文章编号：1009-6744（2010）03-0147-06

# 一种基于路网变化的动态路径规划策略

刘张雷，史忠科*
（西北工业大学 自动化学院，西安 710072）

**摘要：** 就车辆动态时间最短路径诱导问题展开研究，提出了一种便于工程实施的变起点、定目标点的动态行程时间最短路径规划方案．基于该方案，在一种大型方阵图下，就 Dijkstra、A*、D* Lite 等几种动态路径规划算法的计算时间进行了对比分析，针对车载动态导航设备实时性要求高、计算量要求尽可能小的特点，提出了一种基于路网变化的跳变的动态路径规划策略，根据路网中路段权值变化的具体情况，选取更加节省时间的搜索方式．利用东莞市区电子地图和路网历史流量数据进行实验，实验结果表明，该策略可以有效减少路径动态规划的计算时间，有一定的工程应用价值．

**关键词：** 城市交通；动态路径诱导；时间最短路径；LPA*；D* Lite；A*
**中图分类号：** U491 **文献标识码：** A

# A Dynamic Route Planning Strategy Based on Road Network Changes

LIU Zhang-lei, SHI Zhong-ke
(College of Automation, Northwestern Polytechnical University, Xi'an 710072, China)

**Abstract:** A new dynamic time-dependent shortest route planning easily implemented for urban vehicle guidance system is proposed in this paper, based on which computing time of Dijkstra, A*, D* Lite algorithms are compared and analyzed in a large experiment square graph. To meet the real-time needs of the in-vehicle dynamic navigation equipment, a hopping dynamic path planning strategy based on network changes is proposed to reduce the computing time, which picks more effective searching way in light of specific change situation. E-map and history traffic volume data of Dongguan city is used to tests the strategy and the result shows that the strategy is effective and feasible, and the time cost if dynamic route planning can be significantly reduced.

**Key words:** urban traffic; dynamic route guidance; time-dependent shortest path; LPA*; D* Lite; A*
**CLC number:** U491 **Document code:** A

## 1 引 言

动态路径诱导作为先进的交通信息服务系统（ATIS）的组成之一，建立在完善的智能交通信息平台之上，旨在根据路网中实时交通状况合理引导车辆，减少行车延误，使得路网流量趋于平衡．当前国内外动态路径诱导系统的主要工作方式包括实时交通信息发布和动态路径导航两种[1]．

在动态路径导航系统中，车辆动态行车路线的选取，以更切合实际的行程时间为衡量标准，用于降低因道路交通事故、路段改造等突发情况或单

**收稿日期：**2009-07-16 **修回日期：**2010-02-13 **录用日期：**2010-02-23
**作者简介：**刘张雷（1983－），男，山西省晋城市人，硕士生．
* **通讯作者：**zkeshi@nwpu.edu.cn

行、禁行、禁左等交通管制措施所造成的行车延误，提高行程效率．本文就车辆动态时间最短路径诱导问题展开研究，提出了一种既便于工程实施、又能有效减少行车延误的变起点、定目标点的动态行程时间最短路径规划方案．基于该方案，在一种大型方阵图下，就几种动态路径规划算法的实时性进行了对比，针对车载动态导航设备实时性要求高、计算量要求尽可能小的特点，提出了一种基于路网变化的跳变的动态路径规划策略，可以有效减少路径动态规划的计算时间．

## 2 动态行程时间最短路径问题描述

将交通流量随时间动态变化的城市路网表达为时变网络图 $G(V,A,T)$，其中 $V=\{v_1, v_2, \cdots, v_n\}$，是路网所有节点的集合．$A$ 是路网所有边的集合．$T\in[t_0,t_m]$，为路段流量可获知或预测的时间段．边 $A(i,j)$ 的权值 $g_{ij}(t_x)$ 是一个时变量，$g_{ij}(t_x)$ 为 $t_x$ 时刻自节点 $v_i$ 出发，通过路段 $v_i\rightarrow v_j$ 的行程时间．动态时间最短路径问题即可描述为：在图 $G(V,A,T)$ 中，寻找一条 $t_0$ 时刻出发，自起始点 $v_o$ 到目标点 $v_d$ 的路径 $\prod_{v_o v_d}(t_o)=\{(v_o,t_o),\cdots,(v_i,t_i),\cdots,(v_d,t_d)\}$（其中，$t_i$ 为从节点 $v_i$ 出发的时刻），使得沿着该路径，车辆的行程时间最短．

