DOI:10.13282/j.cnki.wccst.2010.09.020

文章编号:1673－4874(2010)09－0084－006

基于增量搜索的车辆动态路径规划算法研究

黄仝宇,胡　刚,傅　惠
(广东工业大学机电工程学院,广东　广州　510006)

摘　要:*D* * *Lite* 算法是一种非常高效的增量搜索算法,适合于动态环境下的路径规划。文章基于 *D* * *Lite* 算法的原理,对该算法的性能及应用于车辆路径规划的适用性进行改进,提高了算法的稳定性和结果的可靠性,并通过试验和算法评价验证了改进的 *D* * *Lite* 算法在复杂、动态环境下规划路径的有效性。

关键词:动态路径规划;*D* * *Lite* 算法;改进
中图分类号:*U*491.1+2　　文献标识码:*A*

Research on Determination of Car Path Under Vehicle Volume In-crement Study

HUANG Tong-yu, HU Gang, FU Hui
(Guangdong Industrial University, College of Mechanic Engineering, Guangzhou, Guangdong, 510006)

Abstract: D * Lite method is a very efficient calculating method for incremental search, it is a suitable method for dynamic road path planning. Based on the principle of D * Lite, the article improves the existing method and raise the reliability for road path management. It also proves the new method is capable of performing by conducting experiments and evaluations.

Key Words: Dynamic road path planning; D * Lite method; Improving

作者简介
黄仝宇(1984—),男,广东人,在读研究生,研究方向为智能交通、软件开发;胡　刚(1964—),男,博士、教授,主要从事智能交通仿真系统和基于实时堵塞信息的动态导航系统的研究工作;傅　惠(1982—),男,博士、副教授,主要从事交通信息智能预测理论与方法以及路径诱导算法的研究工作。

0　引言

随着城市建设的发展,道路交通问题变得越来越突出。为了解决出行者的路径规划问题,很多学者提出了路径规划的算法,如 Dijkstra、A * 算法等。但是这些算法大多应用于静态网络,也就是说,道路路径是在出行者出发前规划好的,路网中道路的权值是静态的、确定不变的。但是,在实际的路网中,道路的

权值并不是固定不变的，而是随时间的变化在不断变化的。传统的处理方法是对新的格局再次进行彻底的搜索，计算量大且效率不高。

1 A* 算法

A* 算法是最常用的优化路径规划算法，它利用启发性信息引导优化路径的搜索，提高了规划的效率，为静态环境下具有完全信息的路径提供了有效的解决方案。由于路况的动态性、信息的不确定性和不完全性，车辆在沿路径行进的过程中，随着实时交通信息的获得，经常会发现有些路段的权重发生了变化，剩下的路径需要重新规划，而且要求路径重新规划的速度足够快，以满足车辆行驶的实时性要求，这使得 A* 算法不能满足实时路况下车辆重新规划的要求。

2 动态路径规划算法

一些实时系统常常需要根据外界环境的变化不断修正自身，这样就会产生一系列变化较小的相似问题，此时应用增量搜索[1]（Incremental Search）将会非常有效。增量搜索的基本思想是利用先前的搜索信息来提高本次搜索的效率。主要的增量搜索算法有 LPA* 算法以及在它的基础上发展的 D* Lite 算法和 DD* Lite 算法等[2]。LPA* 算法的实质是在 DynamicS WSF-FP 算法的基础上加入了 A* 算法的思想，而 D* Lite 算法则是结合了 D*（Dynamic A*）和 LPA* 算法的优势[3]，它既利用先前的搜索信息来提高本次搜索效率，又通过使用启发信息进一步压缩搜索树的大小，与重复利用 A* 算法相比，搜索效率更高。

3 D* Lite 算法

D* 算法虽然可以实现动态环境下路径规划和重新规划，但是它比较复杂，原理不易被研究。Sven Koening 和 Maxim Likhachev 提出了 D* Lite 算法[4]。D* Lite 算法具有与 D* 算法相同的规划结果，但是算法更为简单，速度上略有改善。D* 和 D* Lite 算法都是随着状态之间边代价的变化维护着一个开始状态和一定的目标状态之间的最小代价路径，弧代价的增加或减少以及不同的开始状态都可以处理，适用于动态路径规划[5]。

