经过指定的中间节点集的最短路径算法

黄书力,胡大裟,蒋玉明
HUANG Shuli, HU Dasha, JIANG Yuming

四川大学 计算机(软件)学院,成都 610065
College of Computer Science, Sichuan University, Chengdu 610065, China

HUANG Shuli, HU Dasha, JIANG Yuming. Algorithm for finding shortest path which must go through specified intermediate node set. Computer Engineering and Applications, 2015, 51(11):41-46.

Abstract: The vast majority of researches about the shortest path algorithm, nowadays, focus just on the case starting from the beginning point and ending at the ending point. If additional condition that the shortest path must go through some given nodes of which number is uncertain must be met, then most of the existing classic algorithms are not applicable. A general method based on the classical Dijkstra algorithm and greedy algorithm is presented to solve this kind of problem. The main method is to split the relevant node set into three sub sets, find the local shortest path of connecting the three subset separately to form the global shortest path to be selected, obtain the target path through screening. The time complexity of the algorithm is given by theoretical analysis and the effectiveness of the algorithm is verified by programming calculation.

Key words: Dijkstra algorithm; greedy algorithm; dynamic planning; shortest path; pruning algorithm

摘 要: 目前研究最短路径的算法,多数只是针对从起点出发到达终点的情况。如果限制这条最短路径必须要经过某些指定的中间节点,则现有的一些算法就不再适用了。基于Dijkstra算法和贪心理论,给出了解决此类问题的方法。将相关节点集拆分成三个子集,分别求连通三个子集的局部最短路径,进而形成全局待选最短路径,通过筛选得到目标路径。通过理论分析算法的时间复杂度和实际编程实验确认了该算法的有效性。

关键词: Dijkstra算法;贪心算法;动态规划;最短路径;相关节点
文献标志码: A **中图分类号:** TP391 doi:10.3778/j.issn.1002-8331.1311-0291

1 引言

单源点最短路径算法是图论中的一个重要算法,可以用来解决道路设计和网络选路等诸多动态规划和优化问题。Dijkstra EW.A于1959年提出了著名的Dijkstra算法[1],这是一个经典的单源点最短路径算法,用于计算一个节点到其他所有节点的最短路径。主要思想是以起始点为中心向外层层扩展,直到扩展到终点为止。

目前以Dijkstra算法为基础,针对最短路径问题的研究非常多。通过研究前人给出的算法的复杂性,提出一种新的时空复杂度更低的改进算法[2-6];基于某种特殊环境和条件,给出一种可行的最短路径算法[7-12];基于第一条最短路径算法,扩展到研究前 N 条最短路径算法[13-14]。这些研究或者基于特定领域,给出特定环境下的最短路径算法,或者是针对Dijkstra算法本身,提出时空效率更高的改进算法,给研究最短路径问题提供了非常广阔的思路。然而,在图论中还存在着这样一类未被广泛研究却又有着重要实际研究意义的问题:

(1)“邮递员问题”。邮递员从邮局出发送完信件后回家(或回邮局),需要为他安排行驶路径使得总的行驶距离最短。

(2)“旅行家问题”。给旅行家设计一条旅行线路,使得他从某地出发,游玩一些事先计划的旅游景点后,到达另一目的地的总行驶距离最短。

(3)公交车路线设计问题。给公交车设计一条线

基金项目: 四川省科技攻关计划项目(No.2012GZ0090)。
作者简介: 黄书力,硕士,研究方向:数据库与信息系统、软件工程;胡大裟,博士,讲师,研究方向:软件工程、并行与分布式系统;蒋玉明,博士,教授,研究方向:数据库与信息系统、软件工程。
收稿日期: 2013-11-20 **修回日期:** 2014-01-09 **文章编号:** 1002-8331(2015)11-0041-06
CNKI网络优先出版: 2014-04-09, http://www.cnki.net/kcms/doi/10.3778/j.issn.1002-8331.1311-0291.html

路,使得公交车从起始站出发途经一些重要站点后达到终点站的行驶距离最短。例如,2008年北京奥运会期间所使用的34条经过北京市主要奥运场馆的奥运公交专线的设计[15]。这些奥运公交专线在设计上具有一个共同的特点,就是从公交调度起点站出发后途径一些要求经过的奥运场馆站后(这里忽略公交车经过奥运场馆站点的顺序)到达公交终点调度站。

