文章编号:1000-5862(2013)01-0028-05

稀疏网络的 Floyd 动态优化算法

吴果林[1],金　珍[2],邓小方[3]
(1. 桂林航天工业学院信息工程系,广西 桂林 541004;2. 南昌工程学院理学系,江西 南昌 330099;
3. 江西师范大学软件学院,江西 南昌 330022)

摘要:通过对 Floyd 算法进行研究,提出了一种新的求取任意两点间最短路径的算法:Floyd 动态优化算法. 该算法通过引入插入数组、可达数组以及可发数组,使得算法在求解最短路径前自动修改能够最小化路径的节点,剔除一些无用的节点,最小化语句执行的次数. 算法分析表明,新算法在稀疏网络中比 Floyd 算法在性能上有较大的提高.

关键词:最短路径;Floyd 算法;插入数组;可达数组;可发数组
中图分类号:TP 301.6　　**文献标志码**:A

0　引言

最短路径问题是图与网络中的一个经典问题,在工程规划、地理信息系统、通信和军事运筹学等领域有着非常广泛的应用,对该问题算法的设计和改进研究有着重要的理论和应用价值. 经典的最短路径算法主要有 Dijkstra 算法和 Floyd 算法. Dijkstra 算法是一种用于计算从其中 1 个源顶点到其它所有顶点最短路径的高效算法,如果求所有点对之间的最短路径,可以通过对每个顶点执行 1 次 Dijkstra 算法,便得到任意两点间的最短路径,总的执行时间为 $O(n^3)$(n 为图的顶点的个数). 求取任意两点间最短路径,更为通用、简洁、高效的方法是 Floyd 算法,它的执行时间同样为 $O(n^3)$[1]. 这 2 种算法的缺陷是时间开销为顶点 n 的多项式,不能根据图的点数、边数的实际分布动态调整算法的时间开销,以使得算法的运行速度极大化、运行时间极小化.

近期关于求取任意两点间最短路径问题的研究有很多. 文献[2-3]分析 Floyd 算法的执行过程,发现图中很多插入的节点显然不能使路长变短,通过简化其运算,给出了最短路径问题的 Floyd 优化以及加速算法;文献[4]通过引入可达表、可发表以及仅 1 次插入矩阵的方法对 Floyd 算法进行了改进,改进后的算法尽管能够根据图的顶点和边数的实际分布调整自己的时间开销,但引入可达表、可发表以及仅 1 次插入矩阵后,算法的空间开销为 $4n^2$,比 Floyd 算法高出许多. 文献[5-6]给出了基于边序列的任意两点间最短路径算法以及改进算法,然而,它们都是针对图中边数较少的情况提出的优化方法,若边数较多或者图中每个顶点都存在路时,上述算法的执行时间为 $O(n^4)$,远高于 Floyd 算法的执行时间.

为此,本文通过分析 Floyd 算法的执行过程,利用文献[2-3]的优化思想,设计 1 个带权有向图求取任意两点间最短路径的 Floyd 动态优化算法,新算法能够根据有向图的点数、边数的实际分布动态调整算法的时间开销. 新算法的时间复杂度为 $O(n^2 + M \times \overline{\alpha}n \times \overline{\beta}n)$.

1　最短路径 Floyd 动态优化算法

1.1　Floyd 算法简介

为便于阐述 Floyd 算法的优化思想,先简单介绍 Floyd 算法[7]. Floyd 算法是通过带权邻接矩阵计算来实现的一种方法,其主要思想是从任意 2 个顶点 v_i 到 v_j 距离的带权邻接矩阵 $\boldsymbol{D}^{(0)}$ 开始,首先计算 $\boldsymbol{D}^{(1)}$,即计算 v_i 到 v_j 经过 1 次经转的所有可能路径,经过比较后得到最短路径,代替 $\boldsymbol{D}^{(0)}$ 中对应的路径,迭代出带权邻接矩阵 $\boldsymbol{D}^{(1)}$,$\boldsymbol{D}^{(1)}$ 中元素表示经过 1 次迭代后图中任意两点间最短路,即图中任意两点之间直接到达或只经过 1 个中间顶点的最短路. 然后依次计算 $\boldsymbol{D}^{(2)}$,$\boldsymbol{D}^{(3)}$,…,$\boldsymbol{D}^{(k-1)}$,$\boldsymbol{D}^{(k)}$ 中各元素表

收稿日期:2012-10-25
基金项目:广西省教育厅科研课题(201106LX717)资助项目.
作者简介:吴果林(1977-),男,江西新余人,讲师,硕士,主要从事运筹优化与非参数统计的研究.

