一种基于 Dijkstra 最短路径算法的改进算法

王智广，王兴会，李　妍

（中国石油大学（北京）信息学院，北京 102249）

摘　要：Dijkstra 算法是求解最短路径的经典算法，是在许多应用中解决最短路径问题的理论基础，但实际应用中涉及的许多限制条件要求人们必须对该算法进行改进和优化. 在分析经典 Dijkstra 算法思想的基础上，给出 Dijkstra 算法的一种改进算法. 在该算法中图的存储表示采用邻接表的方式，避免邻接矩阵在工程应用中的局限性. 在最短路径的计算过程中，采用优先级队列与反向 N 叉树相结合的方式，以便通过实现可降级的优先队列来改进 Dijkstra 算法. 给出了改进形 Dijkstra 算法的方法和流程，分析了其算法复杂度，并对改进后的算法进了详细的分析和测试.

关键词：Dijkstra 算法；路网；邻接表；反向 N 叉树；最短路径

中图分类号：TP 319　　**文献标志码**：A　　**文章编号**：1001-8735(2012)02-0195-06

在软件工程中经常会用到计算最短路径的算法[1-3]，例如在路网监控软件中，对可疑车辆进行监控和跟踪时，经常会计算点到点的最短路径. 在计算最短路径的算法中，Dijkstra 算法是比较经典的算法，在该算法中图的存储表示为邻接矩阵. 然而实际路网的结构属于典型的边稀疏网络，如果在软件工程中使用邻接矩阵的方式来存储图（路网），将浪费大量的内存，不仅使算法的空间复杂度增大，而且多次访存操作使得算法的时间复杂度也随之增大. 因此，路网的现实情况需要我们去寻求新的算法，使得该算法在邻接表结构中做最短路径的计算，以适应与路网相关的软件工程中的需要.

1　算法设计思想

1.1　数据结构

为了详细描述本算法的设计思想，首先描述一下在本算法中使用的数据结构，如图 1 所示.

图 1　优先级队列

Fig. 1　Priority queue

在图 1 的优先级队列中，权值小的节点在队列前端，因此在该队列中出队列的操作不变，依然是从队头取数据，而数据入队列时与一般的队列有所不同，按节点的权值从小到大的顺序插入队列中.

邻接表的原理如图 2 所示. 把从某一节点出发的所有边连接在以该节点为首的单链表中，单链表的每一个节点代表一条边，称为边节点. 节点中存储 3 个数据，第 1 个数据是该节点的编号（nNodeID），第 2 个数据是路段权值（dWeight），第 3 个数据是指向同一链表中下一个节点的指针（pLink）. 邻接表有效地解决了邻接矩阵在处理稀疏矩阵时浪费内存空间的问题[1].

城市公路的网络结构，通常为网格结构，特殊情况下路网中也会出现 5 道口、6 道口的情况. 为不失一般性，本算法在计算最短路径的时候考虑了这些特殊的情况.

图 3 中的箭头指向表示节点间的路由依赖关系，如节点 A 指向节点 B，表示从源节点 O 到节点 A 的最短路径，需要经过节点 B，即 A 依赖于 B. 某节点只可以依赖一个节点，但可以被多个节点依赖. 每个节点的数据结构描述如下：

收稿日期：2011-12-22

基金项目：国家自然科学基金资助项目(60803159)

作者简介：王智广(1964 －)，男，内蒙古通辽市人，中国石油大学(北京)教授，博士，主要从事分布式计算、软件工程、计算智能方向的研究，E-mail：cwangzg@cup. edu. cn.

