第六届湖南省研究生数学建模竞赛

题 目　　交通信号灯全局优化调控策略

摘　　要：

近年来，随着我国社会经济的快速发展，城市化的进程日益加速，城市道路网络迅速扩大，人们对于交通的需求也不断增加，同时小汽车的保有量逐年增长，导致城市道路交通拥堵状况逐渐加剧。由于受地理环境及经济等因素的制约，道路建设总是落后于交通量的增长，尤其是中国一些发达城市如北京、上海等发展迅速，且新道路建设的用地有限，导致城市交通道路与车辆的矛盾日益尖锐。因此，本文对交通信号灯全局优化调控策略问题进行了分析，主要研究内容如下：

针对给定路网结构、车辆行程计划的情况下，求解各路口交通灯的调控策略，使得各车辆都能尽快到达目的地的问题。本文从全路网仿真建模和交通信号灯与所属路口建模两方面展开分析，得到了全过程仿真模型，在给定车辆行程、各路口红绿灯周期和各道路的绿灯亮起顺序与时长后，即可对整个场景进行推演，得到各车辆在各路口的排队信息，以及车辆最终的行驶总时长，同时求解得到总分。通过交通信号灯与所属路口建模，设计路口的红绿灯周期与相应的各道路上绿灯亮起的时间和顺序，以提高相应分数。

针对考虑某时刻某条道路上突发道路设施故障，并导致道路无法继续通行的问题，分析讨论交通灯调度受影响的情况，并设计局部调度方案。本文主要从以下两方面考虑： 搜索受影响车辆的最短行驶路径。分析故障路段对交通的影响情况，考虑经过该路段的车辆，以及经过的时刻等信息，并记录车辆在到达故障路口后的起始位置和目标位置，按照行程最短原则重新规划路径。求解局部交通灯调度方案。考虑车辆路线调整问题，重新设计局部路口的交通灯周期，相应各道路的绿灯亮起顺序和时长，最终形成了考虑道路故障的新的解决方案。

。

一 问题背景

交通信号灯在调控城市交通中起到了重要的作用。合理配置交通信号灯调度方法，可以降低道路的通行压力，减少车辆的通行时间。交通信号灯的调度方案是交通网内所有交通灯调度方案的综合。一个路口的交通灯调度方案包括该路口的交通灯在一个周期内亮起绿灯的顺序以及持续时长。假设路网结构和车辆速度不变，配置交通信号灯调度方案关键需要解决以下问题：

(1) 在车辆行程计划不改变的前提下，设计交通信号灯调度方案，使得通行车辆的用时尽可能少，并分析算法适用的城市路网规模。

(2) 在道路突发故障的情况下，讨论车辆改变路径后对信号灯调度方案的影响，并对交通灯调度方案进行局部调整，给出调整后的方案。

(3) 考察所建模型在具体问题中的适用性。在已知路网情况与车辆行程信息的前提下，给出最优的交通信号灯调度方案。并在时刻 30 秒时设置故障，重新规划车辆路径，并给出调整后的交通灯调度方案。

二 模型假设

为辅助问题建模，需要作如下假设与简化：

(1) 城市路网由路口和道路组成，均有唯一的名称。道路连接路口，路口是道路的起点或终点。每条道路均为单向通行，连接两个不同的路口。但是可能有方向相反的两条道路连接相同的两个路口（对应现实中双向通行的街道）。

(2) 连接两个路口的道路，同一方向至多有一条。每条道路的中间不会与其他道路交叉（即，如果出现道路不交叉无法在平面上排布的情况，可视为存在隧道或立交桥）。

(3) 每个路口至少是一条道路的终点，另外一条道路的起点。

(4) 车辆行驶速度相同，因此可用畅通状态下的通行时间表示一条道路的长度。并且，车辆行驶速度不受其他车辆影响。

(5) 每条道路都对应设置一个交通灯，用于控制该条道路上的车辆能否进入该道路终点处的路口。由于道路与交通灯一一对应，因此使用道路名称指代其交通灯，可称为“某道路的交通灯”。

(6) 某路口的交通灯是指，所有以该路口为终点的道路的交通灯。

(7) 交通灯只有红色、绿色两种。红灯表示停止，即，车辆要停在所在道路上，不能穿过该道路终点处的路口。绿灯表示通行，车辆可以穿过该路口，并驰往以该路口为起点的其他道路。

(8) 当某道路的交通灯为红灯时，所有到达该道路终点的车辆排队等待绿灯。

(9) 在任何时刻，每个路口最多只有一个交通灯是绿灯。当一条道路的绿灯亮起时，只有这条道路上的车辆可以进入该路口（并驰往以该路口为起点的某一条道路），其他道路上准备进入该路口的车辆都必须排队等待。

(10) 忽略车辆的几何尺寸，不考虑车辆排队时对空间的要求。

(11) 当交通灯变为绿灯后，队列中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 1 秒。即，如果绿灯持续 n 秒，则队列中前 n 辆车恰好能够通过路口，其余车辆只能等待下一个绿灯。

(12) 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。每辆车的行程经过任意一个路口至多一次。

(13) 初始状态（即 0 时刻），所有的车辆都在各自路径上第一条道路的终点处，该道路的交通灯若为红灯，则等待绿灯；若为绿灯，则准备移动。如果有多辆车在同一条道路的终点处，则 ID 序号较小的车辆排在前面先行。之后，每辆车按照自己的行程计划行驶，遵守交通灯的指示，到达路径中最后一条道路的终点为止，其行程结束，立即被移出，不再考虑。即，车辆不需要在最后一条道路的终点处排队，也不需要进入下一个路口。

三 符号说明

D：总时长，单位秒。

I：路口总数，路口的 ID 为整数，从 0 到 $l-1$。

S：道路总数。

V：车辆总数。

F：车辆按时到达终点的基本得分。

L：通行时间，为整数，表示畅通状态下车辆从道路起点到终点所需要的时间，单位为秒。

车辆 ID：以大写字母 C 开头，后接 3 位整数。

P：表示车辆行程路径的道路总数。

A：调控方案中包括的路口数量。

i：，路口 ID。

N_i：表示路口 i 的调度方案中交通灯信息数。

M_j：表示车辆 j 到达终点时的得分，假设车辆在 T_j 时刻到达终点，那么该车的得分为：

$$M_j=\begin{cases}F+\left(D-T_j\right), & T_j\le D\\ 0, & T_j\ge D\end{cases} \tag{1}$$