3.1 D* Lite 算法描述

在 D* Lite 算法中，问题空间被表示为一组结点和连接它们的边，每条边都有它的费用值。S 表示路网中的所有结点组成的集合。D* Lite 算法仍然是寻找起点 $S\_start \in S$ 到终点 $S\_goal \in S$ 费用最小的路径。为了实现这个目标，它保存了两个重要的值：一个是从每个结点到 s 再到目标位置的估计费用 $g(s)$（目标位置不变，$g(s)$可重复利用）；另一个是估计值 *Right Hand Side*，记作 $rhs(s)$，它满足：

$$rsh(s)\begin{cases} 0 & if = s = s\_goal \\ min_{s' \in Succ(s)}(c(s,s')+g(s')) & otherwise \end{cases} \quad (1)$$

$Succ(s)$表示结点 s 的所有后向结点集合，$Pred(s)$表示结点 s 的所有前向结点集合。$0<c(s,s')<\infty$表示从结点 s 移动到结点 s'的出行费用。

如果 $rhs(s)=g(s)$，则称结点 s 为连续结点（*consistent*），否则，为非连续结点（*inconsistent*）。如果所有的结点都是连续的，则说明从上次路径优化结束到现在为止，路径上所有路段的费用均未发生变化，上次路径规划的结果仍然有效，无需重新规划路径。

与 *A** 算法一样，*D* Lite* 利用一个启发函数和一个优先队列来指导搜索方向和高效地更新价值。启发函数 $h(s,s')$估计从起点 s 到 s'的一条最优路径的价值，对于所有的结点 $s,s',s'' \in S$，均应满足 $h(s,s') \leqslant c^*(s,s')$和 $h(s,s'') \leqslant h(s,s')+c^*(s',s'')$。表里面维护着不连续的结点，这些结点需要更新成为连续结点。

队列里面的结点按照 *Key* 的值来排序（在后面的验证试验里面，优先队列采用 *hashtable* 来

实现)：

$$key(s)=[k1(s),k2(s)]$$
$$=[min(g(s),rhs(s))+h(sstart,s),\quad(2)$$
$$min(g(s),rhs(s))]$$

为了节省每次比较 *key* 值的时间，队列按照升序排列好。如果 $key(s)$ 的优先级比 $key(s')$ 的要低，记作 $key(s)<key(s')$，并同时满足 $iff k1(s)<k1(s')$ 或者 $k1(s)=k1(s')$ 和 $k2(s)<k2(s')$。*D* Lite* 算法从升序排列的优先队列里面取出结点来扩张结点，并更新其前继结点的 *g* 值和 *rhs* 值。当结点状态再次变为连续状态时，将其从结点队列中剔除，直到队列中排在最前面的状态变为当前位置。此时，便完成了一次动态路径优化[6]。

3.2 算法过程

D Lite* 算法用反向 *A** 的搜索算法，从目标点向起始点构造一条初始路径。当检测到环境的改变，*D* Lite* 算法更新受到环境变化影响结点的 *rhs* 值，将变为 *inconsistent* 的结点放回表中，经过扩展计算向搜索状态空间中其他结点传播路径代价增加或者减少的信息。最后，*D* Lite* 算法只要循环处理那些受到环境变化影响的结点，即可计算出从当前位置到目标点的最优路径。这个循环计算过程直到表中没有优先级小于起点 *S_start* 优先级的结点为止，如果计算结束后 *S_start* 的 *rhs* 值为无穷，则不存在到达目标点的路径。

算法过程描述如下：

第一步：初始化起点和终点的 *g* 值和 *rhs* 值。起点的 *g* 值和 *rhs* 值都设为无穷，而终点的 *g* 值为无穷，*rhs* 值则为 0。令 *OPEN* 为空表，由于终点的 *g* 值和 *rhs* 值不相等，将该结点插入队列中。