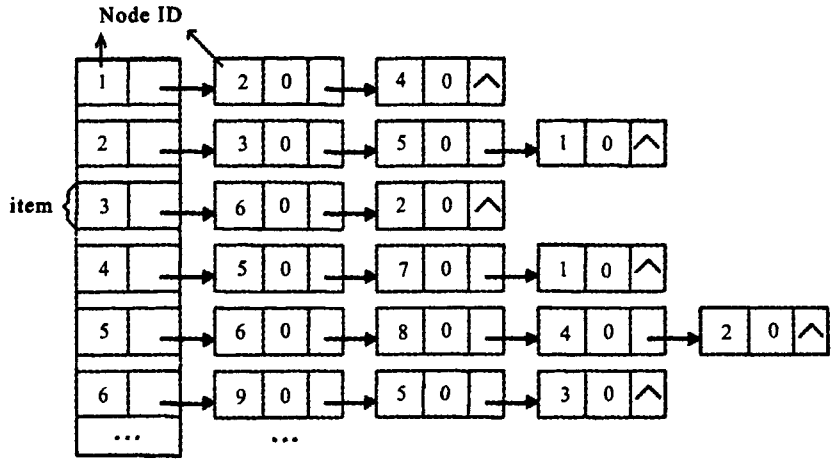

图 2 邻接表结构示意图

Fig. 2 Schematic diagram of adjacency list

```
struct NodeData
// 节点(队列节点、反向 N 叉树节点)信息结构
{
int nNodeID;  // 节点 ID
double dWeight;  // 路由到该节点的路段总长
int nDependence;  // 依赖于该节点的节点个数
NodeData * pLink;
// 指向前驱节点(该节点依赖的节点)的指针
};
```

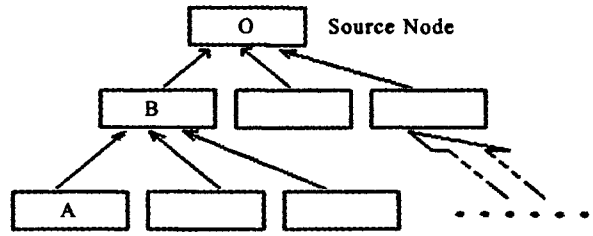

图 3 反向 N 叉树逻辑结构图

Fig. 3 The logical structure of *n*-ary-reverse tree

1.2 算法思想

1.2.1 *算法思想* 算法的流程图如图 4 所示. 算法的主要设计思想描述如下：

步骤 1：将源节点入队列；

步骤 2：取队头节点 h，若 h 节点是目标节点时，则从目标节点到源节点的这一路由路径就是所求的最短路径，这时候需要释放队列中剩余的节点及其依赖关系，算法结束；

步骤 3：将 h 的邻接节点入队列(注意：若某邻接节点已出队列，则对这样的节点不作任何处理；若某邻接节点已在队列中，则需判断该节点的路由路线；若某邻接节点尚未入队列，则将其加入队列)；

步骤 4：若 h 的依赖计数为 0，则释放 h 节点及其依赖. 返回步骤 2.

1.2.2 *优先级队列和反向 N 叉树* 优先级队列在寻找目标节点的过程中，就像是一条贪吃的蛇，它的长度不断扩张. 需要注意的是，图 5 只是优先级队列活动的一个示意图，用以帮助理解算法的计算过程. 在实际计算中，路网节点在优先级队列中的顺序，不一定严格按照上图 5 中“环”上的先后顺序，这是由优先级队列的性质决定的.

采用优先级队列，使得在计算最短路径的过程中，从源节点向周围的节点均匀扩散. 因此，目标节点离源节点越近，计算最短路径的时间越短.

优先级队列的数据节点与反向 N 叉树的树节点在内存中的载体都是相同的. 当队列为空时，沿着目标节点向前找，一直到源节点，就是所求的最短路径. 此时，其他节点(被淘汰的节点)所占用的对空间已全部释放，内存中仅余最短路径上的全部节点. 这里需要注意的是，由于采用反向 N 叉树记录节点间的路由依赖关系，此时的最短路径的排列是从目标节点到源节点的顺序，若在无向图中计算最短路径，可将源节点与目标节点交换位置，即从目标节点出发查找源节点. 若在有向图中计算最短路径，从源节点出发寻找目标节点，得到最短路径后，可利用栈的机制改变这一顺序.