此外,还有诸如要求经过某些特定的中间路由器的特殊的网络选路问题等。这类问题均可以归纳为:

在一个图中,需要计算出从指定的顶点出发,经过一些指定的中间节点(这些需要经过的中间节点的个数不确定),达到指定的终点的最短距离和经过的路径。

对于此类问题,直接使用Dijkstra算法就无法完成了。为此,本文以Dijkstra算法为基础,基于贪心理论[16-17],以无向无权图为例提出了一个解决此类问题的方法,进行了详细阐述,并给出了算法关键部分的伪代码描述。有向图和带权图的算法完全类似,在本文结尾均做了简述。

2 问题描述

如图1所示,假设起点 V_{begin} 为 V_1,用数字1表示;需要通过的中间节点为 V_3,V_4,分别用数字3,4表示;终点 V_{end} 为 V_2,用数字2表示。

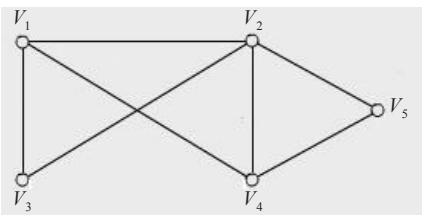

图1 示例图

如果需要求出从节点1到节点2之间的最短路径,用Dijkstra算法可以求出路径为1->2,但是如果要求必须经过中间节点3,则路径应为1->3->2。

2.1 概念定义

相关节点:起点,终点,目标路径必须要经过的指定的中间节点均为相关节点。

自由节点:网络图中,除了相关节点之外的其他节点。

全局备选最短路径:由各段局部最短路径所依次连接起来得到的全局最短路径的一个备选方案。

全局最短路径:全局备选最短路径集合中距离最短的路径。

2.2 问题定义

在一个无向无权图(记为 $G_{old}(V,E)$)中,给定一个任意的起点(记为 V_{begin}),一些指定的中间节点(将这些指定的中间节点的集合记为 S_{mid}),和一个任意终点(记为 V_{end})。要求寻找一条从起点 V_{begin} 出发,经过所有指定的中间节点,到达终点 V_{end} 的距离最短的路径。除此之外,这条最短路径没有任意额外条件。也就是说,这条路径可能会经过网络中的除了起点、终点和指定的中间节点之外的其他节点(即自由节点),同时,这条路径中某些节点可能会出现多次,路径中也可能存在回路。

3 经过指定的中间节点集的最短路径算法

贪心算法,是一种通过分级处理某些最优解问题的方法。贪心算法,在求解过程的每一步中都采取在当前状态下看来是最好或最优的选择,从而希望最终的结果是最好或最优的。如果一个复杂的问题能够分解成几个子问题来解决,并且子问题的最优解能递推到最终问题的最优解,简言之,如果局部最优解能最终推导出全局最优解,那么贪心算法在解决这类问题时将非常有效。

贪心算法求解问题的基本步骤:

(1)把原问题分解成若干个易于求解的简化的子问题。

(2)对每一子问题分别求解,得到所有子问题的局部最优解。

(3)通过对全部子问题的某个局部最优解进行分析、聚合等处理,生成原问题的一个解。

(4)从上一步所求出的原问题的解集或某个初始解出发,通过筛选、迭代等手段求出原问题的解(全局最优解)。

Dijkstra算法的基本思路是先将与起点有边直接相连的节点到起点的距离记为对应的边的权重值,将与起点无边直接相连的节点到起点的距离记为无穷大。然后以起点为中心向外层层扩展,计算所有节点到起点的最短距离。每次新扩展到一个距离最短的点后,更新与它有边直接相邻的节点到起点的最短距离。当所有点都扩展进来后,所有节点到起点的最短距离将不会再被改变,因而保证了算法的正确性[1]。

Dijkstra算法求解最短路径问题的基本步骤如下:

(1)设立 Y 和 N 两个集合,Y 用于保存所有等待访问的节点,N 记录所有已经访问过的节点。

(2)访问网络节点中距离起始节点最近且没有被访问过的节点,把这个节点放入 Y 中等待访问。

(3)从 Y 中找出距离起点最近的节点,放入 N 中,更新与这个节点有边直接相连的相邻节点到起始节点的最短距离,同时把这些相邻节点加入 Y 中。

(4)重复步骤(2)和(3),直到 Y 集合为空,N 集合为网络中所有节点为止。

Dijkstra算法的基本思想和求解步骤决定了Dijkstra算法只能解决最基本的在起点和终点之间求最短路径的问题,无法解决添加了其他限制条件的,如本文中所探讨的这类要求经过指定中间节点集的最短路径问题。