第二步：目标结点作为搜索树的根部不变化，反向搜索得到一条初始最短路径并推荐给驾驶员。

第三步：检测是否达到终点，如果还没有到达终点，则进入第四步；如果已经到达终点，则跳出循环。

第四步：沿推荐路径到下一状态，更新 *S_start* 以反映车辆的位置变化[7]。

第五步：检测边代价是否有变化，假如发生变化，执行第六步操作；假如没有发生变化，说明上一次的规划结果依然有效，无需重新规划路径。

第六步：更新边上的权值和相关结点的状态。移除 *OPEN* 表里面 *key* 值最小的结点 *s* 并更新结点 *s* 的所有后继结点的状态，重新计算最短路径并返回第三步。

第七步：当前位置为终点，到达目的地，算法结束。

这个版本的 *D* Lite* 正确性已经得到证明[7]，但是仍然可以通过数据结构的优化来提高算法的效率，减少搜索时间。此外，*D* Lite* 算法运用于车辆路径规划应考虑问题域的特点，对其进行改进。

4 改进 *D* Lite* 算法

4.1 提高 *D* Lite* 算法的性能

标准 *D* Lite* 算法用于路径规划有几处较影响速度，比如程序在第二步和第六步搜索最优路径和增量传递变化的路权的时候，需要进行复杂的内存分配，这个过程比较耗费时间；其次在判断结点是否在队列里面的时候，往往需要对整个队列进行遍历，这也是一个耗时的操作。对于一个较大的路网，时间耗费将会非常大，影响了规划的效率。

(1)算法中的 *OPEN* 表可以采用 *HashTable* 或者二叉树来代替。为了减少内存的分配，可以将保存历史最短距离的数组设计为如下类型的数组。

```
public class NodeType {
    int x, y, g, rhs, nodeId;
    float h;
    bool IsOPEN;//判断结点是否在优先队列里面
    NodeType preNode;//前继结点
    NodeType nextNode;//后续结点
}
```

preNode 指向该结点的前继结点，而 *nextNode* 指向后续结点。实例化一个以 *Nodetype* 为类型的二维数组，在扩展结点的时候可以容易地

找到相关结点并进行计算。另外 *IsOPEN* 指明该结点是否插入队列中。*IsOPEN* 是一个布尔值，*true* 为在 *OPEN* 表里面，*false* 为未插入或者已经移除。这样的细节处理，使每次只要访问结点的属性即可知道结点是否存在于表中，没有必要每次都去遍历整个表。

(2)当结点的 g 值不等于 *rhs* 值的时候，需把该结点按照 *key* 值大小插入优先队列里面。当队列里面的节点数目较多的时候，逐个比较 *key* 值的大小也会对整个搜索时间产生一些影响。快速排序(*Quicksort*)将整个队列里面的数据分割成独立的两部分，其中一部分的所有数据都比另外一部分的所有数据小，然后通过比较中间值的大小快速定位到某个区间，经过递归运算便可迅速找到插入位置。

4.2 适合车辆路径规划的 $D*Lite$

$D*Lite$ 算法最初用于智能机器人的寻路，在这些机器人上安装传感器来探测周边的环境，当检测到环境的变化就要重新规划路径，以适应外界环境的不断变化。与这些应用不同，$D*Lite$ 算法用于动态车辆路径规划的时候，应该考虑问题域的特征，作一些优化：

(1)在规划好一条最佳路径后，将一部分会影响最终计算结果的路段 *ID* 存储起来。在 $D*Lite$ 算法中，每个结点是由前继结点方向优先搜索得到的，所以我们将先前规划好的最佳路径中的所有路段的 *ID* 以及其邻接路段的 *ID* 存储起来，只有在这些路段的权值变化时才重新规划最佳路径。如图 1 所示，得到一条最佳路径之后，记录下路段 1、路段 2、路段 3、交叉口 A 的邻接路段和交叉口 B 的邻接路段的 *ID* 号，当这些路段的阻抗发生变化才考虑重新规划路径。

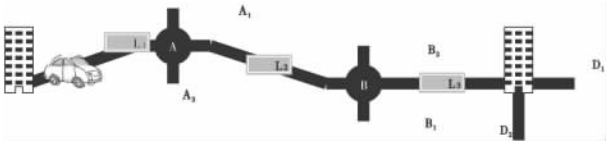

图 1 路网示意图

(2)要过滤掉行程时间变化微小的路段对规划的影响，如果权值的变化在一个可接受的范围内，不执行重新规划，这既符合出行者的心理又可以减少不必要的计算次数，节省了内存资源。