时变网络的时间最短路径问题，并无十分精确的求解方案．Kiseok Sung 等人[2]基于时间窗格(Time Fences)的动态路网展开研究，指出了路段行程时间随时间变化的路网模型(Link Travel Time Model)，不满足实际路网同一路线“先进先出”的特点．提出了一种路段车流速度随时间变化的路网模型(Flow Speed Model)．Evangelos Kanoulas 等人[3]基于这种 Flow Speed Model 路网模型，将各条路段的车流速度按不同时间段进行分类，设置成不同的常数，如高峰时段设置为 0.3 miles/min，其它时段设置为 1 miles/min．在此基础上，给出了求解不同出发时刻，路网两节点间的时间最短路径算法．但相应的模型和算法都有一个前提，即假设路段在某一时间段的行程时间或车流速度是某一定值或某一已知的时变函数，而实际路网的交通状况远比这要复杂，某一路段在各个时间段的车流速度并非一个常量，也无固定的函数表达式．故而不能直接应用于工程实践．基于此，本文提出一种变起点、定目标点的动态行程时间最短路径规划方案．

## 3 变起点、定目标点动态行程时间最短路径规划方案

首先在初始时刻 $t_o$，节点 $v_o$，根据当前各条路段的行程时间，求取一条至目标点的时间最短路径．之后，沿着该路线，在车辆即将到达每个节点 $v_x$ 时，根据各路段的实时权值 $g'_{ij}(t_x)$，进一步修正时间最短路径，重新规划 $v_x$ 至目标节点 $v_d$ 的行程路线．此更新过程不断重复，直至车辆到达目标节点．

该路径规划方案将前方实时的交通状况信息作为反馈，及时修正行程路线．由于在即将到达节点 $v_x$ 时，$v_x$ 与相邻节点构成的路段的权值就是当下时间段的实时权值，从而能使车辆有效地避开拥堵路段．而且重新规划的行程时间最短路线，也在一定程度上基于当前路网的路况信息做出了调整，较之于静态最短路径(以路线物理长度为衡量标准)的诱导方案，更具实时性和合理性．

## 4 算法对比

求解网络节点间最短路径的经典算法包括 Floyd、A*、Dijkstra 等．近年来，人工智能领域，涌现了如 LPA*、D* Lite 等[4,5]新兴的动态路径规划算法，它们都起源于启发式搜索的 A* 算法，应用于动态变化环境中机器人路径探索等方面．

### 4.1 Floyd 算法

其中，Floyd 算法用于求解路网中所有节点对之间的最短路径，时间复杂度为 $O(n^3)$，相当于执行 $n$ 次 Dijkstra 算法，每次求解一个节点到其它所有节点的最短路径，在此，不就该算法展开讨论．

### 4.2 A* 算法

A* 算法，作为启发式算法中很重要的一种，被广泛应用在最优路径求解和一些策略设计的问题中．而 A* 算法最为核心的部分，就在于它的一个估值函数的设计上：$f(n)=g(n)+h(n)$．其中 $f(n)$ 是每个可能试探点的估值，它包括两部分：一部分为 $g(n)$，它表示从搜索起始点到当前点 $n$ 的最小代价的估计值；另一部分，即 $h(n)$，它表示启发式搜索中最为重要的一部分，即当前试探点 $n$

到目标点代价的估值. $h(n)$ 设计的好坏,直接影响到算法能否找到最优解. $h(n)$ 与实际值越接近,估价函数取得就越好. 当 $h(n) \leqslant h_0(n)$, $h_0(n)$ 为实际试探点到目标点的代价值,该算法定能找到最优解.