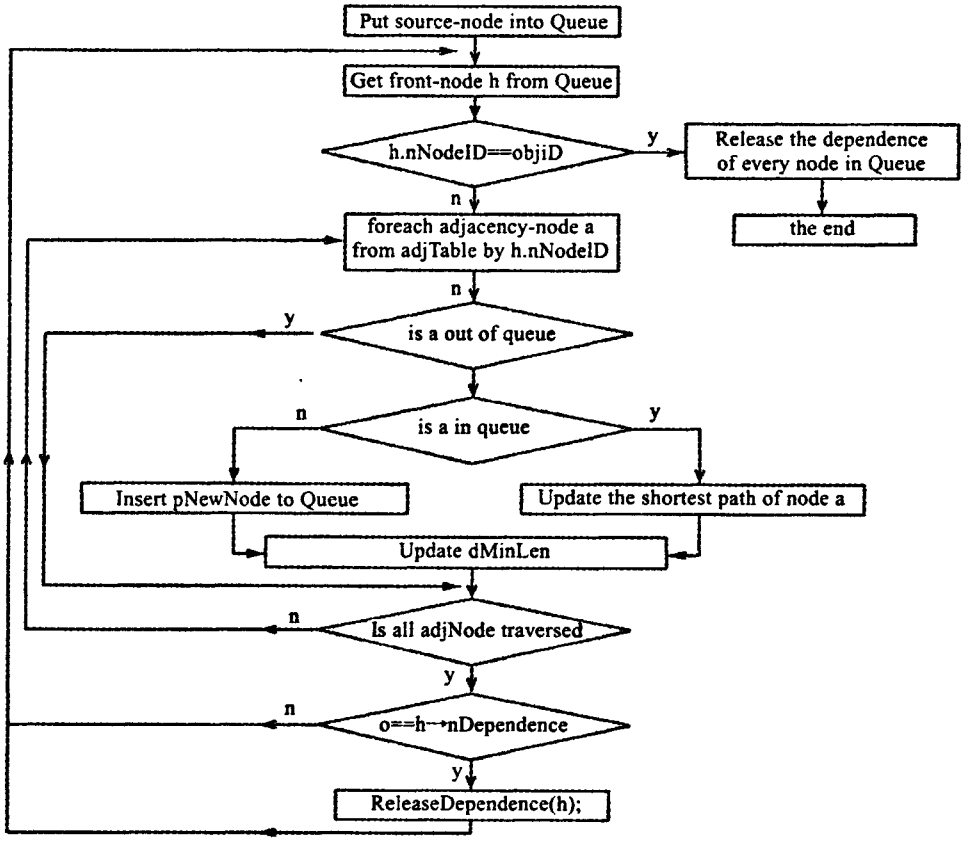

图 4　算法流程图

Fig. 4　Flow chart of algorithm

1.3　局部性原理对算法的影响

局部性原理的历史可以追溯到上世纪 60～70 年代，当工程师观察 CPU 进程的访问内存时发现，一个被访问的存储单元倾向于一个小的聚集的连续区域. 一般来说，局部性原理可以描述为两个方面：① 时间局部性. 如果一个数据项被访问，那么它即将被再次访问. 程序循环和堆栈是时间局部性的决定因素；② 空间局部性. 将要被访问的数据项和当前正在使用的数据项在内存空间分配的地址相近. 空间局部性的决定因素包括引导序列的执行，数据在数组中的连续存储等.

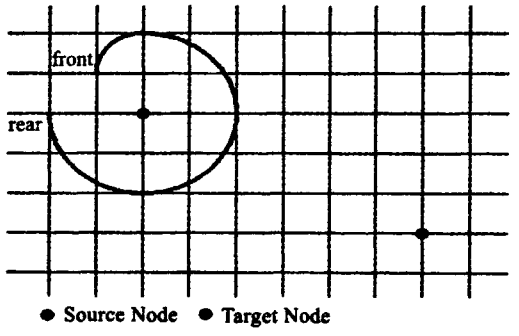

图 5　优先级队列活动示意图

Fig. 5　The activity diagram of priority queue

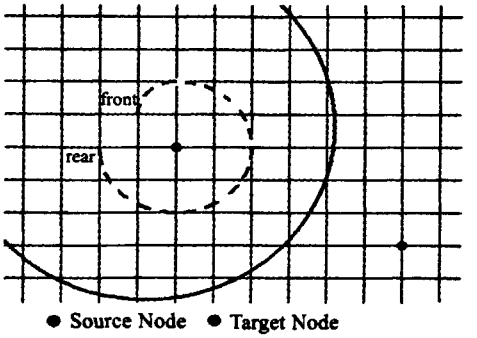

图 6　优先队列的行为

Fig. 6　Behavior of the priority queue

局部性原理的应用范围很宽，文献[4]详细阐述了 INTEREST-BASED LOCALITY，通过这种方式，系统总负荷减少了 3～7. 文献[5]以及其他介绍 SkipNet 网络局部性特点的人在内容和局部性路由路径方面都有所成就.

那么，在局部性原理的指导下，会对本算法所采用的优先队列产生什么样的影响呢？在计算最短路径的

时候，优先队列表现为螺旋形状，并且从源节点到目标节点均匀遍布. 在这个过程中，每一个将要从队列中添加或删除的节点有两个显著特征：在宏观方面，节点的权重会变得越来越大；在微观方面，节点的权重变化在一个微小的范围内波动. 图 6 描述了这个过程. 因此，利用节点在队列中的局部特征以及改进的队列结构和存取算法，将进一步减少算法的时间复杂度，提高了算法的效率.

