报名号 #1274

第三届“ScienceWord 杯”数学中国
数学建模网络挑战赛
承 诺 书

我们仔细阅读了第三届“ScienceWord 杯”数学中国数学建模网络挑战赛的竞赛规则。

我们完全明白，在竞赛开始后参赛队员不能以任何方式（包括电话、电子邮件、网上咨询等）与队外的任何人（包括指导教师）研究、讨论与赛题有关的问题。

我们知道，抄袭别人的成果是违反竞赛规则的,如果引用别人的成果或其他公开的资料（包括网上查到的资料），必须按照规定的参考文献的表述方式在正文引用处和参考文献中明确列出。

我们郑重承诺，严格遵守竞赛规则，以保证竞赛的公正、公平性。如有违反竞赛规则的行为，我们将受到严肃处理。

我们允许数学中国网站(www.madio.net)公布论文，以供网友之间学习交流，数学中国网站以非商业目的的论文交流不需要提前取得我们的同意。

我们的参赛报名号为：1274

参赛队员 (签名)：

队员 1：龚玉婷

队员 2：徐信喆

队员 3：洪杰

参赛队教练员 (签名)：

参赛队伍组别：研究生组

报名号 #1274

第三届“ScienceWord 杯”数学中国

数学建模网络挑战赛

编 号 专 用 页

参赛队伍的参赛号码：（请各个参赛队提前填写好）：

1274

竞赛统一编号（由竞赛组委会送至评委团前编号）：

竞赛评阅编号（由竞赛评委团评阅前进行编号）：

报名号 #1274

2010 年第三届“ScienceWord 杯”数学中国

数学建模网络挑战赛

题　　目　交通网络自适应系统的 Braess 悖论实证研究

关 键 词　Braess 悖论　细胞自动机　多代表性个体　自适应

摘　　要：

本文用随机线性及非线性动态规划方法研究北京二环高峰时段交通堵塞的原因，并用自适应系统的模拟仿真实验验证理论模型。主要结果表明：1. 二环路的道路拥堵符合“Braess 悖论”描述的情况，即当高峰时段二环以内虽然许多路段（东西走向）开通以缓解市区交通压力，但这种做法却降低了整个二环线的交通运行效率，造成多个主要南北走向干道（如文慧桥至复兴门桥、建国门至左安门等路段）交通流的不稳定及车辆拥堵，实际交通效率下降。2. 交通堵塞问题并不会因司机广泛使用 GPS 导航系统而得到缓解，因为司机在缺乏或过分依赖 GPS 导航选择驾驶路线都可能导致“Braess 悖论”问题，司机接受 GPS 导航建议的程度会影响交通网络的运行效率。3.这一结论在细胞自适应系统的仿真模拟实验中得到验证。GPS 导航系统使司机能及时掌握当前路况和最新行车路线，并在此基础上结合自身经验作出判断，虽然提高了临时路段的局部通行能力，却使所有出行者的出行时间都增加了。当 GPS 导航和司机自身的判断达到一定比例时约 10%，全部车辆的行驶时间最接近社会最优的通过时间。所以，“Braess”悖论是否发生不仅和个体是否了解全局状况有关，而且还与司机是否遵循 GPS 建议的最优路线行驶密切相关。

参赛队号 _____

所选题目 _____

参赛密码 _____
（由组委会填写）

英文摘要（选填）

In this paper the causes of traffic jams during peak hours in the second ring of Beijing are analyzed by stochastic dynamic programming (linear and non-linear) and the simulation of self-adaptive multi-agents is carried out to verify our findings. The three conclusions are as follows. First, "Braess Paradox" is found in the traffic blockings of the second ring. The east-to-west roads that are designed to relieve traffic congestion actually cause the instability of car flows on some south-to-north roads (e.g. from Wenhui Bridge to Fuxingmen Bridge, from Jianguomen to Zuoanmen), thus bringing more traffic jams and making the public transportation inefficient. Second, whether the GPS navigation system is helpful in solving the traffic jams depends on the level of acceptance for the drivers. "Braess Paradox" is still possible due to the externality caused by all the drivers. Last, the conclusion is confirmed in the simulation data of a self-adaptive system. If the drivers choose their optimal paths based on both the GPS navigation recommendations and their own experience, the self-adaptive optimization behavior will still suffer from a social welfare loss. However, when the suggestion given by GPS navigation system is denied by 10% of the drivers, the time all the cars take to go though the paths is the closet to the social optimal time spending. In summary, both the traffic situation provided for the drivers and their acceptance of those paths are related with "Braess Paradox" in public transportation.

一、问题重述

1 问题背景

Dietrich Braess 在 1968 年的一篇文章中提出了道路交通体系当中的 Braess 悖论。它的含义是：有时在一个交通网络上增加一条路段，或者提高某个路段的局部通行能力，反而使所有出行者的出行时间都增加了，这种为了改善通行能力的投入不但没有减少交通延误，反而降低了整个交通网络的服务水平。人们对这个问题做过许多研究，在成都市建设当中也尽量避免这种现象的发生。但是在复杂的城市道路当中，Braess 导论仍然不时出现，造成实际交通效率的显著下降。在此，请你通过合理的模型来研究和解决城市交通中的 Braess 悖论。

2 问题提出：

(1)通过健实城市的道路交通情况，建立合理的模型，判断在北京市二球路以内的路网中（包括二环路）出现的交通拥堵，是否来源于 Braess 导论所描述的情况。

(2)请你建立模型以分析：如果司机广泛使用可以反映当前交通拥堵情况的 GPS 导航系统，是否会缓解交通堵塞，并请估计其效果。

二、北京市二环路及相关道路概况

北京市二环路是北京市第一条环城快速公路，于 1992 年 9 月建成通车，它是我国第一条全封闭、全立交、没有红绿灯的城市快速环路。二环路处于北京道路路网的核心位置，围绕旧城而建，全长 32.7 公里。沿线共建朝阳门桥、建国门桥、东便门桥、广渠门桥、光明桥、左安门桥、玉蜓桥、永定门桥、陶然桥、右安门桥、菜户营桥、广安门桥、天宁寺桥、复兴门桥、阜成门桥、官园桥、西直门桥、积水潭桥、安定门桥、雍和宫桥、东直门桥、东四十条桥等关键键的交通枢纽。具体路线见图 1。

三、北京市二环路的 Braess 悖论模型

为了考查北京二环路的拥堵是否和 Braess 悖论有关，我们先考虑一个简单的 Braess 网络模型。

1 基于北京市二环路的 Braess 悖论网络模型

1.1 Braess 悖论网络模型

我们假设北京市二环路的客流量均以文慧门为行程起点进入，以左安门作为行程终点，并且在中途不会经过其他匝道下二环。司机们现有两种走法，一种是逆时针行车，经复兴门到达左安门离开二环线，别一种是顺时针行车经建国桥到达左安门。若选择第一种路线，则文慧门到复兴门的距离约为 5km，复

兴门至左安门的距离约为 11km。若选择第二条路线，则文慧门到建国桥的距离

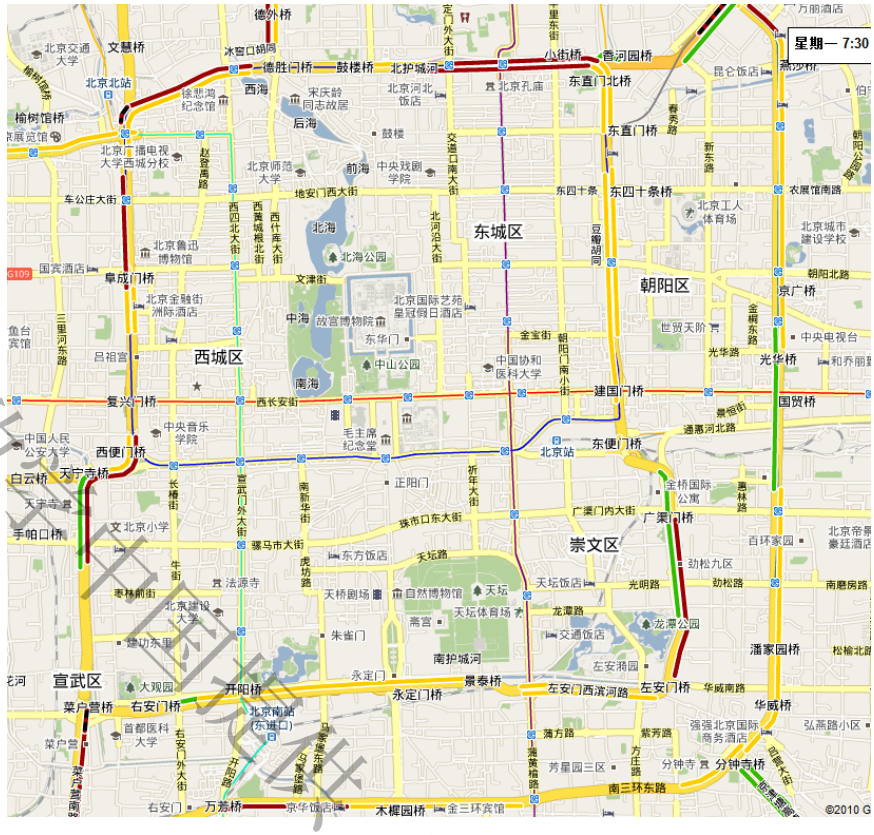

图 1 北京市二环路交通流量图

为 12km，从建国桥到左安门的距离为 4km。现在假设行车时，在某一段的行驶时间与当时处于该段道路上的车流量成正比。这样，可以假设在文慧门至复兴路段和建国桥至左安门段的行驶时间均为 $a_1+b_1n_1$其中 n_1为当时处于该路段的车辆数目。同样，假设文慧门至建国桥段和复兴门至左安门段的平均行驶时间为 $a_2+b_2n_2$，其中 n_2为当时处于该路段的车辆数目。可以得到以下的 Braess 网络模型图：