车辆最短路程长 $T_j^{\min}$：表示车辆 j 在不等任何红灯情况下所经过的总路程，包括在路口的情况（注意这里路程与时间概念是一一对应的，因此后文中路程单位都用秒进行表示）。

四 数学建模

4.1 问题 1

问题 1 的主要任务是在给定路网结构、车辆行程计划的情况下，求解各路口交通灯的调控策略，使得各车辆都能尽快到达目的地。针对该问题，我们可以从以下两方面考虑：

(1) 建立全路网仿真模型，满足：在给定车辆行程、各路口红绿灯周期和各道路的绿灯亮起顺序与时长后，即可对整个场景进行推演，得到各车辆在各路口

的排队信息，以及车辆最终的行驶总时长，同时求解得到总分。

(2) 建立各路口模型，设计路口的红绿灯周期与相应的各道路上绿灯亮起的时间和顺序，使得总分最大。

下面分别对着两类模型进行分析；

4.1.1 全路网仿真建模

针对全路网仿真问题，本文考虑借鉴积分的思想，将场景时间进行离散化，按一定的时间步长进行推演。这里考虑到过路口的时间为 1 秒，车经过道路的用时（也代表道路长度）以及红绿灯时间单位都是秒，因此可以将 1 秒作为时间步长进行推演。这里考虑采用面向对象的编程方式，需要的准备工作如下。

建立道路类，成员变量包括道路名称、道路 ID、道路长度（也即车辆经过的时长）、道路起点的路口 ID，道路终点的路口 ID，道路上绿灯亮起相对时刻（一个周期内），绿灯亮起时长，车辆在当前道路上的排队情况等；

建立路口类，成员变量包括路口 ID，以该路口为起点和终点的道路数量、名称、以及车辆数，经过该路口的车辆总数等；

建立车辆类，成员变量包括车辆 ID，车辆经过的道路数，道路名称序列，车辆是否到达路口标识，车辆是否到达终点标识，车辆所处路段位置，车辆最短路程长（即不等任何红灯时的最大用时）等。

在建立完上述道路类、路口类、车辆类后，可读取给定的文件数据进行赋值，并新建相应的对象。完成参数初始化之后即可按照如下流程进行推演：

初始时刻 t = 0;
同一条道路上车辆按 ID 从小到达排队
while (t <= D)
{
更新车辆参数
判断车辆是否到达终点，是则车辆标记退出，并记录车辆退出的时刻；
判断车辆是否达到路口，是则在相应道路上排队；
判断当前时刻路口绿灯亮起情况，让亮绿灯的道路上排在最前面的一辆车通过路口，并更新该车辆所在道路信息，标记其已经离开路口；同时更新路口排队信息，删除排在最前面的一辆车；
t = t + tstep; 更新时间，表示上述动作完成的时间。
}

推演结束后，即可得到各车辆是否在给定的时间 D 内到达目的地，以及具体的到达时刻。通过给定的公式即可求解得到总得分。

4.1.2 交通信号灯与所属路口建模

针对附录文件中道路数量多，组网密度大的特点，现有智能算法无法对超高维的设计变量进行实验设计与优化，将每个交通信号灯时序作为全局优化的设计变量无疑会导致“维度灾难”。因此，本文采用分层级优化的策略，充分利用车流与路网信息，将路口进行动态分层级建模，层级与路口情况复杂度正相关。通过这一策略达到降维降阶和降低模型复杂度的效果。

下面对模型进行分级处理。对于没有车经过的路口和没有车经过的路不需要

进行建模，下面仅对有车辆通过的路网进行分级建模。分级的基本依据为：经过该路口的车辆数与所涉及的道路数的相对关系。

1.车辆数与道路数相等（m=n）

该情况下可通过仿真计算出理想情况下每辆车到达该路口的最短时间，按照先后顺序可对信号灯绿灯亮的次序进行排序，以保证在理想情况下，该车到达路口时可顺利通过。极端情况下，该情况的信号灯时序的周期可大于最后一辆车到达的时刻。若有两辆车理想情况下同时到达该路口，则按车辆序号排序。结合附件中的具体情况，此情况下的仅有一个路口起步时有车同时处于同一路口需要等待一秒，其他路口完全可以避免车辆等待。

2.车辆数略大于道路数（m-n<=3）

该情况下同样计算车辆到达该路口的理想时间，根据先后顺序对道路进行排序。判断同道路相邻两车间是否有其他车辆需要驶过该路口，若没有则该情况可降级成为上一等级的情况，即多辆同道路且相邻通过路口的车辆可视为一个车辆，绿灯时段应覆盖该部分车所到达的时间节点。若不存在上述情况，则在 2m~4m 的区间内对周期进行寻优，对每个到达时刻取周期的余数，当同一路上的车辆到达时刻的余数符合相邻的情况则可转化为同一车辆的情况，该情况下同样可极大程度避免车辆等待。

3.车辆数大于道路数（m-n>3）

虽然在附录中给的情况下不存在此种复杂情况，为了加强模型的全局通用性和鲁棒性，对这种情况采用车流概率密度的概念，从统计角度模糊处理，对车辆多的道路在同一周期内给予更长的绿灯时间。通过采用打靶法等方法，对模型进行仿真并对比，得到绿灯时间按照正比于车流密度的情况车辆等待时间更小的结论。

经过上述分类，即可对各路口的绿灯亮起时长和顺序进行调控。

4.2 问题 2

问题 2 的主要任务是在给定路网结构、车辆行程计划的情况下，考虑某时刻某条道路上突发道路设施故障，并导致道路无法继续通行时，分析讨论交通灯调度受影响的情况，并设计局部调度方案。其中，受影响而不能通行的车辆按照里程最短原则重新选择后续行驶路径，其它车辆不作调整。针对该问题，本文从以下两方面考虑：

(1) 搜索受影响车辆的最短行驶路径。分析故障路段对交通的影响情况，考虑经过该路段的车辆，以及经过的时刻等信息，并记录车辆在到达故障路口后的起始位置和目标位置，按照行程最短原则重新规划路径。

(2) 求解局部交通灯调度方案。考虑车辆路线调整问题，重新设计局部路口的交通灯周期，相应各道路的绿灯亮起顺序和时长。

下面分别展开说明。

4.2.1 最短路径搜索

考虑到给定的数据中没有路口的平面几何分布关系，难以根据数据确定最短路径的最佳搜索方向，因此，本文采用树搜索策略[1]-[4]寻找最短路径。针对某给定的需要调整路线的车辆，从故障路口出发，搜索所有出发的道路，形成树形结构的数据。通过比较树枝末端的编号与车辆目的地路口编号，即可寻找到满足

