文章编号：2095－0411（2011）04－0045－05

稀疏网络的一个最短路算法及其实现*

李 博[1]，李 宁[2]，康慧燕[1]，元春梅[1]
（1. 常州大学 数理学院，江苏 常州 213164；2. 常州大学 信息科学与工程学院，江苏 常州 213164）

摘要：最短路算法在交通，通信等领域有非常重要的应用，许多网络问题都可以归结为一个最短路问题. Dijkstra 最短路算法是一个非常有效的算法，在计算网络中某一个顶点到其他各顶点的最短路时，如果引入 Fibonacci 堆，则 Dijkstra 算法运行所需要的加法及比较次数大致为 $O(m+n\log n)$，其中，m，n 分别为网络的边数和顶点数. 但由于在算法执行过程中，对 Fibonacci 堆的操作也有一定的代价。本文根据大型稀疏网络的特点，对 Dijkstra 最短路算法提出了一些非常简单的，但是非常有用的改进，并由此得到一个针对大型稀疏网络的 Dijkstra 最短路算法，该算法不需要构造 Fibonacci 堆，并且算法在运行时也只需要加法与比较，其所需要加法和比较的次数为 $O(m+n\log(n!))$，其中 D 为网络中与顶点相关联边数的最大值. 对于大型稀疏网络，如公路交通网络，D 通常比较小，因此，所给算法对这类网络是非常有效的.

关键词：Dijkstra 最短路算法；大型稀疏网络；Fibonacci 堆
中图分类号：O 157.6 **文献标识码**：A

A Shortest Path Algorithm for Large Sparse Network and Its Implement

LI Bo[1]，LI Ning[2]，KANG Hui－yan[1]，YUAN Chun－mei[1]
（1. School of Mathematics and Physics，Changzhou University，Changzhou 213164，China；2. School of Information Science and Engineering，Changzhou University，Changzhou 213164，China）

Abstract：Shortest path algorithm has many important applications in transportation，communications，etc. Many problems arising from such networks may come down to a shortest path problems. On a network with nonnegative－length edges，Dijkstra's shortest path algorithm computes single－source shortest path in $O(m+n\log n)$ time. The time bound assumes that a Fibonacci heap is used during the implementation of Dijkstra's algorithm. As the process of building heaps needs a little complex work，it makes the algorithm uneasy to be used. In this paper，we make use of the features of the large sparse network，make some very simple，but useful，changes in the original Dijkstra algorithm and obtain a new modified Dijkstra's shortest path algorithm for large sparse network. The new algorithm avoids the process of building heap and runs in $O(m+n\log(n!))$ time. Here m，n and D are the number of edges，vertices and the maximal number of edges incident with vertex，respectively. Thus，the new algorithm is very competitive for those sparse networks especially in road traffic networks in which D is often a small number.
key words：Dijkstra's shortest path algorithm；large sparse network；Fibonacci heap

* **收稿日期**：2011—06—22
作者简介：李博（1978—），女，辽宁抚顺人，讲师。

1 Dijkstra 最短路算法简介

假设用一个赋权的有向图 $G=(V, E, \Phi)$ 来表示网络，其中 V 是网络中所有顶点所成非空点集，E 为所有边所成非空集，Φ 是从集合 E 到非负实数的一个映射。设图 $G=(V, E, \Phi)$ 的边数为 m 个，顶点数为 n 个，下面用字母 u，v，u_i 等表示网络图中的顶点，对起点为 u 终点为 v 的边 e 用 $e=u\,v$ 表示，相应的长度用 $\Phi(uv)$ 表示。为了方便表示，用 neighbour（u）表示图中与顶点 u 相连的所有顶点所成集合。对于任意给定的起点 s，Dijkstra 最短路算法[1]给出了从起点 s 出发到图中其他各点的最短路。但关于 Dijkstra 最短路算法的研究一直得到研究领域的长期重视[2-5]。下面给出的算法 1 是 Dijkstra 最短路算法的一个改进算法[2]。

算法 1

改进的 Dijkstra 最短路算法

step 1：令 $l(s)=0$；对所有的 $v\neq s$，$l(v)=\infty$；$S=\{s\}$，$\bar{S}=V-S$；$u_0=s$，$i=0$.

step 2：更新 $l(v)$ 以及集合 S，$\bar{S}$.

（1）对每个 $v\in\bar{S}\cap$neighbor（u_i），计算

$$l(v)=\min\{l(v), l(u_i)+\Phi(u_i\,v)\}$$

（2）计算 $\min_{v\in\bar{S}}\{l(v)\}$，并设在顶点 u_{i+1} 达到极小。

（3）令 $S:=S\cup\{u_{i+1}\}$，$\bar{S}:=\bar{S}-\{u_{i+1}\}$

step 3：如果 $i=n-1$，停止；否则，$i:=i+1$，转 step 2。

上述算法的瓶颈在于计算 $\min_{v\in\bar{S}}\{l(v)\}$。一个最原始的做法是将 $\bar{S}$ 中的每个顶点对应的 $l(v)$ 与其他顶点的比较，这样需要大概 $O(n^2)$ 次比较。如果引入数据结构 Fibonacci 堆，文章［3］表明：相应的最短路算法的运算量为 $O(m+n\log n)$。下面我们根据一些特殊网络，如公路交通网络，稀疏的特点，给出一个针对稀疏网络的 Dijkstra 最短路算法，算法简单，容易实现。该算法在运行时同样仅需要加法和比较，且加法和比较的次数为 $O(m+D\log(n!))$，其中 D 为网络中与顶点相关联边数的最大值。新的算法不需要引入 Fibonacci 堆。