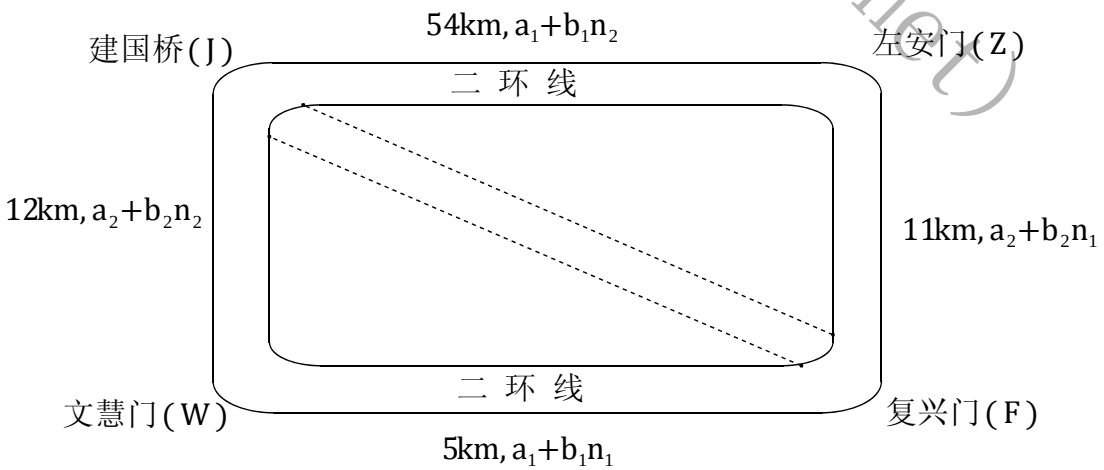

图 1 北京市二环线示意图(匝道不开放)

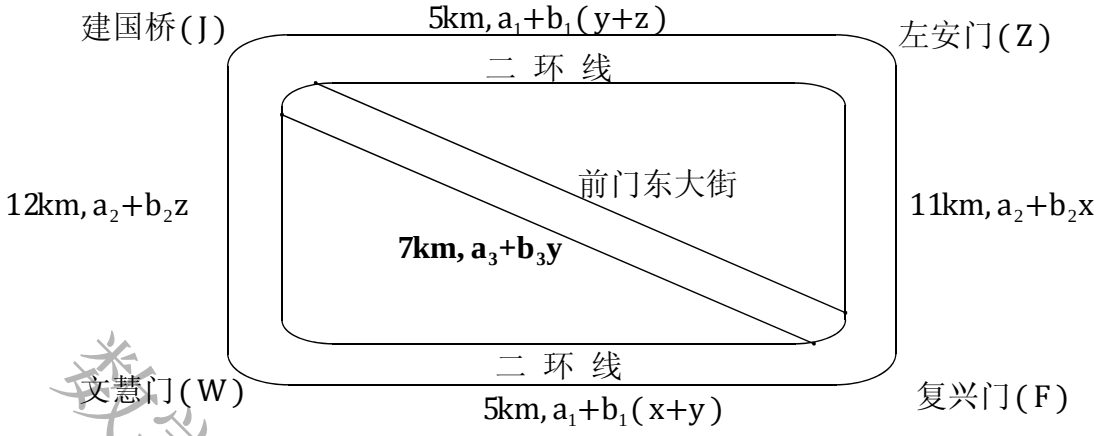

图 3 北京市二环线示意图(匝道开放)

现在的问题是：

第一，假设有 N 辆汽车从文慧门开往左安门，如何在二环上分配车辆才能使每辆汽车的运行时间最短？(如图 2)

第二，如果在复兴门至建国桥的匝道开放，可以在复兴门经匝道下二环线，选择经前门东大街到建国桥(距离为 7km)，再上匝道回到二环，从建国桥至左安门，情况又会如何？(如图 3)

第一个问题比较简单，我们可以建立以下的线性最优化模型：

为方便起见，我们可以将文慧门简记为 W，建国桥简记为 J，复兴门简记为 F，左安门简记为 Z。那么可以得到 N 辆车的总行驶时间为：

$T=(a_1+b_1n_1+a_2+b_2n_1)n_1+(a_2+b_2n_2+a_1+b_1n_2)n_2$，因此，我们将最小化 T 作为我们的目标。由此得到一个简单的规划问题。目标函数是

$$\text{Min} \quad T=(a_1+b_1n_1+a_2+b_2n_1)n_1+(a_2+b_2n_2+a_1+b_1n_2)n_2$$

约束条件为

$$\text{s.t.} \quad n_1+n_2=N$$

构造 Lagrange 函数

$$L(n_1,n_2,\lambda)=(a_1+b_1n_1+a_2+b_2n_1)n_1+(a_2+b_2n_2+a_1+b_1n_2)n_2+\lambda(N-n_1-n_2)$$

求偏得到极值条件为：

$$\frac{\partial L}{\partial n_1}=a_1+a_2+2(b_1+b_2)n_1-\lambda=0$$

$$\frac{\partial L}{\partial n_2}=a_1+a_2+2(b_1+b_2)n_2-\lambda=0$$

$$\frac{\partial L}{\partial \lambda}=N-n_1-n_2=0$$

容易解出：

$$n_1=n_2=\frac{N}{2}$$

因此可以看到，此时每辆车的运行时间为：$t=(a_1+a_2)+(b_1+b_2)\frac{N}{2}$。

注意到上述参数中 a 实际上代表的是道路的长度，道路越长，a 值越大，b 代表的是道路的宽度，b 值越大，道路越宽，路况越好。根据《北京》的数据，我们可以初略地取 N＝3600。另外，再取 $a_1=500$, $b_1=0.03$, $a_2=1200$, $b_2=0.02$, ，代入上式，可得：$n_1=n_2=1790$，即在每条线路上平分车辆才可使得运输效率达到最高，每辆汽车的运行时间约为 1790 个单位。

1.2 Braess 悖论的分析

第二个问题，情况就相对复杂了。为了缓和交通压力，从建国桥到复兴门的匝道开放。因此此时司机有了第三种新的选择，即先选择从文慧门开往复兴门，再由复兴门下匝道，沿前门大街开往建国桥，最后从建国桥上匝道沿二环开往左安门。并且假前门大街由于种种原因仅为单向的开放，并且其路况好于二环，即可以取 $a_1=700$, $b_1=0.015$，现在我们讨论此种情况下的最优车流。

假设最优时车流的选择线路 1：文慧门->复兴门->左安门的司机有 x 人，选择线路 2：文慧门->建国桥->左安门的司机有 z 人，选择新线路，线路 3：文慧门->复兴门->建国桥->左安门的司机有 y 人。那么有：

线路 1 的司机所花的时间为 $T_{WFZ}=a_1+b_1(x+y)+a_2+b_2x$;

线路 2 的司机所花的时间为 $T_{WJZ}=a_2+b_2z+a_1+b_1(y+z)$;

线路 3 的司机所花的时间为 $T_{WFJZ}=a_1+b_1(x+y)+a_3+b_3y+a_1+b_1(y+z)$

那么所有司机会的总时间，即新情况下的优化目标函数为：

Min $T=x*T_{WFZ}+y*T_{WFJZ}+z*T_{WJZ}$

约束条件为：

s.t $x+y+z=N$

构造 Lagrange 函数：

$$L(x,y,z,\lambda)=x*(a_1+b_1(x+y)+a_2+b_2x)+y*(a_2+b_2z+a_1+b_1(y+z))$$

$$+z*(a_2+b_2z+a_1+b_1(y+z))+\lambda(N-x-y-z)$$

求偏导得到：

$$\frac{\partial L(x,y,z,\lambda)}{\partial x}=a_1+2b_1x+b_1y+a_2+2b_2x+b_1-\lambda=0$$

$$\frac{\partial L(x,y,z,\lambda)}{\partial y}=b_1x+a_1+b_1(x+2y)+a_4+2b_4y+a_1+b_1(2y+z)+b_1z-\lambda=0$$

$$\frac{\partial L(x,y,z,\lambda)}{\partial x}=a_2+2b_2z+a_1+b_1(y+2z)+b_1y-\lambda=0$$

