doi:10.3969/j.issn.1672-4348.2016.06.006

导航过程中最短路径的动态调整算法

赵忠孝[1], 冯娴[2]

(1.福州外语外贸学院 信息系, 福建 福州 350202; 2.福建工程学院 软件学院, 福建 福州 350003)

摘要: 在导航过程中,当最短路径道路上有拥挤、堵塞或中断的情况发生时,利用 Dijkstra 最短路径算法中的最短路径长度和前驱结点两个辅助向量数据,可迅速在其邻接结点中选择一条新的最短路径。实现了最短路径的动态调整,从而可以尽快地到达目的地。

关键词: 导航; 图论; 最短路径; Dijkstra

中图分类号: TP301 **文献标志码**: A **文章编号**: 1672-4348(2016)06-0543-04

A dynamic alignment algorithm for the shortest path in the navigation process

Zhao Zhongxiao, Feng Xian

(1. Information Department, Fuzhou College of Internation Studies and Trade, Fuzhou 350202, China;
2. Software College, Fujian University of Technology, Fuzhou 350003, China)

Abstract: The distance (length) of the shortest path and the data of two supplementary vectors of the previous nodes were adopted to quickly generate an alternative shortest path from adjacency nodes via Dijkstra algorithm. The dynamic alignment of the shortest path in navigation process was implemented to reach the destination in prompt moment.

Keywords: navigation; graph theory; shortest path; Dijkstra

最短路径问题是图论中的一个经典课题,在各种导航系统中有广泛的应用。最短路径算法有距离、时间和经济效益等多种判断标准,一般仅以时间花费作为判断最短路径的标准。最短路径算法分静态最短路径和动态最短路径算法。Dijkstra 等传统的最短路径算法属静态最短路径算法,其研究已经比较成熟,动态最短路径算法成为近期研究的热点。在动态最短路径算法中,有限定搜索区域的算法[1],也有限定搜索方向的 A * 算法[2-5],其基本思路是缩小搜索的范围,提高了搜索效率。这些算法都是在路径参数动态变化的情形下重新搜索最短路径,忽略了由静态算法搜索最短路径时所生成的相关数据。现有的动态最短路径的算法都是根据路径中变化权值,重新搜索最短路径,其时间花费均在 $O(N^2)$ 之上。实际上,在导航开始时已经搜索到一条最短路径,并有最短路径长度和前驱结点两个辅助向量的数据。在导航的过程中,只是在最短路径上发生了拥挤、堵塞或中断。此时,并不需要在整个路网中重新搜索,完全可以利用已有的数据,对部分路径进行一些小范围的调整,便可迅速选择一条新的最短路径。

1 相关概念

Dijkstra 最短路经算法是计算带权有向图从源点 V_0 到其他各点的最短路径,按路径长度的递增次序,逐步产生“贪心”(Greedy)的算法。为方便,本文仅以无向图为例进行讨论。实际上,交通线路等大多数是无向图。对于有向图,若某弧不存在,则可认为其对应的边不存在即可。下面先

收稿日期: 2016-06-27

第一作者简介: 赵忠孝(1948-),男,山西闻喜人,教授,研究方向:算法设计与分析、数据库设计。

介绍该最短路径算法中用到的相关概念。

1.1 定义

设有向图 $G=(V,E)$,给定结点 $v_1,v_2,\cdots,v_n \in V$, 边 $e_1,e_2,\cdots,e_m \in E$,用邻接矩阵 $\boldsymbol{C}$ 表示有向图 G,权 $\boldsymbol{C}[i][j]$定义为:

$$\boldsymbol{C}[i][j]=\begin{cases} w_{ij} & \text{若}(v_i,v_j)\text{ 是 }E\text{ 中的边,} \\ \infty & \text{若}(v_i,v_j)\text{ 不是 }E\text{ 中的边,} \end{cases}$$

其中,w_{ij}表示边(v_i,v_j)上权值,∞ 表示一个计算机允许的、大大大于所有边上权值的数。

1.2 相关的数据结构

用 C 语言定义的数据结构:

```
#define                MaxVertexNum 100
// * 最大顶点数设为 100
#define MAX 90 000.0
// * 设定最大权值为 90 000.0
typedef char VertexType; // * 顶点类型设为字符型
typedef float EdgeType;                    // * 边的权值设为实型
typedef struct
{VertexType vexs[MaxVertexNum]; // * 顶点表
EdgeType edges[MaxVertexNum][MaxVertexNum]; // * 邻接矩阵,即边表
int n,e;          // * 顶点数和边数
}MGraph;
int P[N];
float D[N];
MGraph * G。
```