示任意两点间不经过中间顶点或最多允许经过 2^k-1 个中间顶点的最短路. 当 $\boldsymbol{D}^{(k+1)}=\boldsymbol{D}^{(k)}$ 时,得到的带权邻接矩阵 $\boldsymbol{D}^{(k)}$ 中各元素表示对应两顶点之间的最短距离信息,成为最短距离矩阵. Floyd 算法的非形式化描述如下:

Step 1　求初始距离矩阵 $\boldsymbol{D}^{(0)}=(d^{(0)ij})$,其中当 i,j 相邻时,$d^{(0)ij}=W_{ij}$,当 i,j 不相邻时,$d^{(0)ij}=\infty$,$i,j=1,2,\cdots,n$;

Step 2　构造迭代矩阵 $\boldsymbol{D}^{(k)}=(d^{(k)ij})$,其中 $d^{(k)ij}=\min\{d^{(k-1)ir}+d^{(k-1)rj}\mid r=1,2,\cdots,n\}$;

Step 3　若 $\boldsymbol{D}^{(k+1)}=\boldsymbol{D}^{(k)}$,迭代终止; 否则,返回 Step 2.

最短路径 Floyd 算法的 MATLAB 语言描述(称为算法 1)如下:

```
function [D] = function_floyd(D)
n = length(D(:,1));
for r = 1:n
    for i = 1:n
        for j = 1:n
            if(D(i,r) + D(r,j) < D(i,j))
                D(i,j) = D(i,r) + D(r,j);
            end
        end
    end
end
```

算法 1 中 $\boldsymbol{D}$ 为带权邻接距离矩阵. 从算法 1 可知,不管顶点间是否存在边,它的时间复杂度为 $O(n^3)$,这是 1 个关于顶点数 n 的多项式函数. 显然,算法 1 存在很大的优化空间,可以设计这样 1 个算法,它能根据图中顶点间边的数量自动调整算法的时间开销,即当图中每个顶点都是孤立点时,新算法的时间复杂度应接近于 0,当图中任意两点间都存在边时,新算法的时间复杂度为 $O(n^3)$ [8].

1.2　Floyd 算法的动态优化

从算法 1 可以看出,为了设计出能够根据图中顶点间的边数自动调整时间开销的算法,关键是要减少算法 1 的执行次数,为此,先来分析算法 1 的运行过程. 易知,算法 1 首先从顶点 v_1 开始搜索,即选取 v_1 作为中间节点,寻找图中的任意两点 v_i,v_j 经过中间节点 v_1 后,v_i 与 v_j 的路径是否变短,如果变短,则用 v_i 与 v_j 之间新的路径代替原来它们之间的路径; 然后选取 v_2 作为中间节点,寻找图中的任意两点 v_i,v_j 经过中间节点 v_2 后,v_i 与 v_j 的路径是否变短,如果变短,则用 v_i 与 v_j 之间新的路径代替原来它们之间的路径; 依此类推. 在完成最后 1 个节点 v_n 的搜索后,便得到图中任意两点 v_i,v_j 经过所有节点的最短路径,即任意两点间的最短路径. 从对带权邻接矩阵 $\boldsymbol{D}$ 的修改来看,首先寻找第 1 列每个元素 d_{i1} $(i=1,2,\cdots,n)$ 与第 1 行每个元素 $d_{1j}(j=1,2,\cdots,n)$ 的和,判断其和是否小于元素 d_{ij},若和小于 d_{ij},则用 $d_{i1}+d_{1j}$ 的值修改元素 d_{ij} 的值; 其次寻找第 2 列每个元素 $d_{i2}(i=1,2,\cdots,n)$ 与第 2 行每个元素 $d_{2j}(j=1,2,\cdots,n)$ 的和,判断其和是否小于元素 d_{ij},若和小于 d_{ij},则用 $d_{i2}+d_{2j}$ 的值修改元素 d_{ij} 的值; 依此类推,直到第 n 列与第 n 行的每个元素搜索完毕,最后得到任意两点间的最短路径.

