文章编号:1671-9352(2006)04-0040-04

新的 k 最短路算法

李成江

(山东科技大学 信息与电气工程学院,山东 青岛 266510)

摘要:在无向图上,对于任意源点—目的点点对,给出了一个新的 k 最短路算法.这一算法按长度递增给出 k 最短路路径.算法的复杂度为 $O(m+n\lg n+m\lg k)$.这一算法基于动态规划,首先计算出每一点到源点的最短距离,然后从目的点回溯到源点.根据各点的最短距离信息,给出一棵以目的点为根节点,源点为叶子的树表示的 k 最短路路径.

关键词:动态规划;最短路径;无向图

中图分类号:TP319 **文献标识码**:A

A new algorithm to find the k shortest paths

LI Cheng-jiang

(College of Information & Engineering, Shandong University of Science and Technology, Qingdao 266510, Shandong, China)

Abstract: A new algorithm is described to enumerate the k shortest paths connecting a given pair of vertices in a undirected graph. Our algorithm outputs the paths in order by length in total time $O(m+n\log n+m\log k)$. The algorithm is bases on dynamic programming. Firstly compute the shortest distances for every vertex to the source, later trace to the source from the destination on the ground of the distance data and list the k shortest paths.

Key words: dynamic programming; shortest path; undirected graph

0 引言

最短路问题是一个著名问题,它在实际生产生活中有广泛应用.在许多情况下,我们不仅仅要考虑最短路也要考虑次短路、次次短路,即 k 最短路问题[1].k 最短路算法涉及很多领域,如在机器人路径选择、交通工程、通信等方面有实际意义.k 最短路径问题是指在图上,对于给定的源点—目的点,列出路径长度从最短路径到第 k 短的路径.人们从不同的角度研究了它的很多变形.大多数算法都是针对有向图简单路径的[2].David Eppstein 给出了一个算法,对于有向图上的路径允许环的存在[3].John Hershberger 等人在路径替换的基础上给出了一个新的简单路径的有效算法[4].Zuwairie Ibrahim 等人从 DNA 计算角度考虑这一问题[5].W. Matthew Carlyle 等人给出接近最短路径的简单路径的 k 最短路算法[6].Gang Liu 等人给出了一个满足多个限制的算法[7].M. H. Macgregor 考虑了在通信网络中 k 最短路径问题[8].

k 最短路径问题一般指的是有向图上的路径.这是因为对于实际问题在有向图上可能有比无向图上更有效的算法存在,或者实际问题中两点间的边有方向问题.本文处理的是无向图.有许多问题,如交通领域,两点间的边大多数是没有方向约束的.它们直接的数学模型就是无向图.这样处理更有普遍意义.本文利用动态规划思想,先计算出各个点到源点的距离,利用这些数据中包含的信息,从目的点反向逆推.考虑各个路径对最短路路径长度的增加量,依次给出所找的 k 个路径.最短路路径的长度增量当然是零.

收稿日期:2006-03-08

作者简介:李成江(1972-),男,博士研究生,主要研究方向:控制理论与控制工程.

如果算法能够不重复、不遗漏地有序地给出路径的长度增量中最小的 k 个,那么也就相当于给出 k 最短路路径.

本文给出的算法思想简洁,结构简单,适应范围较广.

1　预备知识

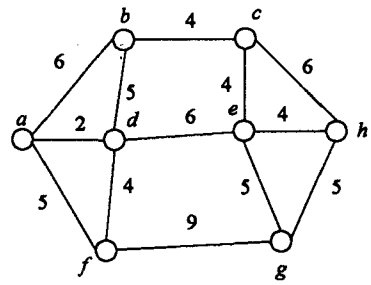

图 1

我们假设输入的图为 $G(V,E)$,如图 1.它有 n 个点,$v_i \in V$,m 条边 $e_i \in E$. 图 $G(V,E)$是一个无向图. 每条边的长度 $l(e)$假定是非负的. 边长有时也称为权重. 这代表的就不一定是路程,可能是某种费用. 在不引起歧义的情况下不加以区别. 我们也可以把这一定义延伸来指任何一条图上路径的长度,例如 $l(p)$. 距离,$d(s,t)$,是指从源点 s 开始到目的点 t 的最短路径的长度. 由于是无向图,$d(s,t)$和 $d(t,s)$是相等的. 我们利用一个图的顶点序列来表示一个路径. 当然序列中相邻的两点间有共同的边. 我们这里的路径是允许包括环的. 这与文献[2,3,6]不同. 最短路径总是不包括环的,即是简单的.