(1)用邻接矩阵来表示带权无向图(图 1)。

(2)辅助向量 float $D[N]$用来存储源点到其余各结点的最短路径长度。

(3)辅助向量 float $P[N]$用来存储源点到其余各结点的最短路径上前驱结点。

图 1 有 11 个结点,图中的数值表示其对应边上的权值,其邻接矩阵如图 2 所示。

由 Dijkstra 最短路径算法,能够计算出从一个结点到其他结点的最短路径。比如,从 V_0 到其他结点的最短路径。在图 3 中,V_0 到各结点的实线路径为最短路径,虚线为原来边。这样,V_0 到 V_6 的最短路径为:$V_0\rightarrow V_2\rightarrow V_5\rightarrow V_6$。$V_0$ 到 V_{10}的最短路径为:$V_0\rightarrow V_2\rightarrow V_5\rightarrow V_7\rightarrow V_{10}$。由于是无向图,从 V_{10}到 V_0 的最短路径应该是相反的一条路径 $V_{10}\rightarrow V_7\rightarrow V_5\rightarrow V_2\rightarrow V_0$,其长度是一样的。在这条路径上,称路径上将要到达的结点为前驱结点,如 V_7 是 V_{10}的前驱结点,V_5 是 V_7 的前驱结点等。

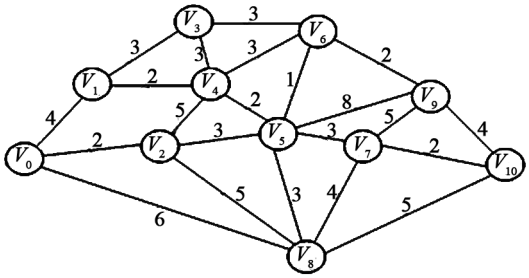

图 1 有权无向图

Fig.1 Weighted undirected graph

$$\boldsymbol{C}=\begin{bmatrix} 0 & 4 & 2 & \infty & \infty & \infty & \infty & \infty & 6 & \infty & \infty \\ 4 & 0 & \infty & 3 & 2 & \infty & \infty & \infty & \infty & \infty & \infty \\ 2 & \infty & 0 & \infty & 5 & 3 & \infty & \infty & 5 & \infty & \infty \\ \infty & 3 & \infty & 0 & 3 & \infty & 3 & \infty & \infty & \infty & \infty \\ \infty & 2 & 5 & 3 & 0 & 2 & 3 & \infty & \infty & \infty & \infty \\ \infty & \infty & 3 & \infty & 2 & 0 & 1 & 3 & 3 & 8 & \infty \\ \infty & \infty & \infty & 3 & 3 & 1 & 0 & \infty & \infty & 2 & \infty \\ \infty & \infty & \infty & \infty & \infty & 3 & \infty & 0 & 4 & 5 & 2 \\ 6 & \infty & 5 & \infty & \infty & 3 & \infty & 4 & 0 & \infty & 5 \\ \infty & \infty & \infty & \infty & \infty & 8 & 2 & 5 & \infty & 0 & 4 \\ \infty & \infty & \infty & \infty & \infty & \infty & \infty & 2 & 5 & 4 & 0 \end{bmatrix}$$