求解，可以得到：

$$x=z=\frac{2(b_1+b_4)N+a_4+(a_2-a_1)}{2b_1+4b_4+2b_2}$$

$$y=\frac{(2b_4-(b_1+b_2))N-(a_4+(a_1-a_2))}{b_1+2b_4+b_2}$$

可以看到，在新的情况中，第一种选择和第二种选择的司机人数仍然相等。我们将上面的数据重新代入，可以得到：x=z=2250, y=-900。该解看似非常奇怪，但是却是符合逻辑的。结论是：1350 辆车走 WFZ 线，2250 辆车沿 WJZ 线走到建国门(J)之后分道扬镳，1350 辆继续前行到左安门，另外 900 辆逆行到达目的地。这样，实际上有 900 辆车走了文慧门—>建国桥->复兴门->左安门(WJFZ)的逆行线路。实际此总路线运行的平均时间为 2533 个单位。这说明，在现在的路网上，由于有前门大街的出现，使得整个交通的运行效率降低，造成了高峰时段二环路的极度拥堵。

2 基于北京市二环路的 Braess 网络的动态分析

以上，考虑的是北京市二环路的静态特征，下面从动力学角度分析其动态优化特性。

考虑一个长期的动态均衡过程，即不妨设车辆可以通过自组织达到最优化分配与动态协调的目的。如上节显示，两条不同线路的综合路况相同，那么假设司机选路的原则为前一天选线路 1 的司机有 p 的可能性在第二天选择线路 2，并记选择第一条线路的司机人数为 x，第二条线路的司机人数为 z，由此可以得到相应的偏微分方程：

$$\begin{cases}\frac{dx}{dt}=-px+pz=p(z-x)\\ \frac{dz}{dt}=-pz+px=p(x-z)\end{cases}$$

因此从微分方程可以明显看出，当且仅当 x=z 时，有 $\frac{dx}{dt}=\frac{dz}{dt}=0$，从而两种路线的选择达到动态均衡，因此与之前的静态解析解得出结论相同。但是考虑开放匝道后的情况，设选择新线路的司机个数为 y，得到以下方程：

$$\begin{cases}\frac{dx}{dt}=ax+byz+cz\\ \frac{dy}{dt}=cy-byz-exy\\ \frac{dz}{dt}=cx+exy+fz\end{cases}$$

其中 a,b,c,d,e,f 均为参数，即在原方程中加入了交叉项的影响。可以原点

(0,0,0)为该方程的平衡点，因此没有车流的静止状态。现考虑一个对平衡点的小扰动(Δx,Δy,Δz)，将方程线性化后可以得到关于（Δx, Δy,Δz）的方程：

$$\frac{d}{dt}\begin{bmatrix}\Delta x\\ \Delta y\\ \Delta z\end{bmatrix}=\begin{bmatrix}a & 0 & c\\ 0 & d & 0\\ c & 0 & f\end{bmatrix}$$

由此可解得其特征根可以表示为：$\lambda_{1,2}=\frac{a+f\pm\sqrt{(a+f)^2-4(af-c^2)}}{2}$，$\lambda_3$=d 所有特征根均大于零，即系统时时失去平衡是不可避免的，路途塞车必然发生。

四 Braess 模型的模拟仿真

1 仿真环境

考虑到灵活性、快捷性以及个人的优势，我们使用了 VC++/MFC 作为编程工具。

2 机制设计

(1)本程度以细胞自动机机制为设计基础。细胞自动机是一种离散模型，在可算性理论、数学及理论生物学都有相关研究。它是由无限个有规律、坚硬的方格组成，每格均处于一种有限种态。整个格网可以是任何有限维的。时也是离散的。每格于 t 时的态由 t-1 时的一集有限格（这集叫那格的邻域）的态决定。 每一格的“邻居”都是已被固定的。（一格可以是自己的邻居。）每次演进时，每格均遵从同一规矩一齐演进。图 4 为其仿真示意图:

(2)公路示意图如下所示，起点为左下角文慧门，终点为右上角左安门。一共有五条公路。从文慧门到左安门一共有三条路线。为方便起见，我们对各段道路进行编号。文慧门至建国桥记为 1，文慧门至复兴门记为 2，复兴门至左安门记为 3，建国桥至左安门记为 4，复兴门至建国门(前门大街)记为 5。线路 1（2-3）和线路 2（1-4）为初始的两条可供选择的线路，线路 3（2-5-4）为由于新开匝道后新加前门大街的一条路线。根据右边“选择路线”中的设定，可以设定走 3 条线路的车辆的百分比。其中，“自适应”选项是指驾驶员可以根据当时交通的状况，自适应选择路线。

(3)车速有两种设计方法：1.每辆车在初始时被随机赋予一个速度，一共分为 5 档；2.每辆车的速度是由当时的路况决定的，汽车经过一条路的时间函数是 T=ai+bif，其中 i=1,2,3,4,5 是路的编号。为简便起见，我们假设公路 1 和 3 的路况函数是一致的，公路 2 和 4 的路况函数是一致的，公路 5 有自己独立的路况函数。

(4)每条公路并行同时只能开一辆汽车，为模型简化，假设每辆汽车的大小相同。在行驶过程中，除十字路口外，相邻辆车之间间距不得小于车长的 3 倍，如果小于的话则停下等待（不能超速）。车辆从出发点出发的起始时间为[0,5]均匀分布的随机变量，这即能保证汽车的出发不是同时的，具有随机性，又能保证对最后结果的影响很小。

(5)模型可以动态设定经过的车辆总数、运行次数。在运行过程中，程序

会自动记录每条公路上当前的车辆数、汽车经过每条公路平均的占用时间以及它的方差、从起点到终点经过的最长时间、所有汽车从起点到终点的加总时间等统计量，以便讨论。

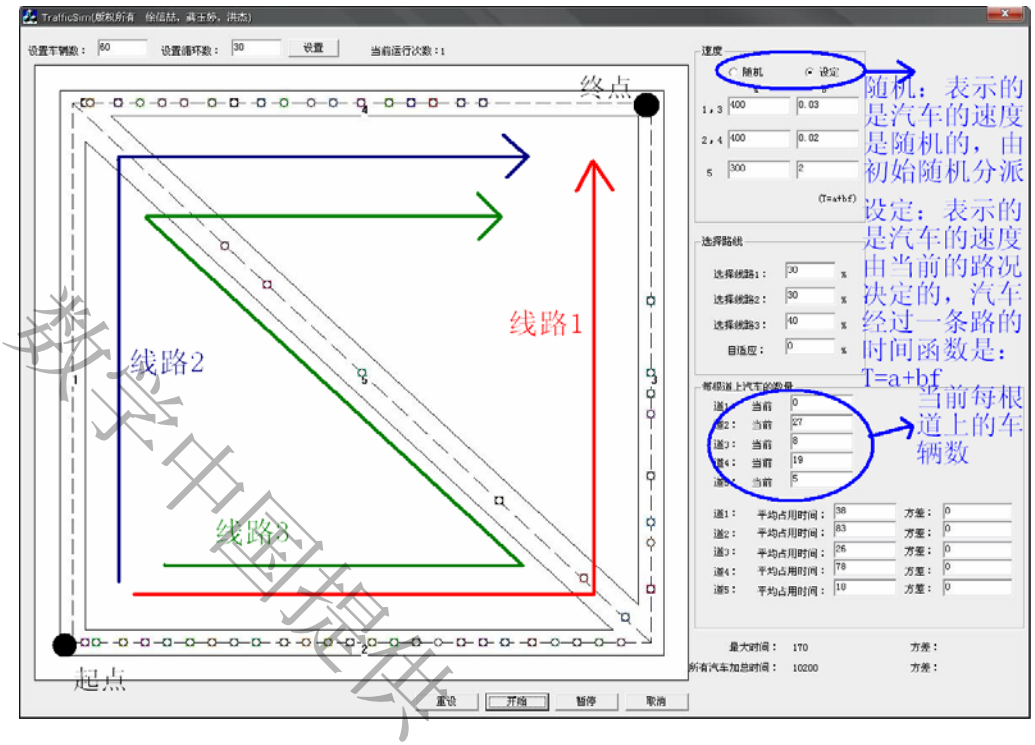

图 4 仿真程序示意图

3 仿真结果及分析

我们共分三种情况进行计算机模拟仿真。第一种是在无 GPS 导航，车辆在不同分流比例下的行驶情况。第二种是在 GPS 导航模式一，即认为个人最优，即选择线路 3 为指导路线时的行驶状况。并且我们用自适应比表示未安装 GPS 的司机的比例。第三种是在 GPS 导航模式二，即从全局最优，认为不选择线路 3 而平均地选择线路 1 和 2 为指导路线时的行驶状况。