长度增量 IN(p)意味着 $l(p)-l(p_0)$,p_0 是源点—目的点之间最短路径,p 是从目的点回溯到源点的一条回溯路径,不一定是最短路径. 回溯路径 p 有两部分组成,$t,\cdots v_i$ 和 $v_i,\cdots s$. 前一部分是实际从目的点到点 v_i 的回溯. 后一部分是从点 v_i 到源点的最短路径. 我们实际上只回溯到点 v_i,后一部分只是想象沿最短路径回溯,实际上没走. 我们可以使用 IN(v_i)表示 IN(p). 我们可以通过比较长度增量来列出各个路径的长度顺序.

2　路径分叉

计算各点到源点的最短距离是容易的. 本算法的关键是如何处理在回溯过程中的路径分叉现象. 一般来说,一个顶点是连接多个边的. 因此回溯中的分叉现象是无法避免的. 在图 2 中,我们假定已经计算出每一点到源点的最短距离(图上已标出). 源点为 a,目的点为 e. 从 e 向 a 回溯就面对 c、d、g、h 4 个相邻的点,面对下一步向哪一个点回溯的问题,即路径分叉. 在处理分叉时,我们要处理 3 种情况. 具体见下面定理.

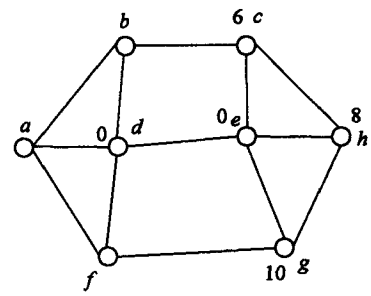

图 2

定理 1　假设已知图中任何一点到源点的距离是已知的,即 $d(s,v_i)$,$v_i \in V$. 回溯路径 $p,t,\cdots v_i,\cdots s$ 回溯到点 v_i. 点 v_j 是点 v_i 的相邻顶点中的一个. 边 e_{ij}是连接点 v_j 和点 v_i 的边.

1)如果 $d(s,v_i)=d(s,v_j)+l(e_{ij})$,那么 IN($v_j$) = IN($v_i$);

2)如果 $d(s,v_i)<d(s,v_j)+l(e_{ij})$,那么 IN($v_j$) = IN($v_i$) + $d(s,v_j)+l(e_{ij})-d(s,v_i)$;

3)如果 $d(s,v_j)=d(s,v_i)+l(e_{ij})$,那么 IN($v_j$) = IN($v_i$) + $2l(e_{ij})$.

证明　我们从点 v_i 回溯到点 v_j.

如果 $d(s,v_i)>d(s,v_j)+l(e_{ij})$,那么我们从点 v_i 到点 v_j,再从 v_j 到源点的最短路径回到源点得距离小于 $d(s,v_i)$.而这是与 $d(s,v_i)$定义相矛盾的.

1)如果 $d(s,v_i)=d(s,v_j)+l(e_{ij})$,那么点 v_j 是在从 s 到 v_i 的某一条最短路径上. 如果不是,那么根据最短路径的定义应该有 $d(s,v_i)<d(s,v_j)+l(e_{ij})$,二者是矛盾的. 因此回溯路径 $t,\cdots v_i$ 和从点 v_i 到源点的最短路径与回溯路径 $t,\cdots v_i,v_j$ 和点 v_i 到源点的最短路径是同一条路径. 因此 IN(v_j)等于 IN(v_i).

2)如果 $d(s,v_i)<d(s,v_j)+l(e_{ij})$那么点 v_j 不在从 s 到点 v_i 的最短路径上. 否则根据定义应该有 $d(s,v_i)=d(s,v_j)+l(e_{ij})$. 因此我们从点 v_i 回溯到点 v_j 实际上是增加了回溯路径的长度. 增加的量是 $d(s,v_j)+l(e_{ij})-d(s,v_i)$. 因此 IN($v_j$) = IN($v_i$) + $d(s,v_j)+l(e_{ij})-d(s,v_i)$.

3)如果 $d(s,v_j)=d(s,v_i)+l(e_{ij})$,那么点 v_i 是在从 s 到 v_j 的最短路上(这就像 $d(s,v_i)<d(s,v_j)+$

$l(e_{ij})$的情况). 这也意味着 $d(s,v_i)<d(s,v_j)+l(e_{ij})$. 根据2)有

$$\mathrm{IN}(v_j)=\mathrm{IN}(v_i)+d(s,v_j)+l(e_{ij})-d(s,v_i),$$

$$\mathrm{IN}(v_j)=\mathrm{IN}(v_i)+d(s,v_i)+l(e_{ij})+l(e_{ij})-d(s,v_i)=\mathrm{IN}(v_i)+2l(e_{ij}).$$

对于图1给出的无向图,我们从 e 向 a 回溯时,面对4个相邻点,计算它们的长度增量,如图3.