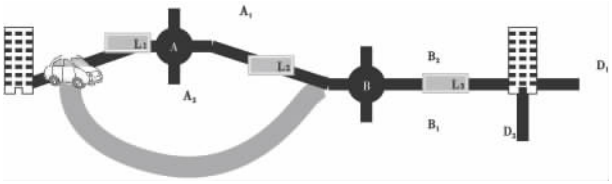

图 2 重新规划示意图

如图 2 所示，按照 $D*Lite$ 算法计算出一条从目的地到车辆的所在位置的最佳路径。当目的地的对应路段(D_1，D_2，L_3)的权值变化较明显，重新规划的时候才有可能规避 B 点，而当在 B 点相关的路段(B_1，B_2，L_2)变化较明显时，重新规划时规避 A 点，依次类推[8]。通过这样的处理，使得重新计算的针对性更强，提高了 $D*Lite$ 的稳定性，同时在一定程度上减少了冗余的计算。

5 试验和算法评价

本文用实验证明改进的 $D*Lite$ 算法在动态环境里面进行路径规划、重新规划的有效性。由于网格图能够直观、有效地反映搜索过程中遍历过的结点，方便模拟路权的变化，而且网格图可以看作是抽象的路网，所以本文通过控制格子的权重来执行规划引擎。

5.1 动态网格试验

八方向的网格例子如图 3～5 所示。图上斜线的格子表示围墙，围墙是不可跨越的；白色的格子表示通路，没有障碍；带阴影的格子表示障碍物，颜色越深表示阻值越大。左上角的格子表示起始位置(*S_start*)，右下角的格子表示目标位置(*S_goal*)。任务是从起始位置开始，寻找一条到达目标位置的最短路径。黑色的格子串表示路径，已经在图 3 中标出。随着时间的改变，网格上

的格子状态也发生了变化，规划引擎可以随着相关格子状态的变化来重新规划路径，从而每次得到的结果都是最优的。改进的 $D*Lite$ 算法很好地满足了这个实时规划的问题。

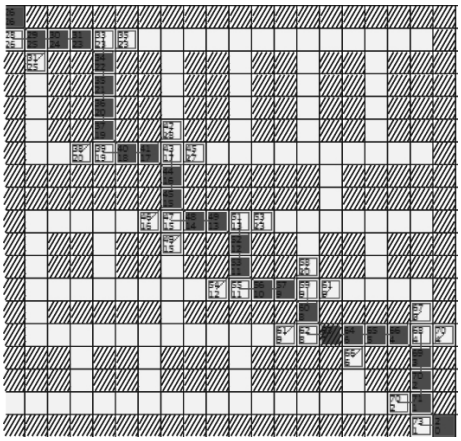

图 3　“道路”畅通无阻的状态下得到的规划路径网格示意图

当该算法执行第一次查词时，它将所有结点的 g 值和 rhs 价值初始化为无限大。实际上我们不能将一大幅网格图的每个格子都进行初始化，而是在搜索时只对我们遇到的格子进行初始化。在接下来的每次迭代计算过程中，每个格子都有 k_1 值和 k_2 值，它们决定着 key 的值。如果格子的 g 值和 rhs 值不相等，就把这个结点按顺序插进队列里面。

