize
import update
import reorder
times=8
Deltat=0.1
Q=1
set=[]
while Q<=times:
    timeset=[]
    cars=initialize.initialize1()
    initialize.initialize2(cars)
    initialize.initialize3(cars)
    Cars=update.allocate(cars)
    for carlist in Cars:
        update.lightset(carlist,200,0.2,11)
    n=0
    T=0
    while not all(carlist==[] for carlist in Cars):
        n+=1
        T+=Deltat
        print("第",n,"次更新")
        for carlist in Cars:
            speeds=[]
            for car in carlist:
                speeds+=[car.speed]
            [carlist,timeset]=update.inupdate1(carlist,timeset,n,speeds)
        Cars=update.reallocate(Cars)
        for carlist in Cars:
            update.lightset(carlist,5,0.3,20)
            carlist=reorder.reorder(carlist)
            update.update2(carlist)
    set+=[round(max(timeset),2)]
    print("第",Q,"次智能模拟完成,模拟时间为",max(timeset))
    Q+=1
print("智能模拟的时间集合为:",set,"平均值为",'%.2f' % float(sum(set)/len(set)))
```

A.6 定步长搜索.py

```
import initialize
import update
import reorder
import copy
import gc
times=2
Deltat=1
k1=[a for a in range(10,510,10)]
#因变量列表存放[f(a)]
sumdepent=[]
averdepent=[]
p=1
while p<=times:
    cars=initialize.initialize1()
    initialize.initialize2(cars)
    initialize.initialize3(cars)
    Cars0=update.allocate(cars)
    k=1
    for a in k1:
        Cars=copy.deepcopy(Cars0)
        timeset=[]
        for carlist in Cars:
            update.lightset(carlist,a,0,27)
        n=0
        T=0
        while not all(carlist==[] for carlist in Cars):
            n+=1
            T+=Deltat
            for carlist in Cars:
                speeds=[]
                for car in carlist:
                    speeds+=[car.speed]
                [carlist,timeset]=update.inupdate1(carlist,timeset,n,speeds)
            Cars=update.reallocate(Cars)
            for carlist in Cars:
                update.lightset(carlist,a,0,27)
                carlist=reorder.reorder(carlist)
                update.update2(carlist)
```

```
        del Cars
        gc.collect()
        sumdepent+=[max(timeset)]
        print("第",p,"次模拟的","第",k,"次搜索结果记录：",a,max(timeset))
        k+=1
    p+=1
averdepent=[i/times for i in sumdepent]
print("搜索完成")
print("自变量集合\n",k1)
print("因变量集合\n",averdepent)
```

A.7 绿灯最优持续时长搜索.py

```
import initialize
import update
import reorder
import copy
times=2
lastingtimes=[i for i in range(10,35,1)]
set=[0*i for i in range(10,35,1)]
r=1
while r<=times:
    cars=initialize.initialize1()
    initialize.initialize2(cars)
    initialize.initialize3(cars)
    cars2=copy.deepcopy(cars)
    id=0
    for time in lastingtimes:
        Deltat=1
        cars=copy.deepcopy(cars2)
        T=0
        n=0
        timeset=[]
        while True:
            print("第",r,"次初始化","绿灯时长为",time,"的第",n+1,"次更新")
            n+=1
            T+=Deltat
            speeds=[]
            for car in cars:
```

```
                speeds+=[car.speed]
            [cars,timeset]=update.update1s(cars,timeset,n,speeds,time)
            if cars==[]:
                break
            else:
                cars=reorder.reorder(cars)
                update.update2(cars)
        set[id]+=max(timeset)
        id+=1
    r+=1
averset=[set[i]/times for i in range(len(set))]
print(averset)
```

A.8 加装倒计时器.py