图 2 邻接矩阵

Fig.2 Adjacency matrix

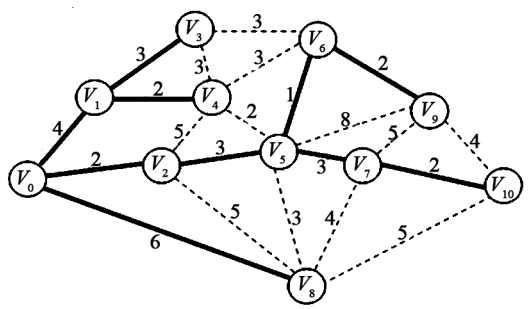

图 3 最短路径图

Fig.3 The shortest path graph

在 Dijkstra 算法结束时,各结点到 V_0 最短路径长度存储在辅助向量 $D[N]$中。具体数值如表 1 所示。

表 1　各结点到 V_0 的最短路径值

Tab.1　The shortest path value of each node to V_0

结点	0	1	2	3	4	5	6	7	8	9	10
最短路径值	0	4	2	7	6	5	6	8	6	8	10

各结点到 V_0 的最短路径上的前驱结点存储在辅助向量 $P[N]$ 中。具体数值如表 2 所示：

表 2　各结点到 V_0 的最短路径的前驱结点

Tab.2　The precursor nodes of the shortest path of each node to V_0

结点	0	1	2	3	4	5	6	7	8	9	10
前驱	0	0	0	1	1	2	5	5	0	6	7

若现在从 V_{10} 沿着最短路径向 V_0 行驶，当行使到 V_5 时发现 V_5-V_2 的道路堵塞或中断。此时，如何迅速地重新选择一条新的最短路径，就是一个迫切需要解决的问题。

2　算法思想

如果 V_5-V_2 之间的道路堵塞或中断，则其对应边上的权值就会增大。例如：$w_{25}=3$ 变成 $w'_{25}=7$，其增值为 4，用 $w=4$ 来表示。这样原来最短路径值也随之发生改变，改变后的值用 $D'[5]$ 表示：

$D'[5]=D[5]-w_{25}+w'_{25}=5-3+7=9$

从 V_5 到 V_0 的最短路径长度由 5 变成了 9。这样，原来的最短路径可能已经不是最短路径了，需要重新计算来确定一条新的最短路径。从 V_5 出发到 V_0 的最短路径一定是经过 V_5 邻接点的一条路径，V_5 邻接点有 V_2、V_4、V_6、V_7、V_8 和 V_9。这些邻接点到 V_0 都是已经有最短路径，现只需要从 V_5 经过其邻接点到 V_0 的所有可能的最短路径中选择一条最短的路径即可。也就是：

$D'[5]=\mathrm{MIN}\{(D[i]+w_{5i}) \mid V_i \in V_5$ 的邻接结点$\}$。

现可将 V_5 的邻接结点划分为两个集合，分别命名为 S_1 和 S_2。其中 S_1 的结点是其到 V_0 的最短路径不经过 V_5 的结点，如 V_2、V_4、V_8 等结点。S_2 的结点是其到 V_0 的最短路径经过 V_5 的结点，如 V_6、V_7 等结点。对于 S_1 中的结点，V_5 经过各结点的最短路径的值计算如下：

$D[2]+w_{52}=2+7=9$，

$D[4]+w_{54}=6+2=8$，

$D[8]+w_{58}=6+3=9$，

其中最小的值为 8，是经过 V_4 的一条路径的值，用 min1=8 表示。但还不能确定 min1=8 就是最短路径的值，还需要和 S_2 中结点的最短路径进行比较。由于 S_2 中结点的最短路径也走 V_5-V_2 这条边，其原最短路径的值自然也会增大，也需要重新确定一条最短路径。对于 $V_i \in S_2$，如果 $D[i]+w_{5i}$ 的值已经大于 min1，则可以排除在外。如：$D[7]+w_{57}=8+3=11$，新的最短路径值一定不会大于 min1，可以排除经过此结点为最短路径经过的结点。而对于 V_6 来说，$D[6]+w_{56}=6+1=7$，其值虽小于 min1，但其原来最短路径经过 V_5，V_6 原来的最短路径也可能已经不是最短路径了。其最短路径的值也还不能直接确定，需要用相同的方法，也就是递归算法来确定 V_6 最短路径的值。如果 V_6 邻接点中仍有结点最短路径经过当前结点，仍需要继续递归，直到没有邻接点的最短路径经过当前结点为止。