2 复杂度分析

2.1 空间复杂度

本算法在计算最短路径的过程中，引入了优先级队列，该队列占用的空间随着目标节点由近及远而逐渐增大，在极端的情况下，需要遍历全部节点. 另外，由于引进辅助标记数组，需要额外 n 个数据空间，n 为图中节点总数，因此本算法的最大空间复杂度为 $O(2n)$.

2.2 时间复杂度

不考虑局部性数据时，需要遍历图中全部的节点，即外循环需要计算 n 次. 内循环主要考虑在队列中查找某个节点或将某个节点入队列的情况，队列的平均长度为(1/2)×2(2n)1/2，而队列的平均搜索长度为队列的长度的一半，因此内循环的平均计算次数为(1/2)×2(2n) 1/2×(1/2)次，即(1/2)×(2n) 1/2. 这样，本算法的最大时间复杂度为 $O(n\times(1/2)\times(2n)\ 1/2)$，可粗略估计为 $O(n3/2)$.

当考虑局部性数据的优先队列时，我们发现，该算法外部循环也需要计算 n 次(见图 9). 内循环平均查找长度为 k 次，这反映了优先队列波动的平均数. 所以该算法的最大的时间复杂度为 $O(k\times n)$，即局部性原理将时间复杂度从 $O(n3/2)$ 降低到 $O(k\times n)$.

3 算法测试

测试本算法的路网数据由 73 个节点和 233 条路段构成. 测试过程完成后，得到一个共有 5 329 最短路径的测试样品，也就是说，测试包括所有点与点之间的路径. 用这些数据除以同样长度的最短路径，然后计算这些数据的均值和标准偏差，结果见表 1 和表 2.

表 1 测试结果

Tab. 1 The result of testing data

源与目标节点间的节点个数	平均计算时间	标准偏差	队列波动平均次数	标准偏差	关联节点平均个数	标准偏差	样品数量
2	9.905	9.694	6.328	8.139	7.586	3.523	232
3	38.167	29.147	27.921	24.357	15.880	6.283	407
4	81.205	55.637	59.922	46.373	24.960	9.521	526
5	127.917	79.319	93.956	65.690	33.559	11.856	590
6	169.377	90.268	122.651	73.648	41.329	12.856	605
7	204.641	90.974	145.837	73.227	48.283	12.609	601
8	230.362	84.130	161.106	66.649	53.837	11.393	566
9	251.134	74.420	172.821	58.913	58.509	9.732	491
10	265.372	63.232	179.328	49.924	62.404	8.156	403
11	274.000	52.908	181.411	41.579	65.408	6.709	316
12	277.590	42.651	179.510	33.226	67.620	5.365	234
13	280.468	34.123	177.165	27.055	69.500	4.245	158
14	281.667	28.227	174.207	23.573	70.805	3.569	87
15	280.816	23.918	170.469	22.23	71.781	2.166	32

测试结果可以通过图 7、图 8、图 9 加以分析. 图 7 中横轴代表节点个数，纵轴代表遍历各节点的计算次数，从图 7 显示的测试结果发现，算法的时间复杂度随着距离(源节点到目标节点)的增加而增大. 分析测试数据发现，当源节点到目标节点间之间的距离和大于或等于 8 时，该算法的时间复杂度无法达到预期水平. 这表明，在特殊条件下(源节点和目标节点都在道路网络的边缘)，该算法的时间复杂度远小于 $k\times n$.

表 2　测试结果分析

Tab. 2　The analysis of testing data

源与目标节点间的节点个数	样品数量	复杂性因子 k	平均波动次数	队列波动比例
2	232	1.305694701	0.834168205	0.638869258
3	407	2.403463476	1.75824937	0.731548196
4	526	3.253405449	2.400721154	0.737910227
5	590	3.811704759	2.799725856	0.734507532
6	605	4.098260301	2.96767403	0.724130195
7	601	4.23836547	3.020462689	0.712648003
8	566	4.278878838	2.992477293	0.699360138
9	491	4.292228546	2.953750705	0.688162495
10	403	4.252483815	2.873661945	0.675760819
11	316	4.18909002	2.773529232	0.662083942
12	234	4.105146406	2.654687962	0.646673151
13	158	4.035510791	2.549136691	0.631676341
14	87	3.978066521	2.460377092	0.618485659
15	32	3.912121592	2.374848498	0.607048744

分析表 1 和表 2 可以发现，复杂度系数 k 随源节点到目标节点之间距离的增加而增加，系数 k 由下式估算：

$$k=\frac{\text{平均计算时间}}{\text{关联节点平均个数}},$$

其中平均计算时间和关联节点平均个数的信息可以由表 1 得到. 尽管在测试结果中，系数 k 基本稳定在 4～4.5，但实际上它依然是一个随着 n 增长而增长的数据.