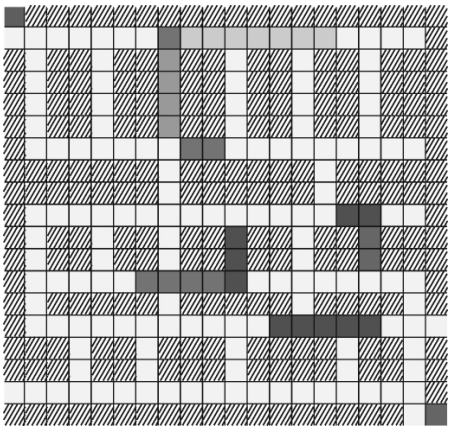

图 4　某些“路段”的权值发生改变网格图

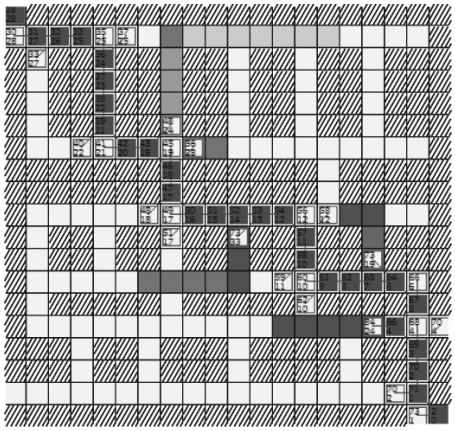

图 5　权值变化后，重新规划得到的路径网格图

图 4 为某些“路段”权重发生改变的网络图。图 3～4 共有 30 个格子的状态发生改变。本程序对权值的变化做了过滤处理。当程序获知相关路段上(10,11)、(11,11)、(12,11)这几个格子的权值变化为足够大的时候，就会重新规划路径。我们先检测权值发生改变的格子周边格子的估计值 (g,rhs)，这些是潜在的最受其影响的估计值。虽然有些邻近的格子不受该变化的影响，但这些结点的起始距离和 rhs 值会随之变化。由于更新的过程中某些格子变得局部不一致，故下一步就是更新到这些局部不一致的格子。这种搜索方式可以避免拜访许多不受该变化影响的不必要的格子。$D*Lite$ 算法可重复利用最后一次搜索的计算结果，并通过逐步更新不一致的结点使路线的重新计算更加方便快捷。图 5 显示运用 $D*Lite$ 算法执行重新规划后所得到的路径。

5.2　$A*$ 算法、$D*Lite$ 算法与改进的 $D*Lite$ 算法比较

在实验中，对不同规模的“路网”分别执行 $A*$ 算法、基本的 $D*Lite$ 算法和改进的 $D*Lite$ 算法。每次测试分为两个部分：(1)将所有可通格子的权值设为 1，分别执行规划引擎，得到一组时间值；(2)以同样的方式赋予某些格子一些权值，触发规划引擎，得到另一组数据。实验结果见表 1。

表 1　实验结果表(单位:s)

网格规模	格子改变个数	$A*$			基本的 $D*Lite$ 算法			改进的 $D*Lite$ 算法		
		初次规划时间	重新规划时间	合计时间	初次规划时间	重新规划时间	合计时间	初次规划时间	重新规划时间	合计时间
30×30	30	0.079 8	0.078 5	0.158 3	0.078 7	0.012 1	0.090 8	0.077 5	0.008 9	0.086 4
30×30	30	0.078	0.072 4	0.150 4	0.076 6	0.011 8	0.088 4	0.077 2	0.008 1	0.085 3
30×30	30	0.077 2	0.073 2	0.150 4	0.081	0.010 8	0.091 8	0.078 1	0.008 3	0.086 4
30×30	30	0.078 4	0.074 9	0.153 3	0.079 6	0.011 6	0.091 2	0.080 3	0.007 9	0.088 2
30×30	30	0.079 4	0.075 6	0.155	0.077 5	0.009 7	0.087 2	0.076 7	0.009 7	0.086 4
60×60	60	0.123 6	0.160 3	0.283 9	0.122 1	0.070 1	0.192 2	0.121	0.042 4	0.163 4
60×60	60	0.123 3	0.153 7	0.277	0.121 5	0.077 6	0.199 1	0.120 7	0.047 1	0.167 8
60×60	60	0.123 8	0.152 8	0.276 6	0.125 2	0.071 3	0.196 5	0.121 5	0.042 9	0.164 4
60×60	60	0.121 2	0.162 8	0.284	0.132 8	0.075	0.207 8	0.119 3	0.046 5	0.165 8
60×60	60	0.141 9	0.149 9	0.291 8	0.121 9	0.072 7	0.194 6	0.120 9	0.047 1	0.168
100×100	100	0.283 2	0.342 2	0.625 4	0.273 1	0.107 5	0.380 6	0.269 9	0.067 5	0.337 4
100×100	100	0.296 7	0.333 4	0.630 1	0.278 7	0.102 6	0.381 3	0.270 2	0.065 5	0.335 7
100×100	100	0.286 6	0.338 6	0.625 2	0.282	0.098 4	0.380 4	0.268 3	0.068 4	0.336 7
100×100	100	0.287 5	0.330 2	0.617 7	0.278 7	0.105 3	0.384	0.269 5	0.062 6	0.332 1
100×100	100	0.282 1	0.331 9	0.614	0.283 4	0.105 5	0.388 9	0.268 7	0.065 3	0.33 4