车辆分流比例			道路平均停留时间					耗时	
路线 1	路线 2	路线 3	道路 WJ	道路 WF	道路 FZ	道路 JZ	道路 FJ	最大耗时	总耗时
50	50	0	68	66	49	60	0	336.9	20214
33	33	33	40	99	31	71	16	407.9	24474
30	30	40	36	105	29	74	18	413.6	24816
0	0	100	0	192	0	99	50	492.7	29562
100	0	0	0	192	100	0	0	442.4	26544
0	100	0	35	108	47	53	10	407.6	24456
30	50	20	65	67	31	74	9	377	22620
20	30	50	23	63	8	61	31	417.7	25122
50	20	30	35	106	47	52	9	405.7	24342

表 1 情况一的平均行驶时间

车辆分流比例			道路停留时间方差					耗时	
路线 1	路线 2	路线 3	道路 WJ	道路 WF	道路 FZ	道路 JZ	道路 FJ	最大耗时	总耗时
50	50	0	168.66	202.49	43.73	48.84	0	336.9	20214
33	33	33	46.22	232.01	45.17	101.96	4.4	407.9	24474
30	30	40	56.93	239.15	26.32	60.68	5.57	413.6	24816
0	0	100	0	45.82	0	0	0	492.7	29562
100	0	0	0	36.28	0	0	0	442.4	26544
0	100	0	88.06	171.16	77.66	124.46	6.49	407.6	24456
30	50	20	48.84	69.34	31.61	91.07	9.33	377	22620
20	30	50	19.34	185.38	8.04	144.28	70.99	417.7	25122
50	20	30	38.77	106.06	11.82	19.21	6.32	405.7	24342

表 2 情况一的道路停留时间方差

车辆分流比例				道路平均停留时间					耗时	
路线 1	路线 2	路线 3	自适应	道路 WJ	道路 WF	道路 FZ	道路 JZ	道路 FJ	最大耗时	总耗时
45	45	0	10	59	75	44	57	4	372.05	22323
40	40	0	20	50	87	40	61	9	391.45	23487
30	30	0	40	33	112	28	73	20	418	25119
20	20	0	60	21	134	20	80	28	441.35	26481
10	10	0	80	15	149	9	91	36	454.6	27276
5	5	0	90	14	152	4	104	40	465.8	27948
0	0	0	100	15	155	0	121	42	485.7	29142

表 3 情况二的平均行驶时间

车辆分流比例				道路停留时间方差					耗时	
路线 1	路线 2	路线 3	自适应	道路 WJ	道路 WF	道路 FZ	道路 JZ	道路 FJ	最大耗时	总耗时
45	45	0	10	101.14	124.66	36.36	58.96	3	372.05	22323
40	40	0	20	127.33	239.88	60.09	103.59	5.52	391.45	23487
30	30	0	40	106.89	367.25	34.45	58.68	13.22	418	25119
20	20	0	60	49.4	256.88	23.31	36.47	11.21	441.35	26481
10	10	0	80	7.63	41.01	13.73	3.4	6.3	454.6	27276
5	5	0	90	7.31	53.29	6.66	54.48	3.4	465.8	27948
0	0	0	100	10.77	95.14	0	24.73	2.15	485.7	29142

表 4 情况二的道路停留时间方差

车辆分流比例				道路平均停留时间					耗时	
路线 1	路线 2	路线 3	自适应	道路 WJ	道路 WF	道路 FZ	道路 JZ	道路 FJ	最大耗时	总耗时
0	0	0	100	15	155	0	121	42	487.7	29242
0	0	10	90	15	154	0	121	42	484.45	29067
0	0	25	75	11	162	0	116	43	484.6	29076
0	0	35	65	10	165	0	114	44	485.3	29118
0	0	50	50	7	171	0	109	45	485.85	29151
0	0	65	35	5	177	0	106	46	487	29220
0	0	75	25	2	184	0	101	47	487.45	29247
0	0	90	10	1	189	0	100	49	490.6	29436
0	0	100	0	0	192	0	99	50	492.7	29562

表 5 情况三的平均行驶时间

车辆分流比例				道路停留时间方差					耗时	
路线 1	路线 2	路线 3	自适应	道路 WJ	道路 WF	道路 FZ	道路 JZ	道路 FJ	最大耗时	总耗时
0	0	0	100	10.77	95.14	0	24.73	2.15	487.7	29242
0	0	10	90	3.57	27.4	0	7.63	0.89	484.45	29067
0	0	25	75	8.87	60.13	0	24.79	1.57	484.6	29076
0	0	35	65	11.78	101.29	0	40.73	2.13	485.3	29118
0	0	50	50	8.73	102.89	0	30.93	1.52	485.85	29151
0	0	65	35	3.71	63.5	0	17.99	1.4	487	29220
0	0	75	25	3.84	44.87	0	7.52	1.57	487.45	29247
0	0	90	10	2.47	69.25	0	7.21	1.17	490.6	29436
0	0	100	0	0	45.82	0	0	0	492.7	29562