上述分析可知,能够分别对算法 1 进行优化. 首先,对于第 1 层循环语句的优化. 由于第 1 层循环语句是搜索使两点间路径变短的中间节点,故对于图中那些出度或入度为 0 的顶点以及孤立点,它不会成为任何顶点间最短路径的中间点,应该避免这样无关的点参加循环以减少时间的复杂度. 为此,可以设计 1 个插入数组 $\boldsymbol{A}$,$\boldsymbol{A}$ 中的元素就是图中那些出度和入度不为 0 的顶点,M 表示插入数组 $\boldsymbol{A}$ 中元素的个数. 于是,可用读取插入数组 $\boldsymbol{A}$ 中的元素代替执行第 1 层循环语句,当图中所有顶点的出度和入度都不为 0 时,读取插入数组 $\boldsymbol{A}$ 中的元素变成执行第 1 层循环语句,这样就动态化了第 1 层循环语句. 如在图 1 的赋权有向图中,v_1 的入度为 0,v_3 的出度为 0,v_5 为孤立点,若选取 v_1,v_3,v_5 作为中间节点,则不可能使图中点间的距离变短,故 $\boldsymbol{A}$ 的元素为 2 个顶点 v_2,v_4,即 $\boldsymbol{A}=[2,4]$.

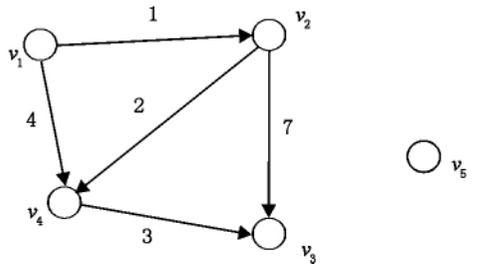

图 1　具有 5 个顶点的赋权有向图

其次,对于第 2 层循环语句的优化. 由于第 1 层循环语句是查找可达第 1 层循环语句给定的中间节点 A_k 有路且能够最小化路径(v_i,v_j) 的点,因此对于那些与中间节点 A_k 无路的点,应该避免它们参加循环,以节省算法的运行时间. 为此,与中间节点 A_k 相对应,引入可达数组 $\boldsymbol{R}^{(k)}$ $(k=1,2,\cdots,M)$,数组 $\boldsymbol{R}^{(k)}$ 的每个元素表示能够到达中间节点 A_k 的顶点,通过对数组 $\boldsymbol{R}^{(k)}$ 中的元素的搜索,便可发现最小化路径 (v_i,v_j) 的节点 v_i. 于是,第 2 层循环语句的运行就简化为对可达数组 $\boldsymbol{R}^{(k)}$ 的元素的搜索. 如在图 1 的赋

权有向图中,由前面的分析可知,能够作为中间节点的顶点为 v_2,v_4,则插入数组 $A=[2,4]$,当选取 v_2 作为中间节点时,能够到达节点 v_2 的顶点只有 v_1,故 $R^{(1)}=[1]$,插入节点 v_2 后,距离矩阵由 $D^{(0)}$ 修改为 $D^{(1)}$:

$$D^{(0)}=\begin{bmatrix}0&1&\infty&4&\infty\\ \infty&0&7&2&\infty\\ \infty&\infty&0&\infty&\infty\\ \infty&\infty&3&0&\infty\\ \infty&\infty&\infty&\infty&0\end{bmatrix}\rightarrow$$

$$D^{(1)}=\begin{bmatrix}0&1&8&3&\infty\\ \infty&0&7&2&\infty\\ \infty&\infty&0&\infty&\infty\\ \infty&\infty&3&0&\infty\\ \infty&\infty&\infty&\infty&0\end{bmatrix}.$$

此时,能够到达节点 v_4 的顶点有 v_1,v_2,故 $R^{(2)}=[1,2]$.

最后,对于第 3 层循环语句的优化. 考虑到第 3 层循环语句是查找第 1 层循环语句给定的中间节点 A_k 可发的且能够最小化路径(v_i,v_j)的点,因此对于那些与中间节点 A_k 无路的点,应该避免它们参加循环,以节省算法的运行时间. 类似于可达数组 $R^{(k)}$,引入可发数组 $C^{(k)}$ $(k=1,2,\cdots,M)$,数组 $C^{(k)}$ 的每个元素表示中间节点 A_k 可发的顶点,通过对数组 $C^{(k)}$ 中的元素的搜索,便可发现最小化路径(v_i,v_j)的节点 v_j. 同理,第 3 层循环语句的运行就简化为对可发数组 $C^{(k)}$ 的元素的搜索. 如在图 1 的赋权有向图中,插入数组 $A=[2,4]$,当选取 v_2 作为中间节点时,节点 v_2 可发的顶点有 v_3,v_4,故 $C^{(1)}=[3,4]$,插入节点 v_2 后,距离矩阵由 $D^{(0)}$ 修改为 $D^{(1)}$,此时,能够节点 v_4 可发的顶点有 v_3,故 $C^{(2)}=[3]$.

