第33卷 第10期 计 算 机 工 程 2007年5月
Vol.33 No.10 Computer Engineering May 2007
·软件技术与数据库· 文章编号：1000—3428(2007)10—0036—02 文献标识码：A 中图分类号：TP312

机器人路径规划中的双向 Dijkstra 二叉树算法

周 赟，王腾飞，戴光明
(中国地质大学计算机学院，武汉 430074)

摘 要：在分析现有路径规划和碰撞检测方法的基础上，提出了一种新的机器人路径规划方法：双向Dijkstra二叉树算法。在机器人路径规划中应用传统的Dijkstra算法时间复杂度是$O(n^2)$，应用该文提出的算法进行路径规划的时间复杂度为$O(n\log_2 n)$。通过一些数据的检测，验证了在机器人路径规划中，尤其是在测试数据较多的情况下，该算法可以有效提高效率。

关键词：机器人路径规划；最短路径；双向 Dijkstra；二叉树

Bi-directional Dijkstra with Binary Tree Sorted Algorithm in Robot Path Plan

ZHOU Zuan, WANG Tengfei, DAI Guangming
(School of Computer, China University of Geosciences, Wuhan 430074)

【Abstract】 On the basis of analysis of current path plan methods and collision examining methods, a new robot path plan method is put forward: bi-directional Dijkstra with binary tree sorted algorithm. It is well known that Dijkstra algorithm solves the path plan problem in time $O(n^2)$. As an improvement on Dijkstra algorithm, because it begins from start point and end point at one time when it executes the algorithm, and sorts the set of the points which have not been marked by binary tree, the algorithm solves path planning problem in time O (nlog n). And the enhancement on the efficiency in robot path plan with the algorithm has been proved by testing some data, especially in the situation where the number of testing data is very large.

【Key words】 Robot path plan; Shortcut; Bi-directional Dijkstra; Binary tree

在机器人路径规划中，常常需要考虑给定两点 S、T 和若干障碍物，要求找出机器人从 S 到 T 并避开所有障碍物的最短路径，也就是所谓的 ESPO 问题。对于二维的 ESPO 问题，可以将两点和各障碍物之间的关系转换成带权图，从而将二维 ESPO 问题转化成带权图的最短路径问题。

最短路径算法有很多，包括图论基本方法、启发式搜索方法、动态规划方法、神经网络方法等。传统的最短路径算法主要有 Floyd 算法和迪杰斯特拉(Dijkstra)算法等。其中 Floyd 算法用于计算网络中每一对顶点之间的最短路径；Dijkstra 算法用于计算一个源节点到所有其它节点的最短代价路径。Dijkstra 算法由于适应网络拓扑的变化，性能稳定，因此在计算机网络拓扑路径选择得到广泛的应用。

但是它们的时间复杂度与图的顶点数的平方成正比，在顶点较多的情况下就难以满足实际运算的需要。比如在我们的研究过程中，当机器人要避开的障碍物较多时，将障碍物的信息转化为无项带权图的时候顶点数目较多，运行所需的时间就较多。本文提出的双向 Dijkstra 二叉树算法就是为解决上述问题对 Dijkstra 算法的一种改进，经实验证明，在顶点数较多的图上是一种很理想的最短路径算法。

1 经典 Dijkstra 算法简介

Dijkstra 算法用于计算一个源节点到所有其它节点的最短代价路径，它是按路径长度递增的次序来产生最短路径的算法。

1.1 Dijkstra 算法的网络表示方法

交通网络图中的路段和节点，可以抽象为边和顶点，这样整个交通网络就抽象为一张图。对于一张 N 个节点的图，用一个二维 C[N][N]数组存储网络中的数据，若节点 I 与节点 J 之间有边，则 C[I][J]中存储节点 I 到节点 J 的边的权值。

1.2 Dijkstra 算法的实现

标号方法是大多数最短路径算法的核心过程，Dijkstra 算法就是采用这种方法进行最短路径搜索。下面是 Dijkstra 算法的实现流程。其中数组 D[i]记录当前节点 I 到源点 S 的最短路径。初始化 D[i]=MaxInt，D[s]=0；mark[i]是个布尔数组，若已经求得从 S 到 I 的最短路径，则将 mark[i]=true，对于所有的 I，初始化 mark[i]=flase；C[i][j]表示图中从节点 I 到节点 J 的路径长度，如果从 I 到 J 没有通路，则 C[i][j]=MaxInt。

```
(1)While mark[t]=flase do begin
(2)  选取节点 u，满足 u=min{ D[x]| 其中 mark[x]= false },
(3)        for 每个从 u 出发的节点 v do
(4)              D[v]<- min{ D[v]，D[u]+c[u][v]};
(5)      mark[u] <- true;
(6)  end while
```

2 双向 Dijkstra 二叉树算法

2.1 使用二叉树对当前最优节点选择的改进

从上述经典 Dijkstra 算法流程可知，该算法的效率取决于算法的(2)～(4)步，为了提高效率，可以使用优先队列来存

作者简介：周 赟(1985－)，男，本科生，主研方向：信息安全等；王腾飞，本科生；戴光明，博士、教授

收稿日期：2006-06-06 **E-mail：**forward_zuan@163.com

储满足 mark[x]=false 的所有节点。通常，可以用二叉树来表示这个优先队列。同时，规定二叉树每个结点的优先级高于其自己子节点的优先级，具体到这个问题来说，就是 D[i]值小于子节点值。使用数组 T[n]来存储二叉树，n 为数组元素个数，节点 T[i]的左右子树分别为 T[2i]和 T[2i+1]，则节点 T[1]为 D[i]中最小的元素.