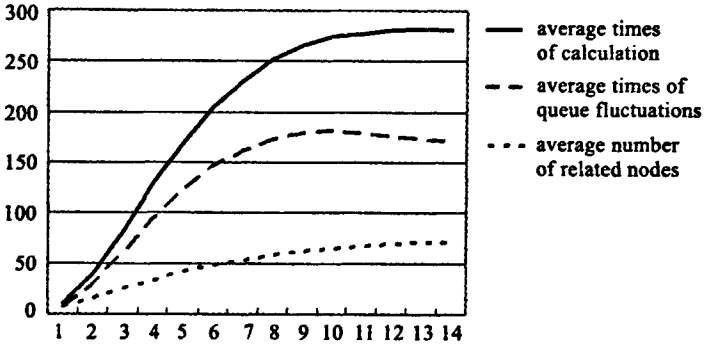

图 7　算法的时间复杂度

Fig. 7　Time complexity of algorithm

图 8 描述了优先队列中一个节点带来的队列波动的平均次数. 图中的曲线显示出明显的下降趋势，其原因在于随着最短路径长度的增加，源节点和目标节点靠近道路网络的边缘，使得参与计算的节点减少，因此结果显示出队列波动减小.

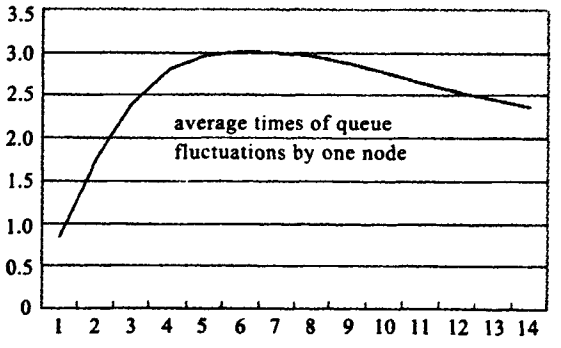

图 8　队列波动

Fig. 8　Queue fluctuations

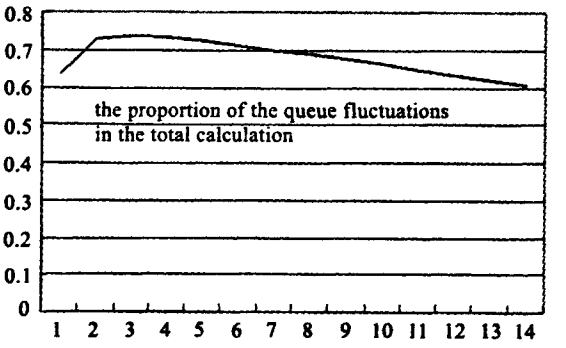

图 9　波动率和总计算队列

Fig. 9　The ratio of queue fluctuations and total calculation

图 9 显示了在总计算次数中队列的波动率(见表 2)，其表达式为 $\text{RATIO}=\frac{\text{队列波动平均次数}}{\text{平均计算时间}}$，其中队列波动平均次数和平均计算时间的信息可以由表 1 得到.

总体而言，波动率不超过0.75. 这一结果至少表明了3个方面的问题：① 波动率(0.75)和 k 值在宏观上是一致的，这表明该实验数据是有效的；② 比值 RATIO 表明，在优先级队列中该算法不会在节点波动上浪费太多的时间；③ 由这一结果可知，算法的时间复杂度和节点数目 n 之间成线性关系.

4 小结

本文提出了计算点到点的最短路径计算算法，如果在工程实践中需要计算单源最短路径，需要对算法做一些局部调整. 首先，每次出队列的节点数据都暂存到数组(或者链表)中；其次，当节点的被依赖计数为0时，不释放该节点. 这样，当队列为空时，留在数组(或者链表)中的各节点构成的反向 N 叉树正好是个节点的最短路径，树根正好是源点. 这种情况下，算法的时间复杂度不变而算法的空间复杂度增加了 n. 在软件工程中，可以根据具体需要调整邻接表的结构，以便消除增加的算法空间复杂度.