综上所述,最短路径 Floyd 动态优化算法的非形式化描述为

Step 1 由初始矩阵 $D^{(0)}$ 构造插入数组 A;

Step 2 对插入数组 A 的每个元素 A_k,计算出可达数组 $R^{(k)}$ 和可发数组 $C^{(k)}$;

Step 3 计算距离

$$d^{(k)R_i^{(k)},C_k^{(k)}}=\min\{d^{(k-1)R_i^{(k)},A_k}+d^{(k-1)A_k,C_k^{(k)}},d^{(k-1)R_i^{(k)},C_k^{(k)}}\mid k=1,2,\cdots,M\}\mid;$$

最短路径 Floyd 动态优化算法的 MATLAB 语言描述(称为算法 2)如下:

```
function [] = function_floydoptimize(D)
% floyd 动态优化算法
% 该函数自动根据每个节点的出度和入度计算任意两点间的最短路
    % D 为距离矩阵
n = length(D(:,1));
A(1) = 0;
k = 1;
for i = 1:n
    s = 1;
    it = 1;
    while(it)
        if((D(s,i) >0) &(D(s,i) < inf))
            it = 0;
        end
        s = s + 1;
        if(s = = n + 1)
            it = 0;
        end
    end
    if(s <(n + 1))
        A(k) = i;
        k = k + 1;
    end
end
A(k) = 0; k = 1;
while(A(k) ~ = 0)
    i = 1; j = 1;
    for w = 1:n
        if((D(w,A(k)) ~ = inf) &
(A(k) ~ = w))
            R(i) = w; i = i + 1;
        end
            if((D(A(k),w) ~ = inf) &
(A(k) ~ = w))
            R(j) = w; j = j + 1;
        end
    end
    for u = 1:(i - 1)
        for v = 1:(j - 1)
            if((D(R(u),A(k)) + D(A(k),
C(v))) <D(R(u),C(v)))
                D(R(u),C(v)) = D(R(u),
A(k)) + D(A(k),C(v));
            end
        end
    end
    k = k + 1;
end
```

1.3　Floyd 动态优化算法的时间和空间复杂度分析

易见，在算法 2 中第 1 阶段循环语句是查找在图中那些出度或入度为 0 的顶点以及孤立点，且是通过搜索带权邻接矩阵 $\boldsymbol{D}$ 的元素而获得插入数组 $\boldsymbol{A}$，故第 1 阶段循环语句的时间复杂度为 $O(n^2)$. 记插入数组 $\boldsymbol{A}$ 中元素的个数为 M，不难得出第 2 阶段循环语句的时间复杂度为 $O(M\times n)$. 由于每次插入中间节点 A_k 时都要修改可达数组 $\boldsymbol{R}^{(k)}$ 和可发数组 $\boldsymbol{C}^{(k)}$，故第 3 阶段循环语句的执行频度随着中间节点 A_k 的变化而变化. 考虑到第 3 阶段循环语句的 $i-1$ 和 $j-1$ 分别表示中间节点 A_k 的入度和出度. 为此，记 α_i 和 β_i 分别表示顶点 v_i 的可达率和可发率，则顶点 v_i 的入度和出度分别为 $\alpha_i n$ 和 $\beta_i n$，$\overline{\alpha}$ 和 $\overline{\beta}$ 分别表示每个顶点的平均可达率和平均可发率，则每个顶点的平均入度和平均出度分别为 $\overline{\alpha}n$ 和 $\overline{\beta}n$，于是，第 3 阶段循环语句的时间复杂度为 $O(M\times\overline{\alpha}n\times\overline{\beta}n)$.

综上所述，算法 2 的时间复杂度为 $O(n^2+M\times\overline{\alpha}n\times\overline{\beta}n)$.

空间复杂度方面. 由算法 1 可知，Floyd 算法的空间复杂度为 $O(n^2)$，算法 2 较 Floyd 算法多引入了插入数组、可达数组和可发数组，但它们的空间复杂度都为 $O(n)$，故算法 2 的空间复杂度仍为 $O(n^2)$.