要求的路线。通过对所有求得的路线的长度，即可确定最短路径。

在进行数搜索时，需要对数据结构和搜索策略进行详细设计。本文考虑采用递归结构体储存数据，并使用递归函数进行搜索[5][6]。

4.2.2 局部交通灯调控

局部交通灯调控与全局类似，同样是分情况讨论，与 4.2.1 节类似，这里不再赘述。

五 模型求解

5.1 问题 1

下面针对第四节给出的模型，对问题进行求解。在完成问题 1 的推演模型后，可以对题目给定的数据进行简单分析，并尽可能找到一些规律。下面以两个工况为例进行分析：

工况 1：各道路的绿灯亮起时长设为固定值。

设各有车经过的道路的绿灯亮起时长为 t_fix，是 1 秒的正整数倍。因此，各路口的交通灯周期就可以根据有车经过的道路数进行判断。同时，考虑直接按路口上各道路的 ID 从小到大的顺序来确定各道路绿灯亮起的顺序。在完成上述设置后，即可按照推演流程完成单次推演，得到各车辆的具体行驶信息。通过调整 t_fix 的具体数值，我们发现，当 t_fix = 1 时，总得分最高。这是比较合理的，因为对于道路较多的路口，由于车辆到来的随机性，路口交通灯周期越短，意味着车辆的平均等待时长也就越短。因此，在该工况下，各路口仅 1 秒的绿灯时长将会带来较好的收益。该工况下的总得分为：130858 分。

工况 1 下，各车的得分与车辆的最短路程如图 1 所示，其中 X 轴为车辆编号，Y 轴为得分或路程，蓝色曲线标识车辆最短路程（单位：秒），橙色曲线表示各车辆得分。从图中可以看出，得分与最短路程基本呈对称关系，这也意味着车辆在路口等红灯的时间不会很长。值得注意的是，在 t_fix > 1 的工况中，都至少有 2 辆及以上的车得分为 0，也即没能在规定时间内到达目的地，且 t_fix 越大，得分为 0 的车越多。但在 t_fix = 1 的工况中，仅有一辆车得分为 0，通过分析可以发现该车辆的最短路程就达到了 886 秒，超过了题目数据中给定的最大周期。也就是说，该车即使是全部顺利通过各路口（不等待任何红灯的情况），也不能按时到达终点。

通过工况 1 的分析可以得到以下结论：

- 题目给定的数据中，存在一辆车不可能实现在固定路线的情况下按时到达终点；
- 在该种工况下，路口交通灯周期越短，车辆延误时长就越短，相应的总得分就最大。

需要说明的是，上述工况与相应结论仅针对题目给定的数据，数据中存在个别路口上道路较多的情况，以及个别道路上经过的车辆较多的情况，如果相应道路上的绿灯时间仍然很短，则该道路上的多辆车都可能受到影响。因此，我们继续分析了下一个工况。

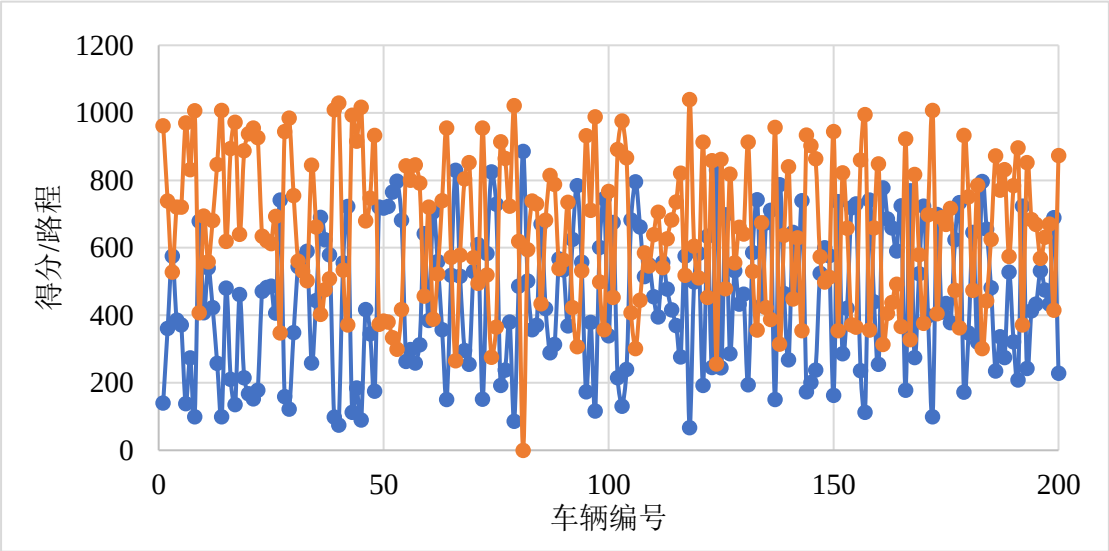

图 1 工况 1 车辆得分与最短路程

工况 2：按车流数给定道路路灯时长

在工况 1 的基础上，我们对各道路的路灯时长进行微调，将其数值设置为该道路上经过的总车辆数，其余参数保持不变。例如，道路 1 以路口 0 为终点，经过道路 1 的车辆的总数为 x，则道路 1 的绿灯时长设置为 x。这样设置的目的是，确保在相应路口上，来车较多的道路上的车到达时，能够以较大的概率遇到绿灯。通过上述设置，采用推演模型进行计算，最终总得分为：131092 分，比工况 1 高出 234 分。

同样地，可以画出相应的车辆得分与最短路程，如图 2 所示。其中 X 轴为车辆编号，Y 轴为得分或路程，蓝色曲线标识车辆最短路程（单位：秒），橙色曲线表示各车辆得分。从图中可以看出，得分与最短路程仍然基本呈对称关系，与图 1 非常相近，且也是同一辆车不能按时到达终点。