### 4.3 Dijkstra 算法

Dijkstra 算法实际上是一种特殊的 A * 算法,它的估价函数 $h(n)$ 为 0,在路径探索过程中,没有任何的启发信息,只是盲目地四处搜索,算法的执行过程与 A * 算法类似,不再赘述.

### 4.4 LPA *、D * Lite 算法

LPA * 算法中,根据网络中边的权值的变化,不断地寻找起始点 $n_o$ 到目标点 $n_d$ 的最短路径. 在该算法中引入了两个参量 $g(n)$ 和 $rhs(n)$, 前者对应于 A * 算法的 $g(n)$, 表示从起始点到当前点 $n$ 最小代价的估计值; $rhs(n)$ 满足如下等式:

$$rhs(n) = \begin{cases} 0, & n = n_o \\ \min_{n' \in Pred(n)}(g(n') + c(n',n)), & \text{otherwise} \end{cases} \quad (1)$$

式中 $c(n',n)$ 表示边 $A(n,n')$ 的权值, $Pred(n)$ 是图 $G(V,A)$ 中指向节点 $n$ 的其它节点的集合,也可以叫做节点 $n$ 的所有父节点的集合. 相应地, $Succ(n)$ 是节点 $n$ 的所有子节点的集合. 当 $g(n) = rhs(n)$ 时,则称节点 $n$ 为连续节点;反之,如果 $g(n) \neq rhs(n)$, 则称节点 $n$ 为非连续节点.

LPA * 算法同时设置了一个参量 $Key(n) = \{key_1(n), key_2(n)\}$, 其中,

$$Key_1(n) = \min(g(n), rhs(n)) + h(n,n_d),$$

$$Key_2(n) = \min(g(n), rhs(n)).$$

规定当 $key_1(u) < key_1(v)$ 或 $key_1(u) = key_1(v) \&\& key_2(u) \leqslant key_2(v)$ 时, $Key(u) \leqslant Key(v)$.

该算法第一次的运算过程与 A * 算法相同. 在随后网络中边的权值发生变化后,需要对变化的边 $A(u,v)$ 的子节点 $v$ 的 $rhs(v)$ 值进行更新(UpdateVertex 操作),并将由此产生的不连续节点 ($g(v) \neq rhs(v)$) 插入到 *OPEN* 表中, *OPEN* 表中的节点按照 *Key* 值由小到大排列.

D * Lite 算法可看作是 LPA * 算法的逆向执行,它是沿着目标节点至起始节点的方向进行搜索的,因此它的 $g(n)$ 值表示的是节点 $n$ 到目标节点 $n_d$ 的最小代价值的估计值,相应的 $rhs(n)$ 定义如下:

$$rhs(n) = \begin{cases} 0, & n = n_d \\ \min_{n' \in Succ(n)}(c(n,n') + g(n')), & \text{otherwise} \end{cases} \quad (2)$$

正是基于这一点,D * Lite 算法比 LPA * 算法更适合于变起点、定目标点的动态路径规划,因为一旦起点变化,无论网络中各边的权值是否发生变化,都会导致 LPA * 算法所有节点的 $rhs(n)$ 值发生改变,从而需要进行大量的 UpdateVertex 操作,影响算法的执行效率;而 D * Lite 算法只需要更新受影响节点的 $rhs(n)$ 值.

### 4.5 算法计算时间对比实验

本文从两条途经来实现变起点、定目标点的动态路径规划方案:经典最短路径搜索算法的重复执行和人工智能领域的动态路径规划算法. 构造了一种 $n \times n$ 方阵图,就几种算法的求解效率进行对比. 方阵中可设置相应的障碍点. 在不影响问题分析的前提下,为便于编程,建立以下的模型:

- 设方阵中各个节点的坐标按醒排依次为 $(0,0)$、$(0,1)$, …, $(0,n-1)$, $(1,0)$、$(1,1)$, …, $(1,n-1)$, …, $(n-1,0)$、$(n-1,1)$, …, $(n-1,n-1)$.
- 相邻两节点 $(x_1,y_1)$ 与 $(x_2,y_2)$ 之间的距离为 $\sqrt{(x_1-x_2)^2+(y_1-y_2)^2}$.
- 当某节点设置为障碍节点后,它到所有节点以及所有节点到它的距离都是 $\infty$.