表 6 情况三的道路停留时间方差

将各表的平均时间绘制成图如下：

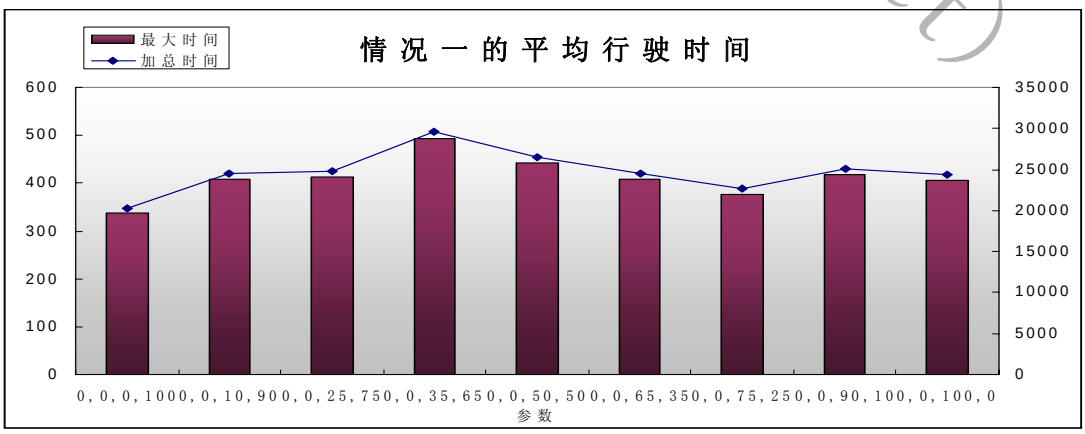

图 5 情况一的平均行驶时间图

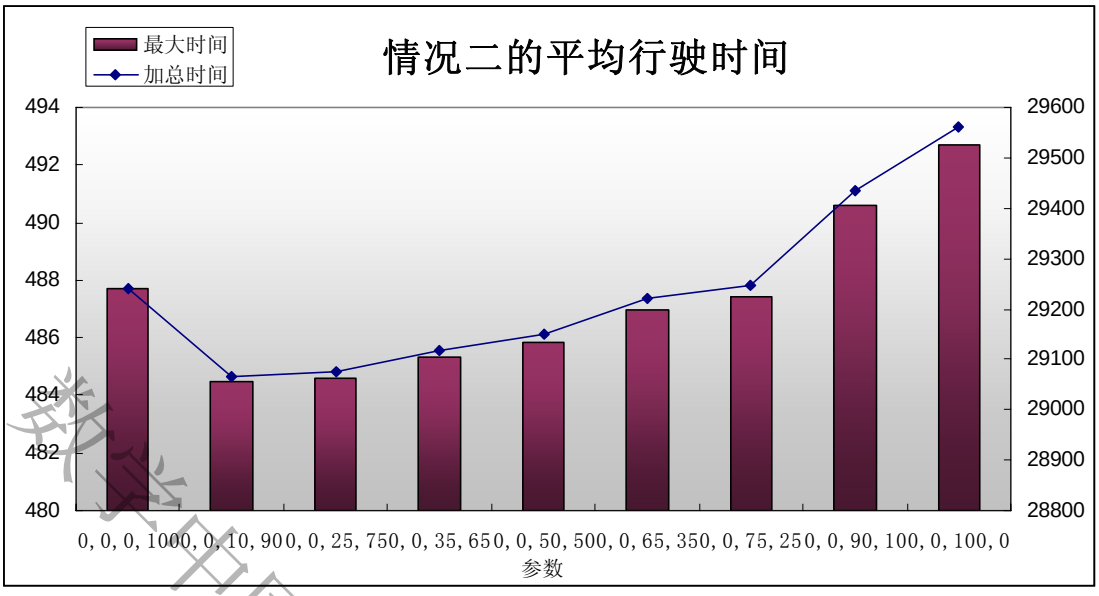

图 6 情况二的平均行驶时间图

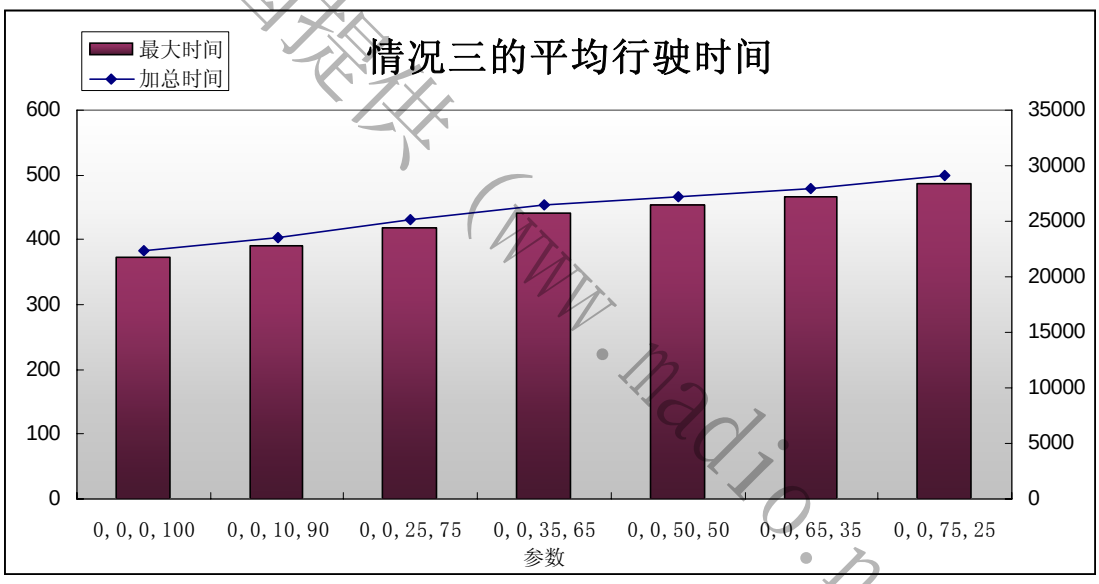

图 7 情况三的平均行驶时间图

根据表中分析的结果，可得到关于交通网络自适应系统的“Braess 悖论”的主要结论有三点：

（1）当所有车辆都不使用 GPS 导航系统时，在交通拥堵的高峰时段，尽管许多东西走向的道路提供了更多车辆的路径，但各车辆在选择自身最优路径的同时加剧了其他道路的拥堵，开辟一条新道路反而降低了交通系统的使用效率，存在“Braess 悖论”中描述的问题。

这主要是因为各车辆在各个交通交叉路口都按照自身可观察到的道路状况选择当前最畅通的道路通行，造成了交通网络中的外部性，各车辆通行时不用对交通堵塞和其他车辆的延误付出成本，各车辆最优化自身利益会造成负的外部性，从而无法达到全社会的 Pareto 最优，

如表 1 所示，当没有 GPS 导航系统时，表中给出的几种车流分布形式下，

所有车流以相同的比例通过路径 1 和路径 2，而不使用路径 3 时，全社会的效应达到最优， 即所有车辆通过整条道路的时间最短（20217s）。而所有车流采用等概率通过路径 1、路径 2 和路径 3 是次优解。这种结果所花费的时间约为（22474s）,虽没有最优情况下的线路效率高，但比其他车流分布形式通过道路所耗时间要短些。从表中还可看出，当各道路车流分布较为均匀时，全体车辆通过的耗时较短，如车流以 30%、20%、50%的比例通过路径 1、2、3 时所用时间为 22620s。相反，如果车流分布趋于极端，即所有的车辆集中于某一条道路上而其他道路处于闲置，则大量车辆会在主干道上拥堵而极大延长通行时间。

（2）当所有车辆广泛使用反映路面状况的 GPS 导航系统时，GPS 系统为司机提供最畅通的路径作为司机参考 GPS 导航提供的行驶建议，而通过 GPS 导航系统只是使各车辆掌握整体路况，没有使全体车辆按照社会最优的路线行驶，即以 50%、50%的比例通过路径 1、路径 2，而放弃新开通的路径 3。

通常情况下，司机如果使用 GPS 导航系统，在决定行驶路径时会参考 GPS 系统提供的实时路况，但一般也不会完全遵循 GPS 指定的路径驾驶，而是根据自身驾驶经验及对周围道路现状的观察，将 GPS 的电子数据和亲身经历结合起来，指定最优最省时的行车路线。基于这样的思路，我们进行的模拟仿真赋予各车辆按照 GPS 导航行驶的比例，即司机虽然了解了 GPS 导航的行车建议，但因自身经验、现时观察等多种原因对这一建议并不采纳，而随机选择其他路径。假设这一比例从 0%、25%、50%、75%、100%逐步增加进行模拟仿真，并将各路段所耗平均时间和所有车辆总耗时列于表 3 中。

根据数据结果，我们发现随着司机对 GPS 导航系统的接受程度不断增加，由于 GPS 导航系统也只是替司机提供最优的行车路径，和完全依靠司机自身做决定的情况相比，并没有从实质上改变作最优决策的主体，因此也无法避免“Braess”悖论谈及的效率损失。

（3） 当所有车辆广泛使用反映路面状况的 GPS 导航系统时，GPS 系统为司机提供最畅通的路径作为司机参考 GPS 导航提供的行驶建议并结合自身经验和观察而选择当时条件下的最优路径，两者相互作用的结果会使全体车辆通过道路的时间经过自适应调节，当只有部分车辆听从 GPS 的建议（表 2 中为 10%）时，全体车辆的行驶时间达到最短，此时全部车辆以 40%、45%、10%的比例通过路径 1、2、3。随后，如果有多余 10%的司机完全依据 GPS 导航提出的路线行驶，则会引导越来越多的车辆向路径 3 行驶，而在路径 1 和路径 2 的主干道上发生越来越严重的拥堵，从而导致全社会效率的降低，车辆通过道路到达目的地的总时间越来越长。这与北京二环在高峰时段发生拥堵的另一个可能的重要原因，即使存在 GPS 导航也无法使全体车辆的运行达到社会最优。

五 结论

本文用随机线性及非线性动态规划方法研究北京二环高峰时段交通堵塞的原因，并用自适应系统的模拟仿真实验验证理论模型。主要结果表明： 二环路的道路拥堵符合“Braess 悖论”描述的情况，交通堵塞问题并不会因司机广泛使用 GPS 导航系统而得到缓解，这一结论在细胞自适应系统的仿真模拟实验中得到验证。所以，“Braess”悖论是否发生不仅和个体是否了解全局状况有关，而且还与司机是否遵循 GPS 建议的最优路线行驶密切相关。

参考文献

【1】 Xin Huang*, Asuman Ozdaglr, and Daron Acemoglu. Efficiency and Braess’ Paradox under Pricing in General Networks. Massachusetts Institute of Technology, May 18, 2005.

【2】 Ana L. C. Bazzan*, Franziskal. Case Studies on the Braess Paradox: Simulating route recommendation and learning in abstract and microscopic models. Transportation Research Part C 13 (2005) 299-319.

【3】 Hagstrom J. Braess’ s Paradox [R] . 2004. Available at: http://tigger.uic.edu/

【4】 Braess D. A paradox on the traffic networks [R] . 2004. Available at: http://homepage.ruhr-uni-bochum.de/

【5】 陈彦光，刘继生. Braess 模型与城市网络的空间复杂化探讨. 地理科学，26 (2006) :658-663.