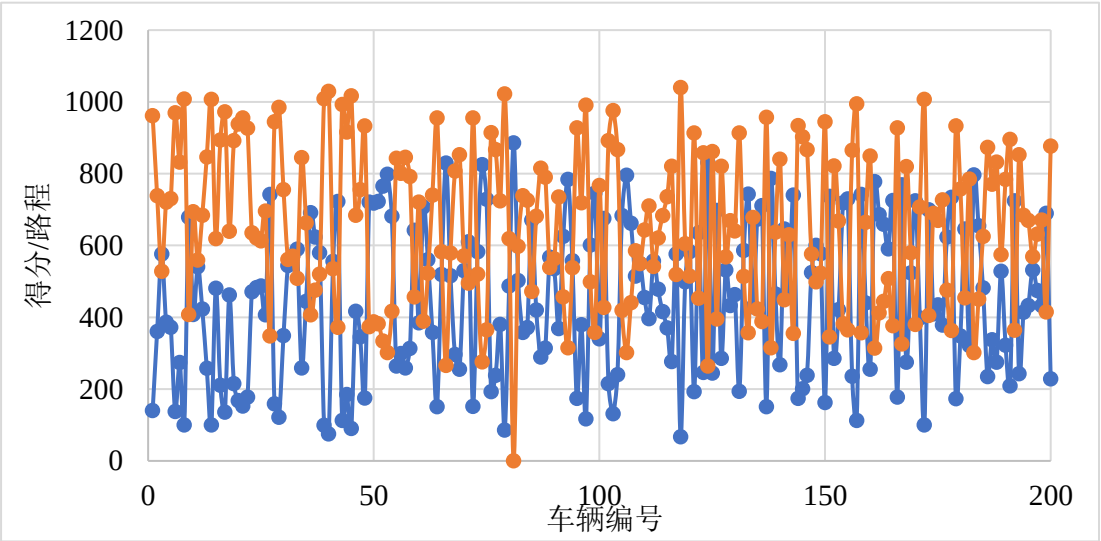

图 2 工况 2 车辆得分与最短路程

进一步，我们可以画出两种工况下车辆得分差值的关系（工况 2 减工况 1），图图 3 所示，其中 X 轴为车辆编号，Y 轴为得分。从图 2 中可以看出，大多数

情况下，工况 2 的得分是超过工况 1 的（在 X 坐标轴以上），但极少数情况下，工况 2 的得分会明显低于工况 1。

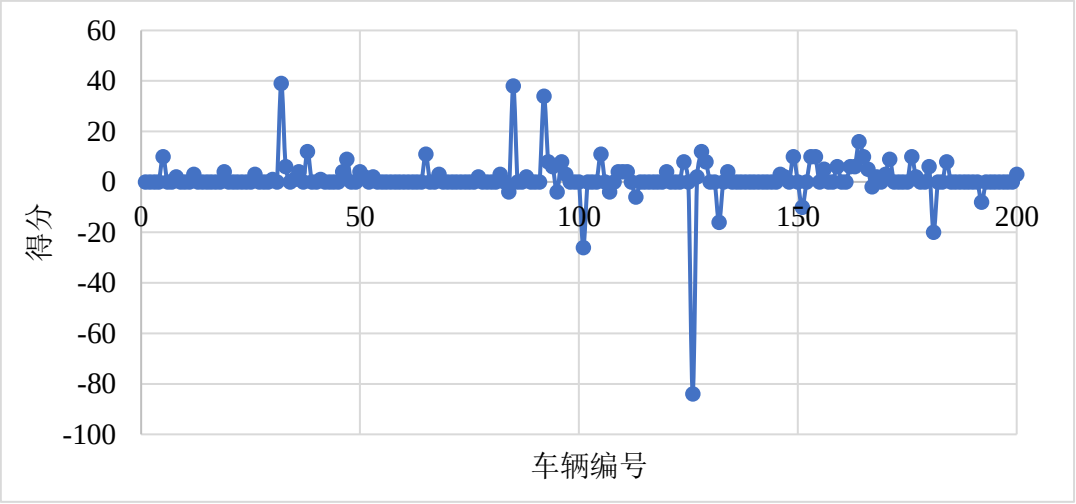

图 3 两种工况车辆得分差（工况 2-工况 1）

因此，通过工况 2 的分析可以得到以下结论：

- 按照道路上通车数量给道路设置绿灯时长时，相比给每个路口设置相等的绿灯时长的情况，能够提高车短时间通过路口的概率，带来更高收益。
- 少数情况下，工况 2 的车辆得分明显低于工况 1，这也意味着直接按来车数设置绿灯时长仍然有较大的优化空间。

5.2 问题 2

采用第 4 节中问题 2 的模型，对问题进行分析，可以求得需要修改形成的车有三辆，ID 分别为 C016，C035 以及 C050，依次对这些车辆的进行搜索即可得到若干条新的路径。需要指出的是，新路径的条数与搜索时递归的嵌套层数密切相关，层数越多，搜索的路径就越多，相应的路径总长度也越长。经过搜索，可以找到三辆车的最短路径，列举如下：

车辆 ID	最优路径
C016	dbh-fjai,fjai-eggf,eggf-ebbe,ebbe-ebi,ebi-ghfc,ghfc-edaj,edaj-ebf,ebf-dgj,dgj-ba,ba-bdce,bdce-ciig,ciig-ffef,ffef-bhej
C035	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-bbgb,bbgb-degj,degj-cddh
C050	gg-djjc,djjc-cede,cede-dci,dci-ebf,ebf-ha,ha-badf,badf-bfda,bfda-je,je-hhfd,hhfd-bjfh

参考文献

[1] Friedl M A， Brodley C E . Decision tree classification of land cover from remotely sensed data. Remote Sensing of Environment,61(3):399-409, 1997.

[2] Fayyad U M， Irani K B . On the Handling of Continuous-Valued Attributes in Decision Tree Generation[J]. Machine Learning,8(1):87-102, 1992.

[3] Pei-Hua, Wang, Yi-Shu, et al. PgpRules: a Decision Tree Based Prediction Server for P-glycoprotein Substrates and Inhibitors.[J]. Bioinformatics (Oxford, England), 2019.

[4] Samantaray S R . Decision tree-based fault zone identification and fault classification in flexible AC transmissions-based transmission line[J]. Iet Generation Transmission & Distribution,3(5):1-6, 2015.

[1] Ong C . On Model-Checking Trees Generated by Higher-Order Recursion Schemes[J]. Proceedings - Symposium on Logic in Computer Science,81-90, 2006.

[2] Hughes J， Pareto L . Recursion and Dynamic Data-structures in Bounded Space: Towards Embedded ML Programming[J]. ACM SIGPLAN Notices, 34(9):70-81,1999.