由这种实验模型的特点可知,每个节点最多有 8 个邻接点,与其相关的边最多有 16 条. 改变网络中一些节点的障碍属性(可通过为 0、不可通过为 1),与该节点相应的网络边的权值即发生了变化.

基于上述 50 × 50、100 × 100 两种规模的方阵图,采用三种方法实现变起点、定目标点的最短路径的求取:重复 Dijkstra 算法、重复 A * 算法、D * Lite 算法. 实验程序采用邻接表存储方阵图信息,程序处在循环执行状态,每次计算最短路径之前,提示用户输入起始点和目标点,并从初始化文件中读取所有障碍节点的编号. 通过查看上次最短路径计算结果,可以有针对性地设置下次计算的起始点及障碍节点位置.

该实验重在对比三种求解方法在网络发生动

态变化后,求解最短路径所花费的时间,因此方阵图中的障碍设置要考虑与前一次计算时的障碍的差别,分以下几种情况:

Ⅰ 与前一次的障碍相比,障碍属性变化的节点靠近改变后的起始点.

Ⅱ 与前一次的障碍相比,障碍属性变化的节点靠近目标点.

Ⅲ 障碍属性变化的节点远离上一次规划路线或各节点属性无变化.

Ⅳ 障碍属性变化的节点在上次规划路线的中间部分.

实验中,设置一个折线形阶梯形状的初始障碍节点组,该障碍组中节点数目占节点总数的 10% 左右. 对应每种情况,设置①、②、③三组障碍,每组障碍均在初始形状上做出相应情况的变动,相邻两组障碍均有所不同. 所得实验数据如表 1 和表 2 所示.

**表 1 50 × 50 方阵图中,目标点指定为 2 499,各算法运算时间对比 (单位:ms)**

**Table 1 Comparison of computing time of three algorithms in 50 × 50 Square Map with target vertex number being 2 499 ( Unit: ms)**

| 障碍节点 | | 起始点 | 重复 Dijkstra 算法 | 重复 A * 算法 | D * Lite 算法 | 最短路径长度 |
|---|---|---|---|---|---|---|
| 情况Ⅰ | ① | 0 | 63 | 62 | 65 | 405.296 |
| | ② | 55 | 69 | 71 | 0 | 324.569 |
| | ③ | 380 | 63 | 62 | 31 | 342.154 |
| 情况Ⅱ | ① | 0 | 77 | 72 | 80 | 405.296 |
| | ② | 190 | 68 | 62 | 110 | 282.397 |
| | ③ | 300 | 62 | 63 | 360 | 319.983 |
| 情况Ⅲ | ① | 0 | 62 | 71 | 74 | 405.296 |
| | ② | 55 | 67 | 63 | 0 | 399.882 |
| | ③ | 380 | 62 | 63 | 15 | 340.397 |
| 情况Ⅳ | ① | 50 | 62 | 63 | 64 | 404.882 |
| | ② | 295 | 63 | 65 | 141 | 360.811 |
| | ③ | 520 | 63 | 62 | 47 | 329.154 |