每一种最短路径算法都有其特定的适用范围. 到目前为止，没有一种算法能适用于所有网路结构. 本算法也不例外. 由于本算法在执行的过程中需要不断刷新邻接表中的节点计数信息，因此，本算法在边稀疏网络中(如路网)的运算效率较高，而在边稠密的网络中算法的运行效率会变低.

参考文献：

[1] 殷人昆. 数据结构 [M]. 2版. 北京：清华大学出版社，2005，120-135.

[2] 陆锋，卢冬梅，崔伟宏. 基于四叉堆优先级队列及逆邻接表的改进型 Dijkstra 算法 [J]. 中国图像图形学报，1999(12)：1044-1050.

[3] Cherkassky B V，Goldberg A V，Radzik T. Shortest Paths Algorithms：Theory and experimental evaluation [J]. Mathematical Programming，1996 73：129-174.

[4] Sripanidkulchai K，Maggs B，Zhang H. Efficient Content Location Using Interest-Based Locality in Peer-to-Peer Systems [J]. IEEE INFOCOM，2003(3)：2166-2176.

[5] Harvey N，Jones M B. A Scalable Overlay Network with Practical Locality Properties [C]// USENIX Association. In Proceedings of the 4th Conference on USENIX Symposium on Internet Technologies and Systems Berkeley，CA，USA，2003：9.

A Kind of Shortest Path Algorithm Based on Dijkstra

WANG Zhi-guang，WANG Xin-hui，LI Yan

(*College of Information and Engineering，China University of Petroleum，Beijing* 102249，*China*)

Abstract：Shortest path algorithm has important practical value in engineering practice and Dijkstra algorithm is a classical algorithm for solving the shortest path. Dijkstra algorithm is the theoretical basis for many practical works to solve the shortest path problem，but many of the restrictions involved in the actual project，therefore，there must be the algorithm improvement and optimization. On the basis of analyzing the classical Dijkstra algorithm，the paper discusses an improved algorithm of Dijkstra algorithm. By using Map which is stored in the algorithm to the adjacency table，it can avoid the limitations of the adjacency matrix in the project；Moreover，in the calculation of the shortest path，using a combination of the priority queue with reverse N-tree. In order to down grade the priority queue to improve the Dijkstra algorithm. The paper puts forwards the methods and processes to improve the shape Dijkstra algorithm，and analyzes the complexity of the algorithm and do some detailed testing and results analysis.

Key words：Dijkstra algorithm；the road network；adjacent table；the reverse N-tree；the shortest path

【责任编辑 陈汉忠】

一种基于Dijkstra最短路径算法的改进算法

作者： 王智广， 王兴会， 李妍， WANG Zhi-guang， WANG Xin-hui， LI Yan
作者单位： 中国石油大学(北京)信息学院,北京,102249
刊名： 内蒙古师范大学学报（自然科学汉文版） ISTIC
英文刊名： Journal of Inner Mongolia Normal University(Natural Science Edition)
年，卷(期)： 2012,41(2)
被引用次数： 7次

参考文献(5条)

1. 殷人昆 数据结构 2005

2. 基于四叉堆优先级队列及逆邻接表的改进型Dijkstra算法[期刊论文]-中国图象图形学报 1999(12)

3. Cherkassky B V;Goldberg A V;Radzik T Shortest Paths Algorithms:Theory and experimental evaluation 1996

4. Sripanidkulchai K;Maggs B;Zhang H Efficient Content Location Using Interest-Based Locality in Peer-to-Peer Systems 2003(03)

5. Harvey N;Jones M B A Scalable Overlay Network with Practical Locality Properties 2003

引证文献(6条)

1. 卢鹏飞,关英子 存在必行路段路网最优路的Dijkstra优化算法[期刊论文]-知识经济 2013(19)

2. 李晶,闫军 基于Dijkstra算法和Floyd算法的物流运输最短路径研究[期刊论文]-科技信息 2012(34)

3. 黄书力,胡大裟,蒋玉明 经过指定的中间节点集的最短路径算法[期刊论文]-计算机工程与应用 2015(11)

4. 刘定军,陈志刚,黄瑞 快速负环检测的负权最短路径算法[期刊论文]-计算机工程与设计 2014(10)

5. 李炯城,李桂愉,肖恒辉,黄海艺 快速检测低密度奇偶校验码围长的新算法[期刊论文]-计算机应用 2012(11)

6. 陈萱华,杨玲,李学亚 最短路径算法在自动测评系统中的应用[期刊论文]-计算机与现代化 2012(12)