附　录

前四部分为 matlab 函数，后面为 C++。

1.根据路口 ID 给出绿灯时序，dt 为时间裕度

```
function seq=t_seq(cross_id,dt)
dt1=1;
cross_data=cross_info('CrossCarStreetT.txt',cross_id);
for k=1:10
    cross_data=sortrows(cross_data,6);
    temp=1;
    Same_post=find_same(cross_data(:,6));
    if ~isempty(Same_post)
        temp=0;
        temp0=1;
        for i=1:size(unique(Same_post(:,4)),1)

            temp1=Same_post(temp0,3);

            for j=1:temp1-1
                temp2=Same_post(temp0+temp1-j,1);
                cross_data(temp2,6)=cross_data(temp2,6)+temp1-j;

            end
            temp0=temp0+temp1;
```

```
            end
        else
            temp=1;
        end

        if temp
            break
        end
    end
    %       temp=[];
    %       for i=1:Same_post(1,3)
    %           temp=cross_data()
    m=cross_data(1,2);
    n=cross_data(1,3);
    seq=zeros(m,5);
    seq(:,1)=cross_id;

    if m==n
        seq(:,2)=cross_data(:,5);
        t=0;
        for i=1:m-1
            if cross_data(i,6)+dt>cross_data(i+1,6)
                seq(i,3:4)=[t,cross_data(i,6)-t];
                t=cross_data(i,6);
            else
                seq(i,3:4)=[t,cross_data(i,6)+dt-t];
                t=cross_data(i,6)+dt;

            end
        end
        seq(m,3:4)=[t,cross_data(m,6)-t];
        seq(end,4)=seq(end,4)+dt;
        seq(:,5)=cross_data(m,6)+dt;
    else
        Same_post=find_same(cross_data(:,5));
        flag=0;
        temp0=1;
        for i=1:Same_post(end,end)
            temp1=Same_post(temp0,3);
            if Same_post(temp0+temp1-1,1)-Same_post(temp0,1)<=temp1
                flag=flag*10;
                for j=1:temp1-1
```

```
                cross_data(Same_post(temp0+temp1-1,1)-j,:)=[];
            end
        else
            flag=flag+1;
        end
        temp0=temp0+temp1;
    end
    if flag==0
        t=0;
        for i=1:m-1
            if cross_data(i,6)+dt1>cross_data(i+1,6)
                seq(i,3:4)=[t,cross_data(i,6)-t];
                t=cross_data(i,6);
            else
                seq(i,3:4)=[t,cross_data(i,6)+dt1-t];
                t=cross_data(i,6)+dt1;
            end
        end
        seq(:,2)=cross_data(:,5);
        seq(m,3:4)=[t,cross_data(m,6)+dt1-t];
        seq(:,5)=cross_data(m,6)+dt1;
    else
        seq=get_period(cross_id,1);

    end

end
```

2.问题1 中等级2和3的模型

```
function seq=get_period(cross_id,dt)
cross_data=cross_info('CrossCarStreetT.txt',cross_id);
cross_data_bak=cross_data;
m=cross_data(1,2);
n=cross_data(1,3);
seq=zeros(m,5);
seq(:,1)=cross_id;
% cross_data=sortrows(cross_data,6);
for p=2*n:4*n
    cross_data=cross_data_bak;
    cross_data=sortrows(cross_data,6);
    cross_data(:,6)=mod(cross_data(:,6),p);
    cross_data=sortrows(cross_data,6);
```

```
    Same_post=find_same(cross_data(:,5));
    temp=find_same(cross_data(:,6));
    tag0=0;
    tag1=0;
%       temp1=1;
    if isempty(temp)
        temp0=1;%Ö¸Õë
        for i=1:Same_post(end,end)
            flag=0;
            temp1=Same_post(temp0,3);
            for j=1:temp1-1
                if Same_post(temp0+j,1)-Same_post(temp0+j-1,1)==1
                    flag=flag+1;
                end
            end
            tag0=tag0+flag;
            tag1=tag1+temp1-1;
%               if flag==temp1-1
%                   break
%               end
            temp0=temp0+temp1;
        end
        if tag0==tag1
            break
        end
    end
end
if tag0==tag1 && p~=4*n
        cross_data=cross_data_bak;
    cross_data=sortrows(cross_data,6);
    cross_data(:,6)=mod(cross_data(:,6),p);
    cross_data=sortrows(cross_data,6);

    Same_post=find_same(cross_data(:,5));
%       flag=0;
    temp0=1;%Ö¸Õë
    for i=1:Same_post(end,end)
        temp1=Same_post(temp0,3);
        if Same_post(temp0+temp1-1,1)-Same_post(temp0,1)<=temp1
%               flag=flag*10;
            for j=1:temp1-1
                cross_data(Same_post(temp0+temp1-1,1)-j,:)=[];
```

```
                end
                temp0=temp0+1-temp1;%Ö¸ÕëÖØÐÂ±ê¶¨
            else
%                   flag=flag+1;
            end
            temp0=temp0+temp1;
        end

        t=0;
        for i=1:m-1%³£¹æÅÅÐò
            if cross_data(i,6)+dt>cross_data(i+1,6)
                seq(i,3:4)=[t,cross_data(i,6)-t];
                t=cross_data(i,6);
            else
                seq(i,3:4)=[t,cross_data(i,6)+dt-t];
                t=cross_data(i,6)+dt;
            end
        end
        seq(:,2)=cross_data(:,5);
        seq(m,3:4)=[t,cross_data(m,6)-t];
        seq(:,5)=p;
    else
        cross_data=cross_data_bak;
        cross_data=sortrows(cross_data,5);
        Y=unique(cross_data(:,5));
        Same_post=find_same(cross_data(:,5));
        X=[];
        Y(:,2)=1;
        temp0=1;
        for i=1:Same_post(end,end)
            Y(Same_post(temp0,1),2)=Same_post(temp0,3);
            temp0=temp0+Same_post(temp0,3);
        end
        Y(:,3)=Y(:,2);
        Y(:,2)=0;
        for j=1:m-1
            Y(j+1,2)=Y(j,2)+Y(j,3);
        end
      seq(:,2:4)=Y;
      seq(:,5)=n;
    end
```