**表 2 100 × 100 方阵图中,目标点指定为 9 999,各算法运算时间对比 (单位:ms)**

**Table 2 Comparison of computing time of three algorithms in 100 × 100 Square Map with target vertex number being 9 999 ( Unit: ms )**

| 障碍节点 | | 起始点 | 重复 Dijkstra 算法 | 重复 A * 算法 | D * Lite 算法 | 最短路径长度 |
|---|---|---|---|---|---|---|
| 情况Ⅰ | ① | 0 | 890 | 781 | 437 | 780.007 |
| | ② | 295 | 912 | 763 | 15 | 628.279 |
| | ③ | 700 | 873 | 782 | 16 | 713.108 |
| 情况Ⅱ | ① | 250 | 890 | 735 | 422 | 669.179 |
| | ② | 520 | 922 | 781 | 1 344 | 693.936 |
| | ③ | 750 | 891 | 812 | 824 | 649.108 |
| 情况Ⅲ | ① | 150 | 891 | 797 | 422 | 729.593 |
| | ② | 500 | 922 | 796 | 31 | 713.936 |
| | ③ | 750 | 922 | 804 | 47 | 663.108 |
| 情况Ⅳ | ① | 150 | 907 | 779 | 438 | 671.451 |
| | ② | 500 | 922 | 781 | 1 015 | 713.936 |
| | ③ | 750 | 906 | 828 | 890 | 663.108 |

### 4.6 实验数据分析

三种方法都能找到当前节点到指定目标点的最优路径.

重复 A * 算法与重复 Dijkstra 算法的比较:

实验中,A * 算法当前节点的估价值 $h(n)$ 取为当前节点到目标节点的欧氏距离,满足估价值小于等于实际值的约束条件. 从实验数据可以看到,重复 Dijkstra 算法在每种情况下的计算时间差别很小,两种规模网络的计算结果中,均方差分别为 4.522 ms 和 16.690 ms,平均计算时间分别为 65.083 ms和904 ms,均方差占平均计算时间的比例分别为 6.95% 和 1.85%. 且在任何一种情况下其

运算时间都大幅高于或接近重复 A * 算法的运算时间,可见重复 A * 算法的搜索效率要高于重复 Dijkstra 算法. 分析其原因,A * 算法正是在启发因子 $h(n)$ 的"引导"下,有针对性地进行节点的扩展,避免了 Dijkstra 算法的盲目性,从而搜寻时间要小于 Dijkstra 算法. 故以下将主要分析重复 A * 算法与 D * Lite 算法的运算结果.

D * Lite 算法与重复 A * 算法的比较:

对表 1 数据:

在情况Ⅰ中,障碍①即是折线阶梯状的初始障碍组,障碍②的设置与障碍①的区别在于,初始阶梯障碍组中靠近起始点 55 的某一层"断裂",由不能穿越变为可以穿越,使得最短路径大大缩短(初始障碍形状下,节点 55 到节点 2 499 的最短路径为 399.882),A * 算法的运算时间 71 ms 远远高于 D * Lite 算法的运算时间 0 ms;根据障碍②下最短路径的计算结果,障碍③的设置使得上次计算所得的最短路线上靠近节点 380 的部分节点的状态由可以穿越变为不能穿越,最短路径长度也受到了影响(障碍②情况下,节点 380 到节点 2 499 的最短路径为 340.397),A * 算法的运算时间 62 ms 远远高于 D * Lite 算法的运算时间 31 ms. 分析原因,由于部分节点障碍属性的变化,对上一次的最短路线产生了影响(可从最短路径长度的变化看出),且这些变化的节点靠近起始节点,在 D * Lite 算法中,只有靠近起始点的那些节点有必要更新其 $rhs(n)$ 值,而对于上一次计算求得的最短路线上,距离目标点较近的节点,它们的 $g(n)$ 值并没有发生改变,可以得到二次利用,不需要像 A * 算法那样从头开始重新计算,故节省了相当一部分的计算时间.