本文基于贪心算法,将原问题分解成几个易于用

Dijkstra算法求解的子问题，先对各个子问题逐一求局部最优解，再在此基础上求全局最优解。具体说来，就是先将预处理后的网络拓扑图中的所有节点分为三个子集合，分别为包含起点的起点集，包含全部的需要经过的中间节点的中间节点集，包含终点的终点集。通过Dijkstra算法依次求起点集到中间节点集之间的局部最短路径，连通中间节点集中所有节点的局部最短路径，中间节点集到终点集之间的局部最短路径，以此求出一条从起点出发经过指定的所有中间节点后到达终点的待选全局最短路径。通过对有限的待选全局最短路径进行筛选，选出其中距离最短的路径，即为满足要求的全局最短路径。

3.1　预处理

3.1.1　点和边的数据提取

从文件中读取节点和边的数据并以邻接矩阵形式存储，同时统计出节点个数和边的条数。

如图2，每一行的两个数表示着两个节点之间有边联通。如果是带权图，则在后面再加一列，显示权重即可。

1，2
1，3
2，4
3，4

图2　网络节点数据的存储格式示例

如图3，邻接矩阵中的元素只有1和max两种，1表示直接有边相连，max表示无边直接相连。max为一个事先定义的足够大的数字。

$$\begin{bmatrix} 0 & 1 & \max & 1 \\ 1 & 0 & 1 & \max \\ \max & 1 & 0 & 1 \\ 1 & \max & 1 & 0 \end{bmatrix}$$

图3　示例图的邻接矩阵表示

3.1.2　剪枝

剪枝，就是从原网络图中删去目标路径肯定不会经过的节点。具体方法为，从起点V_{begin}出发，深度优先遍历该图，那些没有遍历到的节点肯定不会出现在目标路径上，就可以将它们从原网络图中“剪”掉。从而生成新的网络图$G(V, E)$，替换原来的网络图$G_{\text{old}}(V, E)$。

剪枝的目的是在不影响求目标路径的情况下，通过减少网络中节点个数，来大幅提高算法的时空效率。

下面给出从任意起点出发，经过指定的n个中间节点($n \geqslant 0$)，到达指定的任意终点的最短路径算法。

3.2　算法描述

```
Main Method:
1. //判断中间节点之间的连通情况
2. if (!isConnected(V1,V2,V3,…,Vn))
3.       return;
   //path为符合条件全程最短距离
4. path=+∞;
   //将中间节点集形成全排列
5. V1,V2,…,Vn->FullArray(V1,V2,…,Vn);
6. for each Sequence(V1V2…Vn) in FullArray
       //PathMidNode为求中间节点间距离和路径的方法
7.     midPath=PathMidNode();
8.     temp=Dijkstra(Vbegin,V1)+midPath+Dijkstra(Vn,
Vend);
9.     if (temp<path)
10.          path=temp;
```

(1)假设需要经过的指定中间节点个数为n，先判断这n个节点之间的连通情况，若有任意俩节点不连通，则满足条件的路径不存在；反之，进入下一步。

判断是否连通的方法：在这n个节点中，任选一个为根节点，深度优先遍历该图，若其他的$n-1$个所有节点都遍历到了，则是连通的；反之，则为未连通的。

(2)对这n个中间节点做全排列，生成一个中间节点序列。全排列的起点记为V_1，终点记为V_n。计算V_1和V_n之间的最短距离和路径。具体方法，见“求中间节点之间的距离和路径的方法”。

(3)求目标源点到V_1的局部最短路径，即为单源点最短路径，直接使用迪杰斯特拉算法即可。若目标源点和V_1之间的路径经过了自由节点，则将经过的自由节点按序保存到该局部路径中。

(4)求V_1到V_n的经过所有中间节点的局部最短路径，(2)中已求出。

(5)求V_n到目标终点的局部最短路径，即为单源点最短路径，直接使用迪杰斯特拉算法即可。若V_n和目标终点之间的路径经过了自由节点，则将经过的自由节点按序保存到该局部路径中。