这一定理处理当回溯到某一交叉点时,如何计算交叉点相邻节点的路程长度增量. 我们知道了每一条路径相对最短路径的长度增量,也就知道了每一条路径的长度,这样就可以根据路径的长度增量排列它们的顺序.

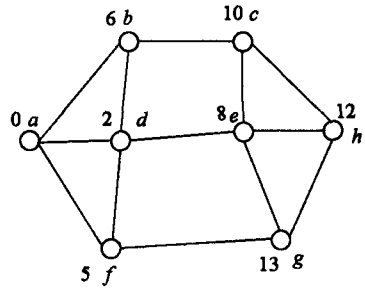

图3

3 计算 k 最短路径算法

在本文中,对于非负权重的无向图,我们给出了一个新的计算 k 最短路径的算法. 算法首先计算每一顶点到源点的最短距离. 然后再根据这些数据,从目的点回溯到源点. 在这个过程中得到用一棵树表示的 k 最短路径. 这棵称为输出树的结构根就是目的点,叶子都是源点. 这样任意一条从叶子到根的路径都在 k 最短路径之内. 叶子的长度增量表示了一条 k 最短路径的长度增量.

计算 $d(s,v_i),v_i\in V$ 有很多方法. 我们采用了动态规划的一个变形,不同于 Floyd 的方法. 我们增加了一个递增的有序序列 sq,用于保存计算各个点最短距离的中间数据. 它的每个记录含有3个域,一个记录点,一个记录点的最短距离或长度增量,一个记录回溯路径当前节点的前驱在输出树中的位置. 它根据第二个域值排序,保持递增顺序. 这降低了复杂度,从 Floyd 方法的 $O(n^3)$降到了 $O(m\lg n)$. 第三个域中数据用于构造输出树输出 k 最短路径. 它在后面的回溯中还要使用,在计算最短距离中不用. 在回溯中这一序列只需保持 k 个记录,多的可以删去. 因为我们只考虑 k 最短路径,这一序列可以用链表、数组等方式实现.

算法第一部分 计算 $d(s,v_i),v_i\in V$.

1) 初始化空 sq,并将源点 s 放入序列中. 每一点最短距离初始化为无穷大;

2) 重复下面步骤直到 sq 变空:

(1) 从 sq 取出第一个点 v_i(序列中它的最短距离最小);

(2) 计算点 v_i 的所有相邻点的 v_j. 如果 $d(s,v_j)>d(s,v_i)+l(e_{ij})$,那么 $d(s,v_j)=d(s,v_i)+l(e_{ij})$.

(3) 加这些最短距离被修改了的相邻点到序列 sq 中. 根据 $d(s,v_j)$大小,保持序列递增有序. 图2显示了计算出的每一点到源点的最短路径距离.

在计算出各点的最短距离后,就可以从目的点 t 回溯. 回溯也使用了序列 sq 保存中间数据. 这里的数据域要保存点、长度增量和前驱在输出树中的位置. 在每次取出点时,将它输出,插入到输出树的相应位置,并计算相邻节点的长度增量 $\mathrm{IN}(v_i)$.

算法第二部分 回溯.

1) 初始化空 sq,并将目的点 t 放入序列中;

2) 重复下面步骤,直到输出源点 k 次:

(1) 取出序列第一个点 v_i(序列中它的长度增量最小),根据其前驱在输出树中的位置加入到输出树中,成为其前驱的孩子节点(如果没有前驱就作为输出树的根节点);

(2) 根据定理1计算 v_j 的所有相邻点的长度增量.

(3) 将这些点、长度增量和前驱 v_i 在输出树中的位置加入到 sq 并根据长度增量保持有序.

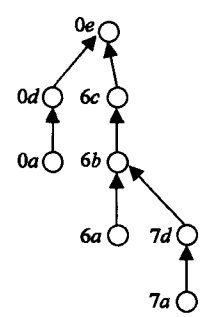

图4

图4显示了图1给出的无向图的3最短路径输出的树. 3条路径是:a,d,e;a,b,c,e 和 a,d,b,c,e. 它们的长度增量分别是:0,6,7.