3.读取道路网状态信息

```
function cross_info=cross_info(file_name,cross_id)
prmt_temp=textread(file_name);
flag=find(prmt_temp(:,1)==cross_id);
flag_id=flag(1,1);
temp=prmt_temp(flag_id,3);
flag_end=flag_id+temp-1;
cross_info=prmt_temp(flag_id:flag_end,:);
```

4.生成全路口信号灯时序，保存在designVec.txt，用来和C++进行交互。

```
temp=textread('CrossCarStreetT.txt');
X=unique(temp(:,1));

fid_output=fopen('designVec.txt','wt');
for i=1:size(X)
    t=t_seq(X(i),10);

    fprintf(fid_output,'%d %d %d %d %d\n',t');

end
fclose(fid_output);
```

C++代码

```
// 仿真模型部分：
// 读取 txt 文件 初始化参数
bool TrafficLightPlan::LoadFileIni()
{
    // 读取场景参数
    ifstream inFile1(".\\InitialData\\parameters.txt", ios::in);
    if (!inFile1)           // txt 文件不存在
    {
        return false;
    }
    else
    {
        inFile1>>m_tTotal;
        inFile1>>m_crossNum;
        inFile1>>m_streetNum;
        inFile1>>m_carNum;
        inFile1>>m_F;

    }
```

```
inFile1.close();

int i, j, k;
// 读取道路参数
m_street.resize(m_streetNum);
ifstream inFile2(".\\InitialData\\streets.txt", ios::in);
if (!inFile2)            // txt 文件不存在
{
    return false;
}
else
{
    string lineData;
    inFile2>>lineData;        // 读取标题行
    for (i=0; i<m_streetNum; i++)
    {
        inFile2>>lineData;        // 读取当前行信息
        stringstream input(lineData);
        getline(input, m_street[i].m_name, ',');
        m_street[i].m_ID = i;            // 人为给定数字 ID
        //m_street[i].m_tstep = m_tstep;
        //m_street[i].m_t = 0;
        string t_data;
        getline(input, t_data, ',');
        m_street[i].m_staCroID = stoi(t_data); //AsStrToInt
        getline(input, t_data, ',');
        m_street[i].m_endCroID = stoi(t_data);
        getline(input, t_data, ',');
        m_street[i].m_len = stoi(t_data);
    }
}
inFile2.close();

// 读取车辆参数
m_car.resize(m_carNum);
m_M.resize(m_carNum);
ifstream inFile3(".\\InitialData\\cars.txt", ios::in);
if (!inFile3)            // txt 文件不存在
{
    return false;
}
else
```

```
{
    for (i=0; i<m_carNum; i++)
    {
        string lineData;
        inFile3>>lineData;        // 读取当前行信息
        stringstream input(lineData);
        getline(input, m_car[i].m_name, ',');
        m_car[i].m_ID = i;        // 人为给定数字 ID
        m_car[i].m_tstep = m_tstep;
        m_car[i].m_t = 0;
        string t_data;
        getline(input, t_data, ',');
        m_car[i].m_passStreetNum = stoi(t_data);
        m_car[i].m_passStreetName.resize(m_car[i].m_passStreetNum);
        m_car[i].m_passStreetID.resize(m_car[i].m_passStreetNum);
        m_car[i].m_passStreetLen.resize(m_car[i].m_passStreetNum);
        m_car[i].m_passStreetWT.resize(m_car[i].m_passStreetNum);
        for (j=0; j<m_car[i].m_passStreetNum; j++)
        {
            getline(input, m_car[i].m_passStreetName[j], ',');
            // 找到路口 ID 和长度
            for (k=0; k<m_streetNum; k++)
            {
                if (m_car[i].m_passStreetName[j] == m_street[k].m_name)
                {
                    m_car[i].m_passStreetID[j] = m_street[k].m_ID;
                    m_car[i].m_passStreetLen[j] = m_street[k].m_len;
                    m_car[i].m_totalMinDis += m_street[k].m_len;
                }
            }
        }
        // 初始时刻车辆位置信息
        m_car[i].m_tStreetOrder = 0;
        m_car[i].m_tStreetPos = m_car[i].m_passStreetLen[0];
        m_car[i].m_totalMinDis -= m_car[i].m_passStreetLen[0];  // 起始段直接在路终
点
        m_car[i].m_totalMinDis += (m_car[i].m_passStreetNum-1)*m_tstep;// 加上除最
后一段的过路口时长
        //cout<<m_car[i].m_totalMinDis<<" ";
    }
}
inFile3.close();
```

```
// 路口信息初始化
int ii, jj;
m_cross.resize(m_crossNum);
m_crossDenNum = 0;
m_designVarNum = 0;
for (i=0; i<m_crossNum; i++)
{
    m_cross[i].m_ID = i;
    m_cross[i].m_existCarStreNum = 0;
    m_cross[i].m_totalCarNum = 0;
    // 查询以该路口为终点的道路，挑选有车辆经过的道路，并记录道路 ID
    for (j=0; j<m_streetNum; j++)
    {
        if (m_street[j].m_endCroID == m_cross[i].m_ID)
        {
            //m_cross[i].m_streetName.push_back(m_street[j].m_name);
            //m_cross[i].m_streetID.push_back(m_street[j].m_ID); // 记录所有以该路口为终点的道路 暂时用不上
            // 查询经过该道路的车辆数
            int tt_data = 0;
            vector<int> t_vec, t_vec2;
            for (k=0; k<m_carNum; k++)
            {
                for (ii=0; ii<m_car[k].m_passStreetID.size()-1; ii++)        // 最后一条道路不需寻找，因为到最后路口就停车了
                {
                    if (m_car[k].m_passStreetID[ii] == m_street[j].m_ID)
                    {
                        tt_data += 1;
                        t_vec.push_back(m_car[k].m_ID);
                        // 求汽车前面经过的长度 第一段路直接在终点 中间需要加过路口时长
                        int len = 0;
                        for (jj=1; jj<=ii; jj++)
                        {
                            len +=   m_car[k].m_passStreetLen[jj]+m_tstep;
                        }
                        t_vec2.push_back(len);
                    }
                }
            }
```

```
                    // 如果有车经过 则记录道路与车信息
                    if (tt_data != 0)
                    {
                        m_cross[i].m_existCarStreNum += 1;        // 表示以该路口为终点的车辆会经过的道路数
                        m_cross[i].m_streetID.push_back(m_street[j].m_ID); // 只有有车辆经过的路口 ID 才需要记录!!!