(6)将以上3条路径依序连接起来即得经过S_{mid}中所有节点的(待选)全程最短路径。将以上3条路径的距离相加即得该条待选全程最短路径的距离。

(7)搜索(6)中求出的路径，其中距离最小的即为待求的全程最短距离，对应的路径就是待求的最短路径。

求中间节点之间的距离和路径的方法：

```
PathMidNode:
1. for V1V2…Vn-1Vn
2. //dist用于统计V0,Vn之间的最短距离
3. int dist=0;
4. for each i(1<=i<n)
5.     if (Edge(Vi,Vi+1)!=+∞)
6.         dist(Vi,Vi+1)=1;
7.     else
8.         dist(Vi,Vi+1)=Dijkstra(Vi,Vi+1);
9.     dist+=dist(Vi,Vi+1);
```

对于一个中间节点序列，若任意俩相邻节点有边直接相连，则它们之间的距离为1(在带权图中，该距离为对应边的权重值)；反之，使用Dijkstra算法求出它们之

间的最短路径和距离。若它们之间的路径经过了自由节点,则应将自由节点保存到该路径中。将所有相邻节点间的距离相加即得 V_1 和 V_n 之间的最短距离;将所有相邻节点间的路径(中间可能会经过一些自由节点)相连即得 V_1 和 V_n 之间的最短路径。

算法整体流程如图4。其中,算法描述部分及流程图中, Dijkstra($V1$, $V2$) 为利用 Dijkstra 算法求解从源点 $V1$ 到 $V2$ 的最短路径的方法,见文献[1]。相关的伪代码及各种语言编写的程序目前都有现成的,在此不作缀述。

3.3　算法演示

下面以图1所示的拓扑结构图为例,演示一次该算法。

步骤 1　对于中间节点集,以3为起点,深度优先遍历该图,能遍历到顶点4,该图是连通的。

步骤 2　利用中间节点集中的所有元素,组成全排列,形成中间节点序列集:3->4,4->3。

步骤 3　利用“求中间节点之间的距离和路径的方法”,对于步骤2所得的每一个序列,求出连通中间节点集的局部最短距离,及对应的局部最短路径:

路径1:3->1->4,距离2(距离相路径同的路径其实还有3->2->4,但是算法只求出一条最短路径)。

路径2:4->1->3,距离2(距离相同的路径其实还有4->2->3,但是算法只求出一条最短路径)。

分别用包含了自由节点的路径1、路径2,更新步骤2中的序列3->4,4->3。

步骤 4　依次对步骤3中求出的每一个中间序列的“起点”,求出起点 V_{begin} 1到它们之间的局部最短距离。

序列1:以3为“起点”:Dijkstra(1,3)。

序列2:以4为“起点”:Dijkstra(1,4)。

步骤 5　依次对步骤3中求出的每一个中间序列的“终点”,求出它们到 V_{end} 2之间的局部最短距离。

序列1:以4为“终点”: Dijkstra(4,2)。

序列2:以3为“终点”:Dijkstra(3,2)。

步骤 6　将步骤4,步骤3,步骤5求出的三段路径连接起来即得经过中间节点集的“全程最短路径”。

序列1:1->3->1->4->2。

序列2:1->4->1->3->2。

步骤 7　对于步骤6中所求出的每一条局部的“全程最短路径”,求出距离最小的路径,1->3->1->4->2就是满足条件的全程最短路径,对应的距离是4。距离相同的路径其实还有1->4->1->3->2,但是算法只求出一条最短路径。如需求出最短距离相同的所有路径,只需将算法伪代码第27~32行略作修改即可。

3.4　算法分析

记 n, m 分别为总节点和必须经过的中间节点的个数。由于Dijkstra算法的时间复杂度为 $O(n^2)$ (n 为网络节点个数),与边数无关,特别适用于稠密图。本算法以Dijkstra算法为基础,算法的时间复杂度与顶点个数和中间节点个数有关,与边数无关,也适用于稠密图。

理论上,该算法的时间复杂度为 $O(n^2 \times m!)$ 。从中不难看出,算法的时间复杂度对网络中节点总数不敏感,可以适用于大规模网络,而对于需要经过的中间节点个数却非常敏感,不太适用于中间节点个数太多的情况。