一般的 k 最短路径是简单路径,不包括环. 但是也有文章考虑了环的存在. 本文对于无向图,图上环一般是不能避免的. 在 k 最短路径上可以有环,也可以根据具体情况不考虑环. 如在交通上,如果认为驾驶者是理性的,就无需考虑 k 最短路径上的环.

4 算法复杂度

本算法中图的存储结构可以采用矩阵、十字链表等等. 附加的序列可以采用链表、数组等. 它的最大长度是 n. 但一般是达不到这一最坏情况的. 因为从序列中取出一个点后才处理它的相邻点. 算法第一部分(3)也消除了一个点被反复送入序列的可能.

对于输出树,如果不考虑 k 最短路径中含有环,那么数的节点数也要小于 m. 但是如果图中含有环,也允许 k 最短路径中含有环,那么由于环的存在,会计算出任意大的 k 最短路径. 输出树的节点数就难以确定.

由于序列保持有序,那么在算法第一部分的一次插入完成比较的复杂度就是 $O(\lg n)$,所有插入的复杂度 $O(n\lg n)$. 在算法的第二部分的插入,由于序列最长为 k,那么一次插入的复杂度为 $O(\lg k)$.

在算法第一部分(2)的比较,每个点仅进入序列一次,那么沿着每条边进行两次比较,总次数为 $2m$,复杂度 $O(m)$. 算法第二部分(2)中进行的比较,和环有关. 如果图中无环或不考虑 k 最短路径中含有环,那么比较次数明显小于 $2m$,复杂度也是 $O(m\lg k)$. 但是如果图中含有环,也允许 k 最短路径中含有环,那么比较次数就无法确定. 因此在图中无环或不考虑 k 最短路径中含有环的情况下,算法复杂度为 $O(m + n\lg n + m\lg k)$.

5 算法的正确性

在回溯过程中,回溯到点 v_i,那么它和它的相邻点 v_j(即将回溯到的)的关系,根据定理 1 是 $\mathrm{IN}(v_j)$ 大于等于 $\mathrm{IN}(v_i)$. 因此如果在回溯中被排除在序列 sq 之外的回溯路径一定在 k 最短路径之外. 也就是说,回溯到源点时序列 sq 中的点代表的路径就是 k 最短路径.

6 结语

本文给出了一个新的计算无向图中 k 最短路径的算法. 算法复杂度 $O(m + n\lg n + m\lg k)$. 本算法在山东交通信息中心的山东高速公路调查分析过程及分析软件中使用效果良好. 对于山东高速公路网络,计算所有两站点间长度小于等于最短路径 130% 的路径,使用本算法在 CPU500M 的机器上所用时间小于 20 min.

参考文献:

[1] Myrna Palmgren, Di Yuan. A short summary on K shortest path: Algorithms and applications[EB/OL]. http://www.esc.auckland.ac.nz/Mason/Courses/LinkopingColGen99/kth.pdf 1999, 2006-01-08.

[2] Eppstein D. Finding the k shortest paths[J]. SIAM Journal on Computing, 1999, 28(2): 652 ~ 673.

[3] John Hershberger, Matthew Maxely, Subhash Suriz. Finding the K shortest simple paths: A new algorithm and its implementation[R]. ALENEX Baltimore, 2003.

[4] Zuwairie Ibrahim, Yusei Tsuboi, Mohd Saufee Muhammad, et al. DNA implementation of k-shortest paths computation[A]. 2005 IEEE Congress on Evolutionary Computation[C]. UK: Edinburgh, 2005, 1:707 ~ 713

[5] QIUJIN WU, JOANNA HARTLEY. Using K-shortest paths algorithms to accommodate user preferences in the optimization of public transport travel[A]. ASCE. The 8th International Conference on Applications of Advanced Technologies in Transportation Engineering[C]. U.S: ASCE, 2004. 181 ~ 186.

[6] W Matthew Carlyle, R Kevin Wood. Near-shortest and K-shortest simple paths[J]. Networks, 2005, 46(2): 98 ~ 109.

[7] G Liu, K G Ramakrishnan. A * Prune: An algorithm for finding k shortest paths subject to multiple constraints[C]. Proceedings of the INFO-COM 2001 Conference, IEEE, Anchorage, Alaska, 2001. 743 ~ 749.

[8] Macgegor M H, Grover WD. Optimized k-shortest-paths algorithm for facility restoration[J]. Software Practice and Experience, 1994, 24(9):823 ~ 828.

(编辑:李晓红)