附录 1：程序部分重要源代码

```
//开始按钮相应函数
void CTrafficSimDlg::OnOK ( )
{
	CStatic* pic =  ( CStatic* ) GetDlgItem ( IDC_PICTURE ) ;
	CDC* dc = pic->GetWindowDC ( ) ;
	CRect rect;
	pic->GetClientRect ( &rect ) ;
	DrawBgd ( dc,&rect ) ;
	UpdateData ( true ) ;
	//设置 Car 的初始参数
	srand ( unsigned ( time ( NULL ) ) ) ;
	for ( int i=0;i<m_carNum;i++ )
	 {
		int rv= rand ( ) %100; //0-99
		if ( rv>=0 && rv<m_P1 )  m_car [ i ] .m_line = 1; //选择 1 线路
		else if ( rv>=m_P1 && rv<m_P1+m_P2 )  m_car [ i ] .m_line = 2; //选择 2
线路
		else if ( rv>=m_P1+m_P2 && rv<m_P1+m_P2+m_P3 )  m_car [ i ] .m_line
= 3; //选择 3 线路
		else m_car [ i ] .m_line = 4; //选择自适应
	 }
	dc->DeleteDC ( ) ;
	SetTimer ( 1,5,NULL ) ;
}

//公路背景函数
void CTrafficSimDlg::DrawBgd ( CDC* dc,CRect* rect )
{
	int h,w,offset;
	h = rect->bottom;
	w = rect->right;
	offset = 30;

	dc->Rectangle ( 0,0,w,h ) ;
	dc->MoveTo ( offset,offset ) ;
	dc->LineTo ( offset,h-offset ) ;
	dc->LineTo ( w-offset,h-offset ) ;
	dc->LineTo ( w-offset,offset ) ;
	dc->LineTo ( offset,offset ) ;

	dc->MoveTo ( 2*offset,3*offset ) ;
	dc->LineTo ( 2*offset,h-2*offset ) ;
```

```
	dc->LineTo ( w-3*offset,h-2*offset ) ;
	dc->LineTo ( 2*offset,3*offset ) ;

	dc->MoveTo ( 3*offset,2*offset ) ;
	dc->LineTo ( w-2*offset,2*offset ) ;
	dc->LineTo ( w-2*offset,h-3*offset ) ;
	dc->LineTo ( 3*offset,2*offset ) ;

	CPen pen;
	CPen* oldpen;
	pen.CreatePen ( PS_DASH,1,RGB ( 0,0,0 ) ) ;
	oldpen = dc->SelectObject ( &pen ) ;

	dc->MoveTo ( offset*3/2,h-offset*3/2 ) ;
	dc->LineTo ( w-offset*3/2,h-offset*3/2 ) ;
	dc->LineTo ( w-offset*3/2,offset*3/2 ) ;
	dc->LineTo ( offset*3/2,offset*3/2 ) ;
	dc->LineTo ( offset*3/2,h-offset*3/2 ) ;
	dc->MoveTo ( w-offset*3/2,h-offset*3/2 ) ;
	dc->LineTo ( offset*3/2,offset*3/2 ) ;

	dc->SelectObject ( oldpen ) ;
	pen.DeleteObject ( ) ;


	dc->TextOut ( offset*3/2,h/2,"1" ) ;
	dc->TextOut ( w/2,h-offset*3/2,"2" ) ;
	dc->TextOut ( w-offset*3/2,h/2,"3" ) ;
	dc->TextOut ( w/2,offset*3/2,"4" ) ;
	dc->TextOut ( w/2,h/2,"5" ) ;
}

//绘制小汽车函数
void CTrafficSimDlg::DrawCar ( CDC* dc, CRect* rect, const Car& car )
{
	int cx,cy;
	int w,h,offset,tick;
	w = rect->right;
	h = rect->bottom;
	offset = 30;
	tick = 100;
	switch ( car.m_position.m_lane )
	 {
	case 10:
```

```
        cx = offset*3/2;
    cy = h-offset*3/2;
        break;
case 20:
        cx = w-offset*3/2;
    cy = h-offset*3/2;
        break;
case 30:
        cx = w-offset*3/2;
    cy = offset*3/2;
        break;
case 40:
        cx = offset*3/2;
    cy = offset*3/2;
        break;
case 1:
        cx = offset*3/2;
        cy = h - 2*offset - ( h-4*offset ) *car.m_position.m_pos/tick;
        break;
case 2:
        cx = 2*offset + ( w-4*offset ) *car.m_position.m_pos/tick;
        cy = h-offset*3/2;
        break;
case 3:
        cx = w-offset*3/2;
        cy = h - 2*offset - ( h-4*offset ) *car.m_position.m_pos/tick;
        break;
case 4:
        cx = 2*offset + ( w-4*offset ) *car.m_position.m_pos/tick;
        cy = offset*3/2;
        break;
case 5:
        {
            int ox,oy,dx,dy;
            ox = w-offset*5/2;
            oy = h-offset*5/2;
            dx = offset*5/2;
            dy = offset*5/2;
            cx = ox - ( ox-dx ) *car.m_position.m_pos/tick;
            cy = oy - ( oy-dy ) *car.m_position.m_pos/tick;
            break;
        }
default:
        break;
```

```
    }

    dc->FillSolidRect ( cx-CAR_WIDTH/2,cy-CAR_WIDTH/2,CAR_WIDTH,CAR
_WIDTH, car.m_color ) ;
    dc->Ellipse ( cx-CAR_WIDTH/2,cy-CAR_WIDTH/2,cx+CAR_WIDTH/2,cy+C
AR_WIDTH/2 ) ;

}

//时间响应函数
void CTrafficSimDlg::OnTimer ( UINT nIDEvent )
{
    if ( nIDEvent == 1 )
    {
        m_cnt++;
        CStatic* pic = ( CStatic* ) GetDlgItem ( IDC_PICTURE ) ;
        CDC* dc = pic->GetWindowDC ( ) ;
        CRect rect;
        pic->GetClientRect ( &rect ) ;
        CDC memDC;
        memDC.CreateCompatibleDC ( dc ) ;
        CBitmap bitmap;
        bitmap. CreateCompatibleBitmap ( dc, rect. right-rect. left,
rect.bottom-rect.top ) ;
        memDC.SelectObject ( &bitmap ) ;
        DrawBgd ( &memDC,&rect ) ;
        int w,h,l;
        int offset = 30;
        w = rect.right - 4*offset;
        h = rect.bottom - 4*offset;
        l =  ( int ) sqrt ( ( rect.right-5*offset ) * ( rect.right-5*offset ) + ( rect.bottom -
5*offset ) * ( rect.bottom - 5*offset ) ) ;
        for ( int i=0;i<m_carNum;i++ )
        {
            if ( m_car [ i ] . m_start <= m_cnt &&
GetForwardDist ( m_car [ i ] .m_position, m_car [ i ] .m_line ) >= CAR_WIDTH*3 )
            {
                //如果速度为设定的，否则取初始随机的值
                if ( m_vtype == 1 )
                {
                    if ( m_car [ i ] .m_position.m_lane == 1 )
                        m_car [ i ] . m_v = h / ( m_a1 + m_b1 *
( double ) m_L1Cnt ) ;
                    else if ( m_car [ i ] .m_position.m_lane == 3 )
```

```
                    m_car[ i ] . m_v = h / ( m_a1 + m_b1 *
( double ) m_L3Cnt ) ;
                else if ( m_car [ i ] .m_position.m_lane == 2 )
                    m_car[ i ] . m_v = w / ( m_a2 + m_b2 *
( double ) m_L2Cnt ) ;
                else if ( m_car [ i ] .m_position.m_lane == 4 )
                    m_car[ i ] . m_v = w / ( m_a2 + m_b2 *
( double ) m_L4Cnt ) ;
                else if ( m_car [ i ] .m_position.m_lane == 5 )
                    m_car [ i ] .m_v = l / ( m_a5 + m_b5 * m_L5Cnt ) ;
            }
            for ( int j=0;j<m_car [ i ] .m_v;j++ )
                NextStep ( m_car [ i ] .m_position, m_car [ i ] .m_line ) ;
        }
        if ( m_car [ i ] .m_position.m_lane != 30 )
        {
            DrawCar ( &memDC,&rect,m_car [ i ] ) ;
        }
        m_car [ i ] .m_Time++;
        if ( m_car [ i ] .m_position.m_lane == 1 ) m_car [ i ] .m_Time_L1++;
        if ( m_car [ i ] .m_position.m_lane == 2 ) m_car [ i ] .m_Time_L2++;
        if ( m_car [ i ] .m_position.m_lane == 3 ) m_car [ i ] .m_Time_L3++;
        if ( m_car [ i ] .m_position.m_lane == 4 ) m_car [ i ] .m_Time_L4++;
        if ( m_car [ i ] .m_position.m_lane == 5 ) m_car [ i ] .m_Time_L5++;
    }

    dc->BitBlt ( 0,0,rect.right,rect.bottom,&memDC,0,0,SRCCOPY ) ;
    bitmap.DeleteObject ( ) ;
    memDC.DeleteDC ( ) ;

    //计算每根道上的汽车数量
    m_L1Cnt=m_L2Cnt=m_L3Cnt=m_L4Cnt=m_L5Cnt=0;
    for ( i=0;i<m_carNum;i++ )
    {
        if ( m_car [ i ] .m_position.m_lane == 1 )
            m_L1Cnt ++;
        else if ( m_car [ i ] .m_position.m_lane == 2 )
            m_L2Cnt ++;
        else if ( m_car [ i ] .m_position.m_lane == 3 )
            m_L3Cnt ++;
        else if ( m_car [ i ] .m_position.m_lane == 4 )
            m_L4Cnt ++;
        else if ( m_car [ i ] .m_position.m_lane == 5 )
            m_L5Cnt ++;
```

```
        }
        m_L1 = m_L1Cnt;
        m_L2 = m_L2Cnt;
        m_L3 = m_L3Cnt;
        m_L4 = m_L4Cnt;
        m_L5 = m_L5Cnt;
        dc->DeleteDC ( ) ;
    }

    m_maxTime.Format ( "%d",m_cnt ) ;
    int ttime=0;
    for ( int i=0;i<m_carNum;i++ )
        ttime+=m_car [ i ] .m_Time;
    m_totalTime.Format ( "%d",ttime ) ;
    UpdateData ( false ) ;
    int target = 0;
    for ( i=0;i<m_carNum;i++ )
    {
        if ( m_car [ i ] .m_position.m_lane != 30 )
            break;
        target ++;