3.5　实验仿真

实验环境为:windows7系统,2.2GHz i3-2330CPU,4 GB内存,编程语言是C++,译环境为VS2010。不失一般性,下面以2013届“中兴捧月”杯程序设计大赛复赛试题2[18]中提供的数据进行测试。

2013届“中兴捧月”杯程序设计大赛复赛试题2题干部分:在一个可以支持至少上千个节点的网络拓扑中,边是有向无权的,两点之间最多有一条边,给出源点

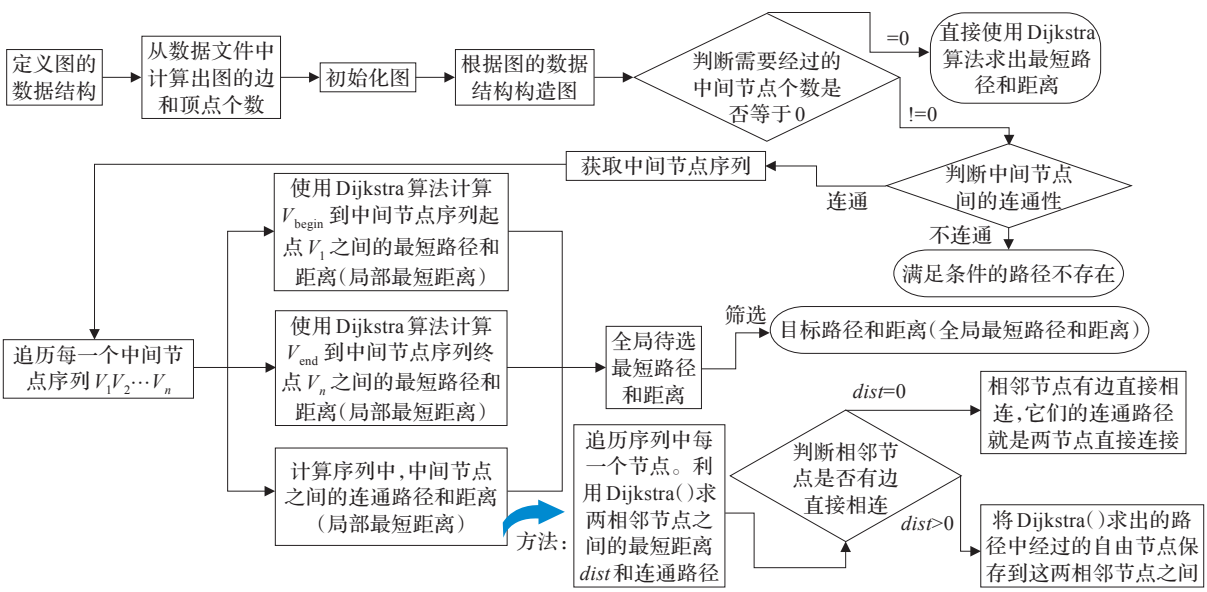

图4　算法流程图

和终点两个点，需要找出满足条件的最短路径：这条路径必须经过一些给定中间节点(节点个数不超过10个)[18]。

3.5.1　实验1：追踪程序的求解过程，验证实验结果的正确性

为了便于追踪、观察程序执行的过程和直观验证最终结果的正确性，本实验从文献[18]所提供的数据中，选取具有50个节点的某稀疏网络进行测试，详细说明程序的求解过程和结果。

网络节点数据如图5。

```
1  2,28      21 24,32     41 43,42
2  4,48      22 26,27     42 43,50
3  9,45      23 26,34     43 43,6
4  10,1      24 27,28     44 44,43
5  10,11     25 27,35     45 45,7
6  11,12     26 28,36     46 45,8
7  11,2      27 29,37     47 46,37
8  12,13     28 30,31     48 46,47
9  13,14     29 32,40     49 47,48
10 15,23     30 32,33     50 48,40
11 16,24     31 32,31
12 17,25     32 33,25
13 18,19     33 33,41
14 18,1      34 36,44
15 20,21     35 37,38
16 20,29     36 37,45
17 20,3      37 39,30
18 21,22     38 39,4
19 22,23     39 40,5
20 23,24     40 40,49
```