                        m_cross[i].m_streetCarNum.push_back(tt_data);      // 只记录有车经过的路口上的车辆数
                        m_cross[i].m_totalCarNum += tt_data;
                        m_cross[i].m_streetCarID.push_back(t_vec);
                        m_cross[i].m_streetCarLen.push_back(t_vec2);
                    }
                }
            }
            //m_cross[i].m_streetNum = m_cross[i].m_streetID.size();
            if ( m_cross[i].m_existCarStreNum == 0)          // 没有车经过该路口
            {
                m_cross[i].m_lightBeDesign = false;
                m_cross[i].m_lightPeriod = 0;                // 全程红灯
            }
            else if ( m_cross[i].m_existCarStreNum == 1)          // 仅有一条道路有车输入
            {
                m_cross[i].m_lightBeDesign = false;
                m_cross[i].m_lightPeriod = m_tTotal;  // 全程绿灯
            }
            else                                             // 有至少两条有车的路经过路口
            {
                m_cross[i].m_lightBeDesign = true;
                m_crossPtDesnID.push_back(m_cross[i].m_ID);            // 记录这些路口 ID
                m_crossStrtNum.push_back(m_cross[i].m_existCarStreNum);  // 记录这些路口存在车的道路数
                m_crossDenNum += 1;
                m_designVarNum += 1; // 该路口红绿灯周期
                m_designVarNum += 2*m_cross[i].m_existCarStreNum;  // 该路口中有车的道路的绿灯开始时刻和时长（一个周期内）
                //cout<<m_cross[i].m_existCarStreNum<<" ";
            }
        }
        // 统计车辆路径和
        m_minDisTol = 0;
        for (i=0; i<m_carNum; i++)
```

```
    {
        m_minDisTol += m_car[i].m_totalMinDis;
    }
    cout<<"总设计变量："<<endl<<m_crossDenNum<<endl;
    cout<<m_designVarNum<<endl;
    return true;
}

// 根据输入的设计变量计算目标值
int TrafficLightPlan::CalTarget(const vector<int> &designVec)
{
    // 初始化得分情况
    m_tar = 0;
    m_M.resize(m_carNum);

    int t = 0;
    int i, j, k, ii, jj;
    // 先判断输入的设计变量是否满足约束
    // 判断每个路口绿灯周期是否重叠 或超过周期
    j = 0;
    int t_cID, t_sID;
    for (i=0; i<m_crossDenNum; i++)
    {
        t_cID = m_crossPtDesnID[i];       // 当前路口 ID
        m_cross[t_cID].m_lightPeriod = designVec[j];    // 当前路口红绿灯总周期
        int t_period = designVec[j];
        vector<int> t_data1, t_data2;
        t_data1.resize(m_crossStrtNum[i]);
        t_data2.resize(m_crossStrtNum[i]);
        for (k=0; k<m_crossStrtNum[i]; k++)  // 当前路口中有车输入的道路 道路上各自绿灯的开始时刻和时长
        {
            t_sID = m_cross[t_cID].m_streetID[k]; // 当前道路 ID
            t_data1[k] = designVec[j+1+2*k];
            t_data2[k] = designVec[j+2+2*k];
            m_street[t_sID].m_lightSta = designVec[j+1+2*k];
            m_street[t_sID].m_lightLen = designVec[j+2+2*k];
            //cout<<t_sID<<" ";
        }
        int tt_data;
        // 对当前路口各路段红绿灯顺序进行排序
        for (ii=0; ii<m_crossStrtNum[i]; ii++)
```

```
        {
            for (jj=m_crossStrtNum[i]-1; jj>ii; jj--)
            {
                if (t_data1[jj] < t_data1 [ii])
                {
                    tt_data = t_data1[ii];
                    t_data1[ii] = t_data1[jj];
                    t_data1[jj] = tt_data;
                    tt_data = t_data2[ii];
                    t_data2[ii] = t_data2[jj];
                    t_data2[jj] = tt_data;
                }
            }
        }
        // 判断是否满足路口绿灯间隔约束 不满足则直接返回
        for (ii=0; ii<m_crossStrtNum[i]; ii++)
        {
            if (t_data1[ii] >= t_period || t_data1[ii]+t_data2[ii] > t_period)   // 有任何一段区间超过周期
            {
                return m_punish;
            }
            if (ii<m_crossStrtNum[i]-1 && (t_data1[ii]+t_data2[ii] > t_data1[ii+1]))  // 有任何两个区间发生重叠
            {
                return m_punish;
            }
        }
        j = j+m_crossStrtNum[i]*2+1;
    }
    //cout<<endl<<endl;

    // 再进行推演
    while (t <= m_tTotal)
    {
        // 车辆更新一步
        for (i=0; i<m_carNum; i++)
        {
            m_car[i].UpdateCar();   // 更新车辆位置
            if (!m_car[i].m_beEnd)  // 如果车辆没有到达终点
            {
                // 车辆如果到路口 则相应路口开始排队
```

```
                if (m_car[i].m_beCross)
                {
                    t_sID = m_car[i].m_passStreetID[m_car[i].m_tStreetOrder];          //
当前车所处道路 ID
                    //cout<<t_sID<<" ";
                    m_street[t_sID].m_lineCarID.push_back(m_car[i].m_ID);
    // 车辆所在道路上排队序列增加一辆车
                    // 判断该路口绿灯亮起情况，决定是否释放一辆车
                    t_cID = m_street[t_sID].m_endCroID;              // 当前道路尽头的路
口 ID
                    int                t_ret                =                t-
m_cross[t_cID].m_lightPeriod*floor(t*1.0/m_cross[t_cID].m_lightPeriod);     // 将时间约束到
当前路口红绿灯周期内
                    if         (!m_cross[t_cID].m_lightBeDesign         ||         (t_ret         >=
m_street[t_sID].m_lightSta && t_ret < m_street[t_sID].m_lightSta+m_street[t_sID].m_lightLen))
        // 如果该路口上，该道路此时正好绿灯 或者该路口不需要设计 则通行
                    // m_cross[t_cID].m_lightPeriod == m_tTotal)//
                    {
                        int t_carID = m_street[t_sID].m_lineCarID[0];          // 排在最前
面的一辆车的 ID
                        m_car[t_carID].m_tStreetOrder += 1;                    // 将车
更新到下一条路