2　算例分析

下面以图 1 给出的赋权有向图为例，应用算法 2 求解任意两点间的最短路径.

根据前面的分析，能够最小化两点间最短路径的中间节点为 v_2，v_4，故插入数组 $\boldsymbol{A}=[v_2,v_4]$. 取中间节点 v_2，v_2 的可达点和可发点分别为 $\{v_1\}$ 和 $\{v_3,v_4\}$，则可达数组和可发数组分别为 $\boldsymbol{R}=[v_1]$ 和 $\boldsymbol{C}=[v_3,v_4]$. 由于 $d_{12}+d_{23}=8<d_{13}=\infty$，$d_{12}+d_{24}=3<d_{13}=4$，修改 $d_{13}=8$，$d_{14}=3$，此时距离矩阵由 $\boldsymbol{D}^{(0)}$ 修改为 $\boldsymbol{D}^{(1)}$.

取中间节点 v_4，v_4 的可达点和可发点分别为 $\{v_1,v_2\}$ 和 $\{v_3\}$，则可达数组和可发数组分别为 $\boldsymbol{R}=[v_1,v_2]$ 和 $\boldsymbol{C}=[v_3]$. 由于 $d_{14}+d_{43}=6<d_{13}=8$，$d_{24}+d_{43}=5<d_{23}=7$，

修改 $d_{13}=6$，$d_{23}=5$，此时距离矩阵由 $\boldsymbol{D}^{(1)}$ 修改为 $\boldsymbol{D}^{(2)}$，变化过程如下:

$$\boldsymbol{D}^{(1)}=\begin{bmatrix}0&1&8&3&\infty\\ \infty&0&7&2&\infty\\ \infty&\infty&0&\infty&\infty\\ \infty&\infty&3&0&\infty\\ \infty&\infty&\infty&\infty&0\end{bmatrix}\rightarrow$$

$$\boldsymbol{D}^{(2)}=\begin{bmatrix}0&1&6&3&\infty\\ \infty&0&5&2&\infty\\ \infty&\infty&0&\infty&\infty\\ \infty&\infty&3&0&\infty\\ \infty&\infty&\infty&\infty&0\end{bmatrix}.$$

至此，便得任意两点间最短路径的距离矩阵 $\boldsymbol{D}^{(2)}$. 在图 1 中，搜索中间节点 v_2，v_4 执行了 5×5 次，搜索可达数组和可发数组执行了 $5\times2\times2$ 次，修改距离矩阵执行了 2×2 次，因此算法 2 总的语句执行次数为 49 次，而 Floyd 算法总的语句执行次数为 5^3 次. 此例中，算法 2 的执行速度是算法 1 执行速度的 2 倍多.

3　试验

硬件环境为 Pentium(R) 4 2.5 GHZ，512 MB 内存，80 GB 硬盘，操作系统为 Microsoft Windows XP SP3，开发工具为 MATLAB 7.0，比较算法 2 与算法 1 在求任意两点间最短路径的性能差别. 试验通过随机函数生成随机有向图，顶点数为 $n=1\,000$，每个顶点的平均边数从 1 条至 $(n-1)$ 条，共测试 200 次，取其平均值，试验结果见表 1.

表 1　Floyd 动态优化算法与 Floyd 算法的测试时间比较

序号	边数	算法 1 时间/s	算法 2 时间/s	算法 1 时间/算法 2 时间
1	1 000	404.450 0	0.128 3	3 152.38
2	2 000	354.767 2	0.464 2	764.26
3	3 000	284.204 5	1.484 2	191.49
4	4 000	239.453 1	4.609 2	51.95
5	5 000	203.092 3	8.823 6	23.02
6	6 000	179.965 6	20.984 4	8.58
7	7 000	161.023 6	38.579 8	4.17
8	8 000	148.770 4	63.957 6	2.33
9	9 000	137.881 2	88.036 0	1.57
10	10 000	129.873 6	111.715 7	1.16
11	20 000	87.634 3	128.159 3	0.68
12	30 000	73.106 2	134.687 3	0.54
13	40 000	67.054 7	136.306 5	0.49
14	50 000	62.821 8	138.492 1	0.45
15	60 000	59.393 6	139.650 2	0.43
16	70 000	57.690 8	140.390 7	0.41
17	80 000	56.059 5	140.879 5	0.40
18	90 000	54.868 6	141.914 0	0.39
19	100 000	53.645 4	141.748 4	0.38
20	200 000	49.228 3	143.734 3	0.34
21	300 000	47.704 6	143.665 6	0.33
22	400 000	46.953 4	144.381 1	0.33
23	500 000	46.496 9	143.650 0	0.32
24	600 000	46.178 2	144.537 3	0.32
25	700 000	45.992 2	144.006 1	0.32
26	800 000	45.754 7	144.793 7	0.32