情况Ⅱ中,障碍①也是初始障碍组,障碍②在初始障碍组的基础上,使得靠近目标点 2 499 的某一层阶梯"断裂",最短路径长度受到影响,原先的最优路线发生了变化,D * Lite 算法的运算时间(110 ms)大大高于 A * 算法的运算时间(62 ms). 障碍③重新采用初始障碍组,这样相对于障碍②,靠近目标点 2 499 的"断裂"阶梯,又重新得到"修复",同样使得上次最短路线的走势受到影响,D * Lite 算法的运算时间(360 ms)依旧大大高于 A * 算法的运算时间(63 ms). 分析原因,由于障碍属性变化的节点的位置靠近目标点,在 D * Lite 算法中,导致上次规划路线上靠近目标点的部分节点的 $g(n)$ 值发生改变,相应引发对 $g(n)$ 值受影响的节点的子节点的 $rhs(n)$ 值的更新,且需要更新的节点的覆盖面也相应要大一些,也就是有相当多的节点需要进行 UpdateVertex 操作,前一次运算结果中可供后一次运算二次利用的 $g(n)$ 值所剩无几,反倒是大批节点的 UpdateVertex 操作占用了很多时间. 可见,在这种情况下,D * Lite 算法劣于重复 A * 算法.

情况Ⅲ中,障碍②与障碍①以及障碍③与障碍②虽然有区别,但每次重新计算的最短路径并没有发生改变. 也就是说,新障碍的设置对原有最短路线并没有影响. 可以看到,D * Lite 算法在这种情况下的优势最明显,障碍发生改变后,其两次运算时间 0 ms 和 15 ms 都远低于 A * 算法的 63 ms 和 63 ms. 其原因也在于,原有最短路线上个节点的 $g(n)$ 值并未发生变化,可以直接被 D * Lite 算法二次利用,故与 A * 算法的完全从头算起的做法比起来要节省时间.

情况Ⅳ中,障碍变化的节点,影响了上一次计算的最短路径的路线,且影响部位在路线中段. 与情况Ⅰ有些类似,即有相当部分的节点,尤其是障碍变化节点到起始节点之间的那些节点,需要进行 UpdateVertex 操作,之后再由中间部分开始朝着起始节点的方向进行搜索扩展. 在这种情况下,D * Lite 算法的运算时间高于情况Ⅰ,而低于情况Ⅱ. 与 A * 算法相比,并没有多少优势,甚至可以说是处在劣势.

表 2 中的数据更清晰地反映了上述现象,在Ⅰ、Ⅲ两种情况中,A * 算法的运算时间最多达到 D * Lite 算法的 50 倍之多. 而在Ⅱ、Ⅳ两种情况中,D * Lite 算法运算时间最坏情况下接近 A * 算法运算时间的两倍.

## 5　跳变策略

针对变起点、定目标点动态路径规划中,D * Lite 算法与 A * 算法各自的优缺点,本文提出一种基于路网变化跳变的动态路径规划策略,用于变化路网中,车辆的动态路径规划.

首先判断发生变化的路段在上一次规划路线中所处的位置(对应于情况Ⅰ、Ⅱ、Ⅲ、Ⅳ),如果属于情况Ⅱ、Ⅳ,则将 D * Lite 算法中各个节点的 $g(n_x)$ 和 $rhs(n_x)$ 值置为 $\infty$,对于固定的目标节点 $n_d$,$rhs(n_d) = 0$,随后开始遍历过程. 经过这样的变化后,相当于 D * Lite 算法的第一次执行,也就

完全等同于 A * 算法的运算过程了. 避免了原先 D * Lite 算法中大量节点的 $rhs(n_x)$ 值的更新运算,从而提高搜索效率. 而对于情况Ⅰ、Ⅲ,则完全按照 D * Lite 算法的执行过程继续下去,由于上一次规划路径中有相当一部分节点的 $g(n_x)$ 值并未发生改变,可重新得到利用,故节省了相当一部分时间. 这种对 D * Lite 算法的改进,保留了 D * Lite 算法"记忆"功能的优点,同时摒除了该功能的不利部分. 改进算法的执行类似于在 D * Lite 和 A * 算法中间进行自动跳变,根据路网中路段权值变化的具体情况,选取更加节省时间的搜索策略.

改进 D * Lite 算法的流程如图 1 所示.