为不失一般性，设发生堵塞或中断的结点为 V_k，也就是要调整从 V_k 到 V_0 的最短路径。由于各结点最短路径的前驱结点存储在向量 $P[N]$ 中，若 $P[i]=k$，则邻接点 V_i 到 V_0 的最短路径直接经过结点 k，否则没有经过。V_k 邻接点则是邻接矩阵 k 行中对应值小于∞的那些结点。其算法形式化描述如下：

```
dynamicshort(MGraph *G,int k,float w)
// * G 为带权有向图,k 为当前结点,w 为路径的增值
{D[k]=D[k]+w ;    // * 修改当前结点到 V0 的路径长度
min1=D[k];
i=0;
while(P[i]≠k)     // * 在最短路径不过 k 点邻接点中寻找最短路径;        (c1)
  { min1=min{D[i]+G->edges[k][i] | Vi ∈ s1};  // * s1 为最短路径不过 Vk 的结点集
    p1=i ++;                                                    (c2)
  }
i=0;
while(P[i]==k)    // * 在最短路径过 k 点邻接点中寻找最短路径;        (c1)
  { if(min1>D[i]+ G->edges[k][i])
```

```
// * 如果当前结点原最短径路值小于min1,需递归寻找该最短路径。 (c2)
dynamicshort (G,i,w);
if(min1>D[i]+G->edges[k][i])
// * 判断新路径是否为最短路径,若是则修改最小值,并记下当前结点。
{ min1=D[i]+G->edges[k][i];
p1=i++;
}
}
P[k]=p1; // * 修改当前结点的前驱结点
D[k]=min1;// * 修改最短路径长度
return;
}。
```

3 算法分析

由 Dijkstra 等单源最短路径算法的结果,是生成以单源结点为根的一颗树。各结点走向树根的路径就是其最短路径。在前进方向上堵塞或中断时,仅需要从其邻接结点寻找最短路径。一般情况下,道路的邻接矩阵是一个稀疏矩阵,每个点的邻接点的数量也十分有限,会大大小于网络总的结点数。在邻接点中,S_1 中的结点数 N_1 一般应大于 S_2 的结点数 N_2。对 S_1 中的结点,只需要 N_1 检索。对于 N_2 中的结点,若 min1 是已经找到的最短路径的值,对于原最短路径大于该值的,不需要进一步再判定。只需要将那些原来的最短路径值小于 min1 的结点递归调用本算法。且每递归一次,其最短路径的值都在增加,若其值超过 min1 则终止。因最短路径是一个树型结构,递归调用到叶结点将会终止。因此,递归调用的层次也是非常有限的。本文对有 20 个结点的网络中的每个结点进行了最短路径调整的测试。每次测试时,将结点与其前驱结点的边值增加一个随机值。使其原来的最短路径发生了改变,统计了各结点搜索新的最短路径其循环次数 $c1$(在算法已有标示)和最短路径值的修改次数 $c2$(在算法已有标示)。各结点搜索次数的具体结果见表 3。

表 3 各结点搜索次数统计

Tab.3 Each node search statistics

结点	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	平均
$c1$	80	40	40	100	40	40	40	40	40	40	120	40	40	40	40	40	40	40	40	49.5
$c2$	5	0	3	13	2	2	1	2	0	0	13	2	3	1	0	0	2	0	1	2.6

对这 20 个结点的网络利用本算法,使改变的路径调整到最短路径只需要平均 49.5 次循环,对最短路径的值进行了 2.6 次修改。其总的时间复杂度一般不会超过 $O(n)$。若用 Dijkstra 最短路径算法进行搜索,则需要 760 次循环,最短路径的值也要进行 76 次修改运算。

4 结束语

在文献[2]中,证明了动态网络的最短路径是一个 NP 完全问题,将动态问题化为若干个时间段,分段是稳定的。每个稳定区间都要解不等式组,迭代次数为 k ,k 为子区间数。依次将每一个稳定区间的解连接起来得到整体的解。其初始要调用 Dijkstra 算法,算法复杂度为 $O(n^2)$,各子区间算法的时间复杂度为 $O(mK)$, $K=\max(n,k)$。文献[3]中算法的时间复杂度比 Dijkstra 算法仅快 1~2 倍。其他文献所给出算法的复杂度都在 $O(n^2)$附近,本算法的时间复杂度远远低于已有的算法。在遇到堵塞或中断事件后,可利用本算法迅速找到另一条到达目的地的最短路径。

参考文献:

[1] 汪晓洁,汤建国,李娟,等.基于可变权值的态最短路径算法[J].新疆大学学报(自然科学版),2015,32(3):342-346.
[2] 林澜,闫春纲,蒋昌俊,等.动态网络最短路问题的复杂性与近似算法[J].计算机学报,2007,30(4):608-614.
[3] 邹亮,徐建闽,朱铃湘.A_算法改进及其在动态最短路径问题中的应用[J].深圳大学学报(理工版),2007,24(1):32-35.