                        m_car[t_carID].m_tStreetPos = 0;                       // 下一条路
的起始位置
                        m_car[t_carID].m_beCross = false;                      // 标记
已经离开路口

    m_street[t_sID].m_lineCarID.erase(m_street[t_sID].m_lineCarID.begin()+0);  // 删除排在
最前面的一辆车
                    }
                }
            }
            else // 记录该车辆到达终点的时间 计算该车的得分
            {
                if (m_car[i].m_t <= m_tTotal && m_M[i] == 0)
                {
                    m_M[i] = m_F+(m_tTotal-m_car[i].m_t);
                    m_tar += m_M[i];
                }
            }
        }
        t += m_tstep;
```

```
    }
    return (m_tTotal+m_F)*m_carNum-m_minDisTol-m_tar;          // 最大得分-当前得分 最
小化该值
}

// 树搜索部分
// 读取已有场景文件后 在给定位置处断开，搜索最佳路径
void TrafficLightPlan::SearchStreet()
{
    // 找到经过断点的 道路 ID 车 ID 所处路口 ID 目标位置道路 ID 搜索连线
    string stretPName = "ebf-ffef";
    int stretPID, crossPID;
    int i, j;
    // 找到断开的道路 ID 以及 经过该道路的车
    vector<Car> passPcar;
    vector<int> passPcarStrtOrdP;      // 断开后 原道路剩余部分起点在原序列上的位置
    for (i=0; i<m_carNum; i++)
    {
        for (j=0; j<m_car[i].m_passStreetNum; j++)
        {
            if (m_car[i].m_passStreetName[j] == stretPName)
            {
                stretPID = m_car[i].m_passStreetID[j];        // 当前道路 ID
                crossPID = m_street[stretPID].m_staCroID; // 断开的道路的起点处路口 ID
                passPcar.push_back(m_car[i]);
                passPcarStrtOrdP.push_back(j);
            }
        }
    }

    // 搜索单量车的最短路径
    SearchSigCarStreet(passPcar[2], crossPID, stretPID);


}


void TrafficLightPlan::SearchSigCarStreet( const Car t_car, const int crossPID, const int stretPID )
{
    // 记录该车原来路线上剩余路口的 ID
    vector<int> cIDtoBeFind;
    cIDtoBeFind.push_back(m_street[t_car.m_passStreetID[t_car.m_passStreetNum-
```

```
2]].m_endCroID);  //暂时考虑直接找终点路口

    //// 给定 ID 往后搜索可能的道路 递归搜索 搜索一定的代数就停下来
    int i, j;
    m_strListNode.nodeLevel = 0;
    m_strListNode.nodeCID = crossPID;
    m_maxSearchLevel = 5;       // 最大搜索层数
    vector<string> newPassStrHis;
    newPassStrHis.resize(m_maxSearchLevel-1);
    SearchNextSLN(m_strListNode, m_strListNode.nodeLevel, crossPID, cIDtoBeFind,
newPassStrHis, stretPID);

    // 保存当前车所有可能更改的路径 注意加上第一次就能搜到的
    int t_len, k;
    for (i=0; i<m_newPassStrList.size(); i++)
    {
        t_len = 0;
        for (j=0; j<m_newPassStrList[i].size(); j++)
        {
            cout<<m_newPassStrList[i][j]<<"    ";
            for (k=0; k<m_streetNum; k++)
            {
                if (m_newPassStrList[i][j] == m_street[k].m_name)
                    t_len += m_street[k].m_len;
            }
        }
        cout<<t_len<<endl;
    }
}


void TrafficLightPlan::SearchNextSLN( StreetListNode &sln, const int nodeLevel, const int
crossPID, const vector<int>& cIDtoBeFind, vector<string> &newPassStrHis, const int stretPID )
{
    sln.nodeLevel = nodeLevel;
    sln.nodeCID = crossPID;
    int i, j;
    vector<int> fcID1;
    for (j=0; j<m_streetNum; j++)
    {
        if (crossPID == m_street[j].m_staCroID && j!= stretPID)  // 找到以断开路口为起点
的道路 除了故障道路外
```

```
            {
                fcID1.push_back(m_street[j].m_endCroID);
                sln.pstrList.push_back(m_street[j].m_name);      // 得到父路口到子路口的所有
道路
            }
        }
        sln.strListNode.resize(sln.pstrList.size());    // 后续子节点个数

        // 判断这些道路的终点是否能回到原路线上 并收集能回去的路线
        vector<int> sameCID, fp, bp;
        if (JudgEqual(fcID1, cIDtoBeFind, sameCID, fp, bp)) // 找打了回到原路径的路口 ID
        {
            // 记录所有找到的路径
            for (i=0; i<sameCID.size(); i++)
            {
                sln.pstrEndList.push_back(sln.pstrList[fp[i]]);
            }
        }

        // 如果找到路径就返回
        int ii;
        if (sln.pstrEndList.size() != 0 )//|| nodeLevel > 5)
        {
            cout<<"找到可行路径！"<<endl;
            for (i=0; i<sln.pstrEndList.size(); i++)
            {
                // 记录当前层级以前的点
                vector<string> t_vec;
                for (ii=0; ii<nodeLevel; ii++)
                {
                    t_vec.push_back(newPassStrHis[ii]);
                }
                t_vec.push_back(sln.pstrEndList[i]);
                m_newPassStrList.push_back(t_vec);
            }
            return;
        }
        else if (nodeLevel >= m_maxSearchLevel-1)           // 搜索若干层，如果搜索不下去则
停止
        {
            return;
        }
```

```
    // 进行下一节点判断
    for (i=0; i<sln.strListNode.size(); i++)
    {
        newPassStrHis[sln.nodeLevel] = sln.pstrList[i];
        SearchNextSLN(sln.strListNode[i], nodeLevel+1, fcID1[i], cIDtoBeFind, newPassStrHis,
stretPID);
    }
}


bool TrafficLightPlan::JudgEqual( const vector<int> &fcID, const vector<int> &bcID, vector<int>
&sameCID, vector<int> &fp, vector<int> &bp )
{
    sameCID.clear();
    fp.clear();
    bp.clear();
    int i, j;
    for (i=0; i<fcID.size(); i++)
    {
        for (j=0; j<bcID.size(); j++)
        {
            if (fcID[i] == bcID[j])
            {
                sameCID.push_back(fcID[i]);
                fp.push_back(i);
                bp.push_back(j);
            }
        }
    }
    if (sameCID.size() != 0)
        return true;
    else
        return false;
}
```