续表 1

序号	边数	算法 1 时间/s	算法 2 时间/s	算法 1 时间/算法 2 时间
27	900 000	45.590 6	143.812 5	0.32
28	950 000	45.587 4	144.390 6	0.32
29	990 000	45.662 4	144.382 7	0.32
30	999 000	45.592 2	144.301 6	0.32

通过以上数据可以看出,在边数相对较少的稀疏图中,算法 2 速度比算法 1 有较大的优势;随着边数的增加,算法 1 的运行时间逐渐减少,算法 2 的运行时间逐渐增加,当图中每个顶点有 10 条左右边时,算法 2 与算法 1 的速度相当;当图中每个顶点有 100 条及更多边时,算法 1 的速度比算法 2 的要快,且稳定在 3 倍左右. 这主要是当边数较多时,顶点间的最短路径修改较少,算法 1 执行次数减少,算法 1 的运行时间相应也会减少;对算法 2 而言,当边数较多时,插入数组、可达数组以及可发数组的元素逐渐增加,算法 2 的执行次数也要增加,此时算法 2 的运行时间比算法 1 不再具有优势;另一方面,计算插入数组,修改可达数组和可发数组增加了程序的运行时间,因此算法 1 的速度比算法 2 的更快. 特别地,在完全有向图中,新算法的运算时间大约是 Floyd 算法的 3 倍.

4 结论

针对稀疏图,本文提出了 1 种新的求取任意两点间最短路径的算法: Floyd 动态优化算法. 该算法能够根据有向图的边数动态决定最短路径的求解时间. 理论分析和实验表明,在稀疏图中,新算法比 Floyd 算法在速度上有较大的提高,随着图中边数的增加,新算法的速度优势慢慢减弱,且最终新算法的速度比 Floyd 算法要慢一些,但它们的数量级始终相同,也没有本质的差异.

5 参考文献

[1] 严蔚敏,吴伟民. 数据结构 [M]. 北京: 清华大学出版社,1997.

[2] 张德全,吴果林,刘登峰. 最短路问题的 Floyd 加速算法与优化 [J]. 计算机工程与应用,2009,45(17):41-43.

[3] 张德全,吴果林. 最短路问题的 Floyd 优化 [J]. 许昌学院学报,2009,28(2):10-13.

[4] 李洪波,王茂波. Floyd 最短路径算法的动态优化 [J]. 计算机工程与应用,2006,42(34):60-63.

[5] 徐小玲,彭京,石葆梅,等. 一种基于边序列的任意两点间最短路径算法 [J]. 计算机工程与应用,2005,41(29):88-90.

[6] 刘韵,何建农. 基于交通网络最短路径搜索的改进算法 [J]. 计算机工程与应用,2007,43(14):220-222.

[7] 程理民,吴江,张玉林. 运筹学模型与方法教程 [M]. 北京: 清华大学出版社,2000.

[8] 杨大地,张春涛. 均匀两点交叉遗传算法 [J]. 重庆师范大学学报: 自然科学版,2004,21(1):26-29.

The Floyd Dynamic Optimization Algorithm of the Sparse Network

WU Guo-lin[1], JIN Zhen[2], DENG Xiao-fang[3]

(1. Department of Information Engineering, Guilin University of Aerospace Techology, Guilin Guangxi 541004, China;
2. Department of Science, Nanchang Institute of Technology, Nanchang Jiangxi 330099, China;
3. Software School, Jiangxi Normal University, Nanchang Jiangxi 330022, China)

Abstract: By searching for Floyd algorithm, a new method between any two nodes of the shortest path algorithm is proposed, which is Floyd dynamic optimization algorithm. By introducing insert array, reachable array and starting-off array, the algorithm changes automatically path nodes, eliminates some useless nodes and minimizes the number of execution statements before solving the shortest path. The algorithm analysis shows that the new algorithm has more performance than Floyd algorithm in sparse network.

Key words: shortest path; Floyd algorithm; insert array; reachable array; starting-off array

(责任编辑: 曾剑锋)