(下转第 571 页)

4　总结

本文提出采用最小偏差直线插补算法实现转子绕线机两轴联动控制。并通过仿真对比两种插补算法，提出的方法具有较小的绕线误差；将最小偏差直线插补算法运用在 RW400 大型绕线机上进行实际绕线测试，结果表明所提出算法的绕线机绕制的转子排线间距均匀，排列紧密，单圈误差小于 0.1%。最小偏差直线插补算法能较好地应用于大型电机转子绕线机上，具有较高的绕线精度。

参考文献：

[1] 王艳新，赵春锋.基于 PLC 的双飞叉绕线机控制系统设计[J].组合机床与自动化加工技术，2009(7)：72-74.

[2] 惠晶.绕线机步进驱动自动控制系统[J].电气传动，2005，35(8)：56-59.

[3] Shi Yaoyao，Tang Hong，Yu Qiang. Development of NC tape winding machine[M]. London：Springer，2008：753-762.

[4] 刘洪玮，郭吉丰，孙云云，等. 两轴伺服控制的绕线机系统[J].电气传动，2010，40(9)：9-12.

[5] 郭永环，范希营，刘凤国.一种新型的数字积分直线插补算法的研究[J].制造技术与机床，2012(4)：164-167.

[6] 李腾飞，凌有铸，刘敬猛.基于最小偏差法插补技术的 FPGA 设计与实现[J].重庆理工大学学报(自然科学版)，2013，27(5)：90-93.

[7] 伍晓亮，田怀文，江晓亮，等.精密电阻绕线机送丝机构的设计与改进[J].现代制造工程，2016(1)：105-109.

[8] Kiyoshi O，Yasuaki O，Hideo D.A speed control method for a pm motor using a speed observer and a low-resolution encoder [J].Electrical Engineering in Japan，2003，143(143)：66-75.

[9] 惠晶，王伟.两轴同步协调控制的绕线机系统[J].机床与液压，2013(13)：71-74.

[10] 叶春林，刘佳. NUM 数控系统参数的调整与应用[J]. 设备管理与维修，2015(S2)：288-291.

[11] 陈劲松. NUM 数控系统的使用与维修(第 7 讲 NUM 数控系统的软件及通信)[J]. 机械工人(冷加工)，2003(7)：77-78.

(特约编辑：黄家瑜)

(上接第 546 页)

[4] 周琳，陈发钢.车载导航系统中动态最短路径研究[J].牡丹江师范学院学报(自然科学版)，2013(3)：23-25.

[5] 章昭辉.一种基于离散变权网络的动态最短路径快速算法[J].计算机科学，2010，37(4)：238-240.

[6] 张一珂，刘鸿剑，朱志斌，等.基于车辆导航的一种改良动态最短路径算法[J].科技广场，2009(5)：26-28.

[7] 任子辉，王坚.紧急事件的动态交通流模型及双向动态最短路诱导算法[J].计算机应用，2008，28(11)：2955-2960.

[8] 陈晓红，王艳娟.改进的 Dijkstra 算法在动态路径引导中的实现[J].科学技术与工程，2008，8(22)：6024-6027.

[9] 田鹏飞，王剑英.动态最短路径算法及其仿真 [J].计算机仿真，2007，24(6)：153-155.

[10] Huang A B，Wu B Q，Zhan F B. A shortest path algorithm with novel heuristics for dynamic transportation networks[J]. Intenational Journal of Geographical Information Science，2007(6)：625-644.

[11] Chabini I，Lan S. Adaptations of the A * algorithm for the computation of fastest in deterministic discrete time dynamic networks[J]. IEEE Transactions on Intelligent Transportation Systems，2002，3(1)：60-74.

[12] SharmaY，Saini S C，Bhandhari M. Comparison of dijkstra shortest path algorithm with genetic algorithm for static and dynamic routing network[J]. International Journal of Electronics and Computer Science Engineering，2012，1(2)：516-525.

(特约编辑：黄家瑜)