    }
    if ( target >= m_carNum )
    {
        m_avgT1 = m_avgT2 = m_avgT3 = m_avgT4 = m_avgT5 = 0;
        for ( i=0;i<m_carNum;i++ )
        {
            m_avgT1 += m_car [ i ] .m_Time_L1;
            m_avgT2 += m_car [ i ] .m_Time_L2;
            m_avgT3 += m_car [ i ] .m_Time_L3;
            m_avgT4 += m_car [ i ] .m_Time_L4;
            m_avgT5 += m_car [ i ] .m_Time_L5;
        }
        m_avgT1 = m_avgT1/m_carNum;
        m_avgT2 = m_avgT2/m_carNum;
        m_avgT3 = m_avgT3/m_carNum;
        m_avgT4 = m_avgT4/m_carNum;
        m_avgT5 = m_avgT5/m_carNum;
        UpdateData ( false ) ;
        m_curRumtime++;
        m_curRT.Format ( "%d", m_curRumtime ) ;
        UpdateData ( false ) ;
        //判断 curRuntime 是否等于 Runtime，如果是，则结束，否则重新
跑;
```

```
if ( m_curRumtime>=m_runtime )
{
    KillTimer ( 1 ) ;
    //计算统计量
    stat [ 0 ] .value [ m_curRumtime-1 ] = ( double ) m_cnt;
    stat [ 0 ] .num = m_curRumtime;
    stat [ 1 ] .value [ m_curRumtime-1 ] = ( double ) ttime;
    stat [ 1 ] .num = m_curRumtime;
    stat [ 2 ] .value [ m_curRumtime-1 ] = ( double ) m_avgT1;
    stat [ 2 ] .num = m_curRumtime;
    stat [ 3 ] .value [ m_curRumtime-1 ] = ( double ) m_avgT2;
    stat [ 3 ] .num = m_curRumtime;
    stat [ 4 ] .value [ m_curRumtime-1 ] = ( double ) m_avgT3;
    stat [ 4 ] .num = m_curRumtime;
    stat [ 5 ] .value [ m_curRumtime-1 ] = ( double ) m_avgT4;
    stat [ 5 ] .num = m_curRumtime;
    stat [ 6 ] .value [ m_curRumtime-1 ] = ( double ) m_avgT5;
    stat [ 6 ] .num = m_curRumtime;
    stat [ 0 ] .Calculate ( ) ;
    stat [ 1 ] .Calculate ( ) ;
    stat [ 2 ] .Calculate ( ) ;
    stat [ 3 ] .Calculate ( ) ;
    stat [ 4 ] .Calculate ( ) ;
    stat [ 5 ] .Calculate ( ) ;
    stat [ 6 ] .Calculate ( ) ;

    //显示
    m_maxTime.Format ( "%.2f", stat [ 0 ] .avg ) ;
    m_maxtimeVar.Format ( "%.2f", stat [ 0 ] .var ) ;
    m_totalTime.Format ( "%.2f", stat [ 1 ] .avg ) ;
    m_totaltimeVar.Format ( "%.2f", stat [ 1 ] .var ) ;
    m_avgT1 = stat [ 2 ] .avg;
    m_VarL1 = stat [ 2 ] .var;
    m_avgT2 = stat [ 3 ] .avg;
    m_VarL2 = stat [ 3 ] .var;
    m_avgT3 = stat [ 4 ] .avg;
    m_VarL3 = stat [ 4 ] .var;
    m_avgT4 = stat [ 5 ] .avg;
    m_VarL4 = stat [ 5 ] .var;
    m_avgT5 = stat [ 6 ] .avg;
    m_VarL5 = stat [ 6 ] .var;
    UpdateData ( false ) ;
}
else
```

```
            {
                KillTimer ( 1 ) ;
                //记录统计量
                stat [ 0 ] .value [ m_curRumtime-1 ] = ( double ) m_cnt;
                stat [ 1 ] .value [ m_curRumtime-1 ] = ( double ) ttime;
                stat [ 2 ] .value [ m_curRumtime-1 ] = ( double ) m_avgT1;
                stat [ 3 ] .value [ m_curRumtime-1 ] = ( double ) m_avgT2;
                stat [ 4 ] .value [ m_curRumtime-1 ] = ( double ) m_avgT3;
                stat [ 5 ] .value [ m_curRumtime-1 ] = ( double ) m_avgT4;
                stat [ 6 ] .value [ m_curRumtime-1 ] = ( double ) m_avgT5;
                OnButtonReset ( ) ;
                OnOK ( ) ;
            }
        }
        CDialog::OnTimer ( nIDEvent ) ;
}

//小汽车下一步动作函数
bool CTrafficSimDlg::NextStep ( CarPosition& pos, int Line )
{
    if ( Line == 1 )
    {
        switch ( pos.m_lane )
        {
        case 10:
            pos.m_lane = 2;
            pos.m_pos = 0;
            break;
        case 20:
            pos.m_lane = 3;
            pos.m_pos = 0;
            break;
        case 30:
            return false;
        case 2:
            if ( pos.m_pos<99 )  pos.m_pos++;
            else
            {
                pos.m_lane = 20;
                pos.m_pos = 0;
            }
            break;
        case 3:
            if ( pos.m_pos<99 )  pos.m_pos++;
```

```
            else
            {
                pos.m_lane = 30;
                pos.m_pos = 0;
            }
            break;
        default:
            return false;
        }
    }
    else if ( Line == 2 )
    {
        switch ( pos.m_lane )
        {
        case 10:
            pos.m_lane = 1;
            pos.m_pos = 0;
            break;
        case 30:
            return false;
        case 40:
            pos.m_lane = 4;
            pos.m_pos = 0;
            break;
        case 1:
            if ( pos.m_pos<99 )  pos.m_pos++;
            else
            {
                pos.m_lane = 40;
                pos.m_pos = 0;
            }
            break;
        case 4:
            if ( pos.m_pos<99 )  pos.m_pos++;
            else
            {
                pos.m_lane = 30;
                pos.m_pos = 0;
            }
            break;
        default:
            return false;
        }
    }
```

```
else if ( Line == 3 )
{
    switch ( pos.m_lane )
    {
    case 10:
        pos.m_lane = 2;
        pos.m_pos = 0;
        break;
    case 20:
        pos.m_lane = 5;
        pos.m_pos = 0;
        break;
    case 30:
        return false;
    case 40:
        pos.m_lane = 4;
        pos.m_pos = 0;
        break;
    case 1:
        if ( pos.m_pos<99 ) pos.m_pos++;
        else
        {
            pos.m_lane = 40;
            pos.m_pos = 0;
        }
        break;
    case 2:
        if ( pos.m_pos<99 ) pos.m_pos++;
        else
        {
            pos.m_lane = 20;
            pos.m_pos = 0;
        }
        break;
    case 3:
        if ( pos.m_pos<99 ) pos.m_pos++;
        else
        {
            pos.m_lane = 30;
            pos.m_pos = 0;
        }
        break;
    case 4:
        if ( pos.m_pos<99 ) pos.m_pos++;
```

```
                else
                {
                    pos.m_lane = 30;
                    pos.m_pos = 0;
                }
                break;
            case 5:
                if ( pos.m_pos<99 )  pos.m_pos++;
                else
                {
                    pos.m_lane = 40;
                    pos.m_pos = 0;
                }
                break;
            default:
                return false;
            }
        }
        else if ( Line = 4 )
        {
            return NextStepAdaptive ( pos ) ;
        }
        else return false;
        return true;
}

//自适应选择线路函数
bool CTrafficSimDlg::NextStepAdaptive ( CarPosition& pos )
{
        switch ( pos.m_lane )
        {
        case 10:
            //动态判断选择线路 1 和 2
            {
                int T1 = m_a1 + m_b1 *  ( double ) m_L1Cnt;
                int T2 = m_a2 + m_b2 *  ( double ) m_L2Cnt;
                if ( T1>=T2 )
                {
                    pos.m_lane = 2;
                    pos.m_pos = 0;
                }
                else if ( T1<T2 )
                {
                    pos.m_lane = 1;
```

```
                pos.m_pos = 0;
            }
        }
        break;
    case 20:
        //动态判断选择线路 3 和 5
        {
            int T3 = m_a1 + m_b1 * ( double ) m_L3Cnt;
            int T5 = m_a5 + m_b5 * ( double ) m_L5Cnt;
            if ( T3>=T5 )
            {
                pos.m_lane = 5;
                pos.m_pos = 0;
            }
            else if ( T3<T5 )
            {
                pos.m_lane = 3;
                pos.m_pos = 0;
            }
        }
        break;
    case 30:
        return false;
    case 40:
        pos.m_lane = 4;
        pos.m_pos = 0;
        break;
    case 1:
        if ( pos.m_pos<99 )  pos.m_pos++;
        else
        {
            pos.m_lane = 40;
            pos.m_pos = 0;
        }
        break;
    case 2:
        if ( pos.m_pos<99 )  pos.m_pos++;
        else
        {
            pos.m_lane = 20;
            pos.m_pos = 0;
        }
        break;
    case 3:
```

```
            if ( pos.m_pos<99 )  pos.m_pos++;
            else
            {
                pos.m_lane = 30;
                pos.m_pos = 0;
            }
            break;
        case 4:
            if ( pos.m_pos<99 )  pos.m_pos++;
            else
            {
                pos.m_lane = 30;
                pos.m_pos = 0;
            }
            break;
        case 5:
            if ( pos.m_pos<99 )  pos.m_pos++;
            else
            {
                pos.m_lane = 40;
                pos.m_pos = 0;
            }
            break;
        default:
            return false;
        }
        return true;
}

//判断小汽车的前面是否有车辆的函数
bool CTrafficSimDlg::IsEmptyForward ( CarPosition& pos, int Line )
{
        CarPosition tmp;
        tmp.m_lane = pos.m_lane;
        tmp.m_pos = pos.m_pos;

        NextStep ( tmp, Line ) ;

        CStatic* pic =  ( CStatic* ) GetDlgItem ( IDC_PICTURE ) ;
        CDC* dc = pic->GetWindowDC ( ) ;
        CRect rect;
        pic->GetClientRect ( &rect ) ;

        for ( int i=0;i<m_carNum;i++ )
```

```
    {
        int ccx,ccy;
        int tcx,tcy;
        GetCXCY ( dc,&rect,m_car [ i ] .m_position,m_car [ i ] .m_line,ccx,ccy ) ;
        GetCXCY ( dc,&rect,tmp,Line,tcx,tcy ) ;
        if ( m_car [ i ] . m_position. m_lane == tmp. m_lane &&
m_car [ i ] .m_position.m_lane != 5 )
            if ( ( ( ccx-tcx <= CAR_WIDTH*2 ) && ( ccx-tcx > 0 ) ) || ( ( tcy-ccy >
0 ) && ( tcy-ccy <= CAR_WIDTH*2 ) ) )
            {
                dc->DeleteDC ( ) ;
                return false;
            }
        if ( m_car [ i ] . m_position. m_lane == tmp. m_lane &&
m_car [ i ] .m_position.m_lane == 5 )
            if ( ( ( tcx-ccx <= CAR_WIDTH*2 ) && ( tcx-ccx > 0 ) ) || ( ( tcy-ccy >
0 ) && ( tcy-ccy <= CAR_WIDTH*2 ) ) )
            {
                dc->DeleteDC ( ) ;
                return false;
            }
    }

    dc->DeleteDC ( ) ;

    return true;
}

//判断小汽车前方车辆距离函数
int CTrafficSimDlg::GetForwardDist ( CarPosition& pos, int Line )
{
    CarPosition tmp;
    tmp.m_lane = pos.m_lane;
    tmp.m_pos = pos.m_pos;

    //NextStep ( tmp, Line ) ;

    CStatic* pic = ( CStatic* ) GetDlgItem ( IDC_PICTURE ) ;
    CDC* dc = pic->GetWindowDC ( ) ;
    CRect rect;
    pic->GetClientRect ( &rect ) ;
    int mindist=100000;

    for ( int i=0;i<m_carNum;i++ )
```

```
    {
        int ccx,ccy;
        int tcx,tcy;
        GetCXCY ( dc,&rect,m_car [ i ] .m_position,m_car [ i ] .m_line,ccx,ccy ) ;
        GetCXCY ( dc,&rect,tmp,Line,tcx,tcy ) ;
        if ( m_car [ i ] . m_position. m_lane == tmp. m_lane &&
m_car [ i ] .m_position.m_lane != 5 )
            if ( ( ccx>tcx ) || ( tcy>ccy ) )
            {
                if ( ccx>tcx && ( ccx-tcx ) < mindist )
                    mindist = ccx-tcx;
                if ( tcy>ccy && ( tcy-ccy ) < mindist )
                    mindist = tcy-ccy;
            }
        if ( m_car [ i ] . m_position. m_lane == tmp. m_lane &&
m_car [ i ] .m_position.m_lane == 5 )
            if ( ( tcx>ccx ) && ( tcy>ccy ) )
            {
                if ( ( tcx-ccx ) *( tcx-ccx ) + ( tcy-ccy ) *( tcy-ccy ) <
mindist*mindist )
                    mindist = ( int ) sqrt ( ( double ) ( ( tcx-ccx ) *( tcx-ccx ) +
( tcy-ccy ) *( tcy-ccy ) ) ) ;
            }
    }

    dc->DeleteDC ( ) ;

    return mindist;
}

bool CTrafficSimDlg: : GetCXCY ( CDC* dc, CRect* rect, CarPosition& pos, int
Line,int& cx,int& cy )
{
    int w,h,offset,tick;
    w = rect->right;
    h = rect->bottom;
    offset = 30;
    tick = 100;
    switch ( pos.m_lane )
    {
    case 10:
        cx = offset*3/2;
      cy = h-offset*3/2;
        break;
```

```
        case 20:
            cx = w-offset*3/2;
          cy = h-offset*3/2;
            break;
        case 30:
            cx = w-offset*3/2;
          cy = offset*3/2;
            break;
        case 40:
            cx = offset*3/2;
          cy = offset*3/2;
            break;
        case 1:
            cx = offset*3/2;
            cy = h - 2*offset - ( h-4*offset ) *pos.m_pos/tick;
            break;
        case 2:
            cx = 2*offset + ( w-4*offset ) *pos.m_pos/tick;
            cy = h-offset*3/2;
            break;
        case 3:
            cx = w-offset*3/2;
            cy = h - 2*offset - ( h-4*offset ) *pos.m_pos/tick;
            break;
        case 4:
            cx = 2*offset + ( w-4*offset ) *pos.m_pos/tick;
            cy = offset*3/2;
            break;
        case 5:
            {
                int ox,oy,dx,dy;
                ox = w-offset*5/2;
                oy = h-offset*5/2;
                dx = offset*5/2;
                dy = offset*5/2;
                cx = ox - ( ox-dx ) *pos.m_pos/tick;
                cy = oy - ( oy-dy ) *pos.m_pos/tick;
                break;
            }
        default:
            return false;
        }
        return true;
}
```

```
//重新设置参数函数
void CTrafficSimDlg::OnButtonReset ( )
{
	if ( m_car != NULL )
		delete [ ] m_car;
	UpdateData ( true ) ;
	m_cnt = 0;
	m_carNum = m_vNum;
	m_car = new Car [ m_carNum ] ;
	srand ( unsigned ( time ( NULL ) ) ) ;
	for ( int i=0;i<m_carNum;i++ )
	 {
		m_car [ i ] .m_position.m_lane = 10;
		m_car [ i ] .m_position.m_pos = 0;
		m_car [ i ] .m_line = rand ( ) % 3 + 1; // 1-3
		m_car [ i ] .m_v = rand ( ) % 5 + 1; // 1-5
		m_car [ i ] .m_start = rand ( ) % 50; //0-49
		m_car [ i ] .m_Time = 0;
		m_car [ i ] .m_Time_L1 = 0;
		m_car [ i ] .m_Time_L2 = 0;
		m_car [ i ] .m_Time_L3 = 0;
		m_car [ i ] .m_Time_L4 = 0;
		m_car [ i ] .m_Time_L5 = 0;
		int r,g,b;
		r=rand ( ) %255;
		g=rand ( ) %255;
		b=rand ( ) %255;
		m_car [ i ] .m_color = RGB ( r,g,b ) ;
	 }

	m_L1Cnt=0;
	m_L2Cnt=0;
	m_L3Cnt=0;
	m_L4Cnt=0;
	m_L5Cnt=0;

	m_maxTime = _T ( "" ) ;
	m_totalTime = _T ( "" ) ;

	m_L1 = 0;
	m_L2 = 0;
	m_L3 = 0;
	m_L4 = 0;
```

```
	m_L5 = 0;

	UpdateData ( false ) ;
}

//设置初始参数函数
void CTrafficSimDlg::OnButtonSetInit ( )
{
	// TODO: Add your control notification handler code here
	UpdateData ( true ) ;
	m_cnt = 0;
	m_carNum = m_vNum;
	m_car = new Car [ m_carNum ] ;
	srand ( unsigned ( time ( NULL ) ) ) ;
	for ( int i=0;i<m_carNum;i++ )
	 {
		m_car [ i ] .m_position.m_lane = 10;
		m_car [ i ] .m_position.m_pos = 0;
		m_car [ i ] .m_line = rand ( )  % 3 + 1; // 1-3
		m_car [ i ] .m_v = rand ( ) % 5 + 1; // 1-5
		m_car [ i ] .m_start = rand ( )  %100; //0-99
		m_car [ i ] .m_Time = 0;
		m_car [ i ] .m_Time_L1 = 0;
		m_car [ i ] .m_Time_L2 = 0;
		m_car [ i ] .m_Time_L3 = 0;
		m_car [ i ] .m_Time_L4 = 0;
		m_car [ i ] .m_Time_L5 = 0;
		int r,g,b;
		r=rand ( ) %255;
		g=rand ( ) %255;
		b=rand ( ) %255;
		m_car [ i ] .m_color = RGB ( r,g,b ) ;
	 }

	m_L1Cnt=0;
	m_L2Cnt=0;
	m_L3Cnt=0;
	m_L4Cnt=0;
	m_L5Cnt=0;

	m_curRumtime = 0;

	::AfxMessageBox ( "设置完成！" ) ;
}
```