图5　测试一使用的网络节点数据

上述网络节点数据所形成的网络拓扑结构如图6。

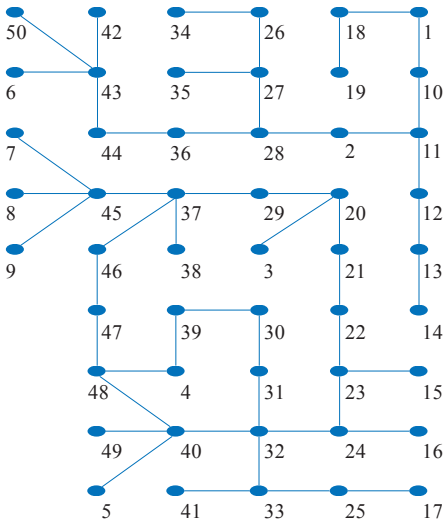

图6　测试1使用的网络拓扑结构

为便于完整展示程序的执行过程，这里选取起点、终点和必须经过的中间节点时，需要确保满足条件的路径是存在的。同时，为了使中间结果序列不至太长而影响到直观验证结果的正确性，这里选取起点20，终点32，需要经过的中间节点为15，16，31来进行实验展示。

关键的求解步骤及部分中间结果如表1。

通过观察该实验所选择的网络图的拓扑结构，不难发现实验结果的正确性。另外，通过在几十组不同拓扑结构和节点数的网络中随机选取10个以内数量不等(中间节点个数选取10个以内的原因见实验2)的中间节点进行测试(无论满足条件的路径是否存在)，通过直观验证，证明了该算法在计算满足条件的最短路径和距离的准确率达到了100%。

表1　实验1中程序执行关键步骤及结果

关键步骤	结果
(1)判断中间节点个数是否为0和中间节点连通性	vecMidNode [3](15,16,31)； bConnect true
(2)求中间节点序列	[capacity] 6 [0] [3](15,16,31) [1] [3](15,31,16) [2] [3](16,15,31) [3] [3](16,31,15) [4] [3](31,15,16) [5] [3](31,16,15)
(3)求中间节点序列之间的最短连通路径及距离	[0] [5](15,23,24,16,31) [1] [6](15,23,24,32,31,16) [2] [5](16,24,23,15,31) [3] [5](16,24,32,31,15) [4] [6](31,32,24,23,15,16) [5] [5](31,32,24,16,15)
(4)求起点(V_{begin})到每个中间节点序列起点的连通路径及距离	[0] [5](20,21,22,23,15) [1] [5](20,21,22,23,15) [2] [6](20,21,22,23,24,16) [3] [6](20,21,22,23,24,16) [4] [7](20,21,22,23,24,32,31) [5] [7](20,21,22,23,24,32,31)
(5)求终点(V_{end})到每个中间节点序列起点的连通路径及距离	[0] [2](31,32) [1] [3](16,24,32) [2] [2](31,32) [3] [4](15,23,24,32) [4] [3](16,24,32) [5] [4](15,23,24,32)
(6)将步骤(3)、(4)、(5)所得路径距离相加即得全局待选最短距离。从中筛选出距离最短的路径，即为目标路径	[10](20,21,22,23,15,23,24,16,31,32)

3.5.2　实验2：探索该算法在实际应用中面对不同网络环境时的时间效率

由于难以有效地对程序执行的空间效率进行监测，下面仅通过在程序中设置计时器记录程序的执行时间来粗略探索算法的时间效率。分别通过在50个，500个和5 000个节点的网络中的多组数据测试，求算术平均后，得出数据如表2~4。因为中间节点个数超过10时，程序执行时间较长。因此，在表2~4中，中间节点的个数分别取值为1，5，10。

表2　程序在50个节点的网络中执行的时间表

图的节点总数	50	50	50
中间节点个数	1	5	10
时间消耗/ms	0	110	88 660

表3 程序在500个节点的网络中执行的时间表

图的节点总数	500	500	500
中间节点个数	1	5	10
时间消耗/ms	0	110	88 660

表4 程序在5 000个节点的网络中执行时间表

图的节点总数	5 000	5 000	5 000
中间节点个数	1	5	10
时间消耗/ms	577	13 430	大于600 000

由此可见,即使面临庞大的网络图,只要需要经过的中间节点数不太多,程序的执行时间都非常可观。本算法具有很好实际应用前景。

但是,当中间节点个数大于10时,程序的执行时间较长。同时,由于网络的稠密稀疏程度存在差异,中间节点集在网络中的分布情况存在差异,编写程序的语言和程序的编写逻辑存在差异,用于编程的计算机性能存在差异等,仅仅依靠节点个数测试出来的程序的执行时间仅能作为参考,无法准确预测算法在不同网络拓扑图中的实际使用效果。另外,从时间复杂度公式和本算法编制的程序执行结果都可以看出,随着中间节点个数增大,程序执行的时间增加非常明显。