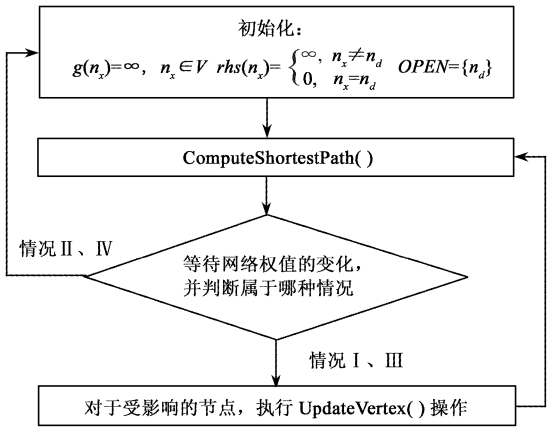


图 1 改进的 D * Lite 算法

Fig.1 Optimized D * lite algorithm

本文进一步就改进后的 D * Lite 算法,在 100 × 100 方阵图下四种障碍变化情况,进行了验证实验,表 2 中四种情况 12 个障碍下,其运算时间(单位:ms)依次为 437、15、16、780、840、438、422、31、47、438、800、840. 结果表明改进后的算法完全可以根据网络变化情况,实现计算策略的自动跳变,从而选择更节省时间的计算途经求解最优路径.

## 6 应用实例

本文基于东莞市区各路段两年的历史流量数据,结合东莞市电子地图,采用文中提出的变起点、定目标点的动态路径规划方案,开发了面向 WinCE 客户端的动态路径诱导系统进行实验仿真. 路网中各路段的权值指定与其实时的行程时间进行绑定. 严格意义上,各路段的权值在实时地发生着改变,但某些变化并未超出驾驶员的容忍范围,文献[6]提出了一种设定容忍阈值的观点,即与前一时间段相比,只有在当前时段路段的权值的变化超过一定的比例的情况下,才认为该路段的权值发生了改变. 本系统的设计取容忍阈值为 35%.

基于该仿真平台,以 5 分钟为一个时间段,根据各路段的实时流量,采用本文提出的跳变的动态路径规划策略,进行车辆的动态路径规划实验. 结合实际的路况信息,比较了相邻两个时间间隔下的规划结果,分析结果表明,改进后的算法能够根据路网实时流量数据的变化,有效避开车流量大的路段,及时地完成时间最短路径的动态规划. 即使在路网流量有较大变化时,其计算时间也无明显延滞.

## 7 研究结论

本文就城市路网动态时间最短路径问题展开研究,提出了一种便于工程实施的变起点、定目标点的动态路径规划方案. 基于该方案,在一种较大规模的方阵图网络下,就几种求解算法的实时性进行了对比分析,并提出了一种基于网络变化的跳变的动态路径规划策略. 仿真实验表明,该跳变策略能够有效减少运算时间,有一定的工程应用价值.

**参考文献:**

[1] US Department of Transportation. Traveler information system in europe[EB/OL]. http://international.fhwa.dot.gov/travelinfo/contents.cfm, 2008.8.2009.7.

[2] Kiseok Sung, Bell M G, Seong M, et al. Shortest paths in a network with time-dependent flow speeds[J]. European Journal of Operational Research, 2000,121(1): 32 - 39.

[3] Evangelos Kanoulas, Du Y, Xia T, et al. Finding fastest paths on a road network with speed patterns[C]. 22nd International Conference on Data Engineering, 2006:10 - 19.

[4] Dave Ferguson, Likhachev M, Anthony Stentz. A guide to heuristic-based path planning[C]. International Conference on Automated Planning and Scheduling (ICAPS), 2005.

[5] Sven Koenig, Maxim Likhachev. D * Lite[C]. Eighteenth National Conference on Artificial Intelligence, 2002: 476 - 483.

[6] 姜桂艳, 郑祖舵, 白竹, 等. 基于记忆机制的动态交通路径优化算法[J]. 吉林大学学报(工学版),2007, 37(5): 1043 - 1048. [JIANG Gui-yan, ZHENG Zu-duo, BAI Zhu, et al. Dynamic traffic path optimization algorithm based on mnemonic mechanism[J]. Journal of Jilin University (Engineering and Technology Edition), 2007, 37(5): 1043 - 1048.]