标准的 $A*$ 算法是对新的环境格局再次进行彻底的搜索,从而得到新的路径;而基本的 $D*Lite$ 算法虽然能够处理动态环境下的路径规划与重新规划,但是未针对车辆导航这一领域进行优化。然而通过试验我们发现,在初次规划路径时,改进的 $D*Lite$ 算法、$D*Lite$ 算法的效率与 $A*$ 算法相当;在路况发生改变,相关路段的权重发生明显变化并需要重新规划的时候,改进的 $D*Lite$ 算法略优于 $D*Lite$ 算法,并明显优于 $A*$ 算法。当搜索规模较大的时候,这种新型的 $D*Lite$ 算法的优势更加明显,减少了搜索时间,符合动态车辆导航实时性要求高的特点。

6　结语

$D*Lite$ 算法利用先前的搜索信息来提高本次搜索的效率,在此基础上加入启发信息来选择下一个扩展结点,是一种非常高效的重搜索算法,适用于动态环境下的路径规划。根据车辆路径规划问题领域的特征,本文对 $D*Lite$ 进行改进,使改进后的算法能够适用于实时路况下的动态车辆路径规划,同时提高了算法的稳定性和结果的可靠性。本文最后通过试验验证了改进 $D*Lite$ 算法的优异性,实现了动态环境下的路径规划,并与 $A*$、$D*Lite$ 算法作了比较。当环境改变以致影响原有规划结果的时候,本文所讨论的改进型 $D*Lite$ 算法只需要重新计算部分状态量,而不需要在整个环境中重新规划,提高了算法的运行效率。

参考文献

[1] *Koenig S, Likhachev M, Furcy D. Lifelong Planning A* [J]. *Artificial Intelligence Journal*, 2004, 155 (1-2): 93-146.

[2] *Ayorkor Mills-Tettey G, Anthony Stentz, Bernardine Dias M. DD* Lite: Efficient incremental search with state dominance* [R]. *Technical Report, CMU-RI-TR-07-12, Robotics Institute, Carnegie Mellon University*, 2007.

[3] *Koenig S, Likhachev M, Liu Y, Furcy D. Incremental heuristic search in artificial intelligence* [J]. *Artificial Intelligence Magazine, Summer*, 2004, 25 (2): 99-112.

[4] *Koenig S, Likhachev M. Improved fast replanning for robot navigation in unknown terrain* [C]. *In: Proc. of the 2002 IEEE Int'l Conf.*

[5] *Frigioni D, Marchetti-Spaccamela A, Nanni U. Fully dynamic algorithms for maintaining shortest path trees* [J]. *Journal of Algorithms*. 2000, 34 (2): 251-281.

[6] *Stentz A. The focused D* algorithm for real-time replanning* [C]. *In: Proc. of the Int'l Joint Conf. on Artificial Intelligence. IJCAI. San Fransisco: Morgan Kanfmann Publishers*, 1995. 1652-1659.

[7] 焦立男. 地面移动机器人运动规划与运动协调的若干算法研究 [D]. 南京理工大学, 2008.

[8] 王泉啸, 蔡先华. 动态最佳路径算法研究 [J]. 城市勘测, 2009 (1).

[9] 田鹏飞, 王剑英. 动态最短路径算法及仿真 [J]. 计算机仿真, 2007, 24 (6).

[10] 李　枫, 迟洪钦. 方格取数问题的动态规划算法 [J]. 计算机应用与软件, 2009, 26 (6).

基金项目:广东省科技计划项目 基于 *RFID* 的交通信息获取与动态导航服务关键技术》(项目编号:2009*B*010800052)

收稿日期:2010-08-23