对于二叉树定义如下操作：

Push(x)：往优先队列后面插入一个新节点，并不停地与比其优先级低的父节点换位，称为向上调整，使得二叉树仍然满足“父节点优先级低于子节点优先级”的性质。

Pop()：删除优先队列的根结点，此节点即为当前最优节点。将最后一个叶子节点作为新的根节点，然后不停地与比其优先级高的子节点交换，称为向下调整。

Update(i)：先将节点 i 不停地向上调整，然后不停地向下调整。这是为了实现算法第 4 步优先队列中节点优先级改变时，对节点的调整，使其仍然满足性质。

由于每次都将新节点都加在数组T末尾，因此二叉树高度始终为$\log_2 n$。调整时最坏是将一个节点从最低层升到最高层，所以最坏情况下需进行 $\log_2 n$次操作，平均情况下为 $\frac{\log_2 n}{2}$，因此上述操作时间复杂度为$O(\log_2 n)$。所以使用优化队列的Dijkstra算法复杂度为$O(n\log_2 n)$，比经典算法有了较大的改进。

2.2 双向 Dijkstra 二叉树算法

(1)算法思路

通常在基本 Dijkstra 算法中，要找到到终点的最短路径，必须先找到所有比终点最短路径短的所有最短路径，在顶点分布均匀的情况下，所要找到的最短路径条数和两顶点之间的距离的平方成正比。如果从两个端点同时开始搜索，到相遇时(即从两个端点都找到了到达同一顶点的最短路径)所要找到的最短路径条数只有基本 Dijkstra 算法的一半，然后再从结果中找出最短路径，从而在图的顶点较多时能大幅度缩短搜索时间。

若起点为 S，终点为 T，搜索时在节点 K 相遇，则 SK＋KT 为从 S 到 T 的一条路径，然后与已保存的最短路径比较，看是否要更优，是则保存为最短路径。若搜索时当前路径长度长于最短路径，舍弃该节点，不进行拓展。

(2)算法实现

1)图的表示

定义一个数据类型：

```
 typedef struct ttype
{
        int id;                 // 与其相连的节点号
        double cost;            // 边的权值
        struct ttype *next;     // 下一个相邻的节点信息
}vtype;
 Vtype map[maxnode];            // map[i] 表示第 I 个节点的信息
```

2)算法过程

配合上述的二叉树算法，本文的双向 Dijkstra 二叉树算法如下。使用一个 Flag[n]表，表示节点 n 是否已经被访问过。Flag[i]初始为 0。1 表示被正向访问过，-1 表示被反向访问过。建两个堆 f、b，分别代表从正、反向计算时的优先队列。堆初始时各只有一个元素，分别为计算的出发点 s 和 t。

```
While (f、b 不都为空) do begin
    If (f 不为空) then begin
        Out<-  f.pop();
      If (out 花费大于最短路径长度) then 不处理节点 out
        Else if (out 逆向计算时已经被访问过) then
            比较并更新最短路径
        Else begin
            标记 Flag，out 被正向计算访问过
            For 每个从 out 出发可到达的节点 do
                更新优先队列的值
        End begin
        End if
    End begin
    If (b 不为空) then begin
        Out<-  b.pop();
      If (out 花费大于最短路径长度) then 不处理节点 out
        Else if (out 反向计算时已经被访问过) then
            比较并更新最短路径
        Else begin
            标记 Flag[out]被逆向计算访问过
            For 每个从 out 出发可到达的节点 do
                更新优先队列的值
        End begin
        End if
    End begin
End while
输出最短路径
```

(3)时间复杂度分析

本算法从两个方面降低经典算法的时间复杂度：(1)两端开始的搜索，每一端要访问的节点平均不会超过n/2；(2)在标记完一个节点并更新为标记节点到起点或终点的路径长度后，要从未标记节点中选取到起点或终点路径长度最短的顶点。本算法采取优先队列的方法，与经典算法相比，只需要$O(\log_2 n)$时间。综上所述，本文算法的时间复杂度为$O(n\log_2 n)$，所以，总的时间花费要比$O(n^2)$少很多。

3 算法检验

根据以上介绍的方法，笔者在 VC 中实现了双向 Dijkstra 二叉树算法，数据是随机生成的。实验结果如表 1 所示。

表 1 实验结果

节点数	边数	经典 Dijkstra	改进算法	效率比
2 000	41 779	72ms	11ms	6.546
2 000	6 921	75ms	10ms	7.500
2 500	64 926	121ms	13ms	9.308
2 500	10 190	124ms	12ms	10.333
3 000	94 241	167ms	15ms	11.133
3 000	14 078	182ms	14ms	13.000
4 000	163 712	308ms	19ms	16.211
4 000	23 814	321ms	17ms	18.882
5 000	255 130	473ms	29ms	16.310
5 000	35 885	480ms	17ms	28.232
6 000	49 932	721ms	21ms	34.333
8 000	86 480	1 188ms	32ms	37.125

4 结论

本文提出的双向 Dijkstra 二叉树算法原理简单，实现方便，虽然这种方法增加了其它开销，但在顶点较多的图上，效果很明显，是一种优秀的最短路径求解方法。

(下转第40页)