4 结束语

实际上,无论是“邮递员”、“旅行家问题”还是公交车专线设计和网络选路问题,还是其他类似问题,不同节点之间的距离或者网络时延都是不一样的,是带权的,因此带权图更适合实际生产生活实际。针对带权图的算法跟无权图十分相似,对于上面提出的算法,稍作修改即可用于带权图。

(1)邻接矩阵的元素不再只是1和max了,而是实际的权重(边的长度,网络的时延等)。

(2)算法描述部分“求中间节点之间的距离和路径的方法”中伪代码的第五行,若序列中的俩相邻节点是有边直接相连的,则距离为邻接矩阵中对应行列的值。

针对有向图,算法跟无向图完全相同,无需更改即可直接使用。

在中间节点个数较少时,该算法时间复杂度极低。该算法在需要经过的中间节点个数较少的最短路径问题中,例如,邮递员送信路线选择,针对重要线路和站点的公交路线设计,以及旅行家问题等许多基于图的优化问题中,都有着广泛的应用前景。为在经过指定节点集的较大规模稠密网络中求最短路径提供了新的思路。尽管如此,如何改进该算法,以便在海量网络节点数和大量中间节点集的情况下提高算法时空效率,还需要进一步的研究和探索。

参考文献:

[1] Dijkstra E W.A note on two problems in connexion with graphs[J].Numerische Mathematik,1959,1:269-271.

[2] Sang Yongsheng,Yi Zhang.A modified pulse coupled neural network for shortest path computation[J].Journal of Computational Information Systems,2010,6(9):3095-3102.

[3] Goldberg A V,Kaplan H,Werneck R F.Reach for A*:efficient point-to-point shortest path algorithms[C]//ALENEX,2006,6(2):129-143.

[4] Orlin J B,Madduri K,Subramani K,et al.A faster algorithm for the single source shortest path problem with few distinct positive lengths[J].Journal of Discrete Algorithms,2010,8(2):189-198.

[5] 张锦明,洪刚,文锐,等.Dijkstra最短路径算法优化策略[J].测绘科学,2009,34(5):105-106.

[6] 王智广,王兴会,李妍.一种基于Dijkstra最短路径算法的改进算法[J].内蒙古师范大学学报:自然科学汉文版,2012,41(2):195-200.

[7] 徐庆征,柯熙政.求解最短路径的遗传算法中若干问题的讨论[J].计算机工程与设计,2008,29(6):1507-1509.

[8] Chan T M.More algorithms for all-pairs shortest paths in weighted graphs[J].SIAM Journal on Computing,2010,39(5):2075-2089.

[9] 林澜,闫春钢,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.

[10] 唐晋韬,王挺,王戟.适合复杂网络分析的最短路径近似算法[J].软件学报,2011,22(10):2279-2290.

[11] 王卫强,孙强.求图中受顶点数限制的所有最短路径的算法[J].计算机工程与设计,2008,29(7):1754-1757.

[12] 叶金平,朱征宇,王丽娜,等.智能交通中的高效多准最短路径混合算法[J].计算机应用研究,2011,28(9):3301-3304.

[13] 柴登峰,张登荣.前N条最短路径问题的算法及应用[J].浙江大学学报:工学版,2002,36(5):531-534.

[14] 王峰,游志胜,曼丽春,等.Dijkstra及基于Dijkstra的前N条最短路径算法在智能交通系统中的应用[J].计算机应用研究,2006,23(9):203-205.

[15] 姚广铮,孙壮志,孙福亮,等.北京奥运公交专线规划及评价方法[J].城市交通,2008,6(3):29-34.

[16] Koutsoupias E,Papadimitriou C H.On the greedy algorithm for satisfiability[J].Information Processing Letters,1992,43(1):53-55.

[17] Martello S,Toth P.Knapsack problems[M].New York:Wiley,1990.

[18] “中兴捧月”杯程序设计大赛复赛试题[EB/OL].[2013-12-10].http://company.dajie.com/zte/competition?r=13844132497.