```
import initialize
import update
import reorder
import gc
def updateacc1s(self):
    if self.disstopline>85:
        if 17-self.speed<2*Deltat:
            self.acceleration=(17-self.speed)/Deltat
        else:
            self.acceleration=2
    elif self.disstopline>=20 and self.disstopline<=85:
        if self.speed>6:
            if self.speed-6>=2*Deltat:
                self.acceleration=-2
            else:
                self.acceleration=-(self.speed-6)/Deltat
        elif self.speed<6:
            if 6-self.speed>=2*Deltat:
                self.acceleration=2
            else:
                self.acceleration=(6-self.speed)/Deltat
        else:
            self.acceleration=0
    else:
```

```
        if self.light==1:
            if abs(self.speed-6)/Deltat<=2:
                self.acceleration=(6-self.speed)/Deltat
            else:
                if self.speed>6:
                    self.acceleration=-2
                else:
                    self.acceleration=2
        else:
            if self.speed/Deltat<=2:
                self.acceleration=-self.speed/Deltat
            else:
                self.acceleration=-2
def updateacc2s(self):
    if self.d_c>3*self.speed and self.d_v<0:
        if -self.d_v>=2*Deltat:
            self.acceleration=2
        else:
            self.acceleration=-self.d_v/Deltat
    elif self.d_c<3*self.speed and self.d_v>0:
        if self.d_v>=2*Deltat:
            self.acceleration=-2
        else:
            self.acceleration=-self.d_v/Deltat
    else:
        self.acceleration=0
def update2s(cars):
    i=0
    for car in cars:
        i+=1
        if car.d_v==None:
            updateacc1s(car)
        else:
            updateacc2s(car)
times=100
Deltat=1
Q=1
set=[]
while Q<=times:
    cars=initialize.initialize1()
```

```
    initialize.initialize2(cars)
    initialize.initialize3(cars)
    n=0
    T=0
    timeset=[]
    while True:
        n+=1
        T+=Deltat
        speeds=[]
        for car in cars:
            speeds+=[car.speed]
        [cars,timeset]=update.update1s(cars,timeset,n,speeds,11)
        if cars==[]:
            break
        else:
            cars=reorder.reorder(cars)
            update2s(cars)
    cars.clear()
    del cars
    gc.collect()
    Q+=1
    set+=[max(timeset)]
    print("第",Q-1,"次模拟完成,模拟时间为",max(timeset))
print("模拟的时间集合为:",set,"平均值为",'%.2f' % float(sum(set)/len(set)))
```

B 附录二

表一

与停止线距离 (m)	模拟平均用时 (s)
10	692.5288235
20	692.5288235
30	692.5288235
40	692.5288235
50	692.4111765
60	692.4111765
70	692.4111765
80	692.4111765
90	657.0582353
100	657.0582353
110	657.0582353
120	657.0582353
130	639.5288235
140	639.5288235
150	639.5288235
160	719.5288235
170	719.5288235
180	719.5288235
190	692.5288235
200	692.5288235
210	692.5288235
220	692.5288235
230	692.5288235
240	692.5288235
250	692.5288235
260	692.5288235
270	692.5288235
280	665.5876471
290	665.5876471

300	665.5876471
310	665.5876471
320	665.5876471
330	665.5876471
340	665.5876471
350	665.5876471
360	665.5876471
370	665.5876471
380	665.5876471
390	638.5876471
400	638.5876471
410	638.5876471
420	638.4111765
430	638.4111765
440	638.4111765
450	638.4111765
460	638.5288235
470	638.5288235
480	638.5288235
490	638.5288235
500	639.0582353

致谢

非常感谢导师李玉玲的指导和耐心解答，没有导师的指导和建议，我或许会犯不少错误；我还要感谢不懈奋斗的自己，没有因为遇到难障碍就退缩不前，而是不断寻找解决问题的方法并攻克难关。此外，我要感谢父母给我的物质和精神上的双重支持，是他们给予了我勇敢向前的力量。最后，非常感谢北京师范大学图书馆和北京师范大学提供的数据库访问权限，这些优质的数据库对我撰写论文提供了巨大的帮助。