2 Dijkstra 最短路算法及改进

在这一节里，为了比较容易掌握上述算法思想，先给出一个针对稀疏网络的 Dijkstra 型最短路算法[6]，然后给出其改进算法，最后给出算法的一些实现技巧和分析结果。

算法 2

针对稀疏网络的 Dijkstra 型最短路算法

step 1：令 $l(s)=0$；对所有的 $v\neq s$，$l(v)=\infty$；

将数表 l 按升序排列（仅仅需要将 $l(s)$ 放在数表 l 的第一个位子上即可）。

令 $S=\{s\}$，$\bar{S}=V-S$；$\mathrm{low}_0=2$；$u_0=s$，$i=0$

step 2：更新 $l(v)$ 以及集合 S，$\bar{S}$，并将数表 l 排序。

（1）for $v\in\bar{S}\cap$neighbor（u_i），do

if $l(v)>l(u_i)+\phi(u_i\,v)$

$l(v)=l(u_i)+\phi(u_i\,v)$

1）用 $\mathrm{high}_i{}^v$ 记录 $l(v)$ 在数表 l 中的位子。

2）将 $l(v)$ 插入到数表 l 的位子 low_i 和 $\mathrm{high}_i{}^v$ 之间，使数表 l 按升序排列。

end if

end for

（2）令 u_{i+1} 为数表 l 位子 low_i 上元素所对应的顶点。

（3）令 $S:=S\cup\{u_{i+1}\}$，$\bar{S}:=\bar{S}-\{u_{i+1}\}$，$\mathrm{low}_{i+1}=\mathrm{low}_i+1$。

step 3：如果 $i=n-1$，停止；否则，$i:=i+1$，转 step 2。

容易看出，算法 2 与算法 1 的区别在于计算 $\min_{v\in\bar{S}}\{l(v)\}$ 的方法不同，因此，算法 2 在算法 1 的条件下，同样能够获得从顶点 s 到其他各点的最短路。

另外，在应用上述算法 2 处理大型稀疏网络最短路问题时，发现数表 l 中的元素在算法开始的一段运行时间内通常大部分元素保持不变仍是无穷，因此，在算法 2 对数表 l 进行排序时，实际上有许多元素的位子不需要调整，每一步仅仅需要在一个相对小些的范围内把 $l(v)$ 重新插入数表 l 中，以保证数表 l 能够按升序排列。据此考虑，将算法 2 进行改进，并由此得到算法 3。

算法 3

针对稀疏网络最短路算法的改进

step 1：令 $l(s)=0$；对所有的 $v\neq s$，$l(v)=\infty$，

令 $S=\{s\}$，$\bar{S}=V-S$，

$N=$Null 为空集，

num _ $N=0$（num _ N 用来记录集合 N 中的元素个数）

$low_0=1$，$u_0=s$，$i=0$

step 2：更新 $l(v)$ 以及集合 S，$\bar{S}$，并将数表 l 排序。

(1) for $v\in\bar{S}\cap$neighbor (u_i), do

if $l(v)>l(u_i)+\phi(u_i v)$

$l(v)=l(u_i)+\phi(u_i v)$

if $v\notin N$

$N:=N\cup\{v\}$,

num _ N = num _ $N+1$

$high_i^v$ = num _ N

else

$high_i^v$ 等于 $l(v)$ 在数表 l 中所在位子的大小，把数表 $\{l(u), u\in N\}$ 按升序排列后的数表记为 l 。

end if

将 $l(v)$ 插入到数表 l 的位子 low_i 和 $high_i^v$ 之间，使数表 l 按升序排列。

end if

end for

(2) 令 u_{i+1} 为数表 l 位子 low_i 上元素所对应的顶点。

(3) 令 $S:=S\cup\{u_{i+1}\}$，$\bar{S}:=\bar{S}-\{u_{i+1}\}$，$low_{i+1}=low_i+1$。

step 3：如果 $i=n-1$，停止；否则，$i:=i+1$，转 step 2。

需要再提一下的是，算法 3 仅仅是将算法 2 中的排序过程简化了。与算法 2 不同的是，算法 3 把不需要调整排列次序的元素没有放到数表 l 中，从而降低了排序的工作量，数值试验表明，算法 3 所需的排序工作量大致是算法 2 的一半。

为了能进一步说明算法 3 是一个对于稀疏网络有效的算法，先给出上述算法 3 实现时用到的技巧，然后再给出算法的一些分析结果。

3 算法实现技巧与分析

3.1 算法的实现技巧

算法 3 的实现关键是要适当处理好其 step 2 中插入一步，为此，引入一个 n 维向量 step _ size，其分量 step _ size (i) 由下面的过程 1 给出。

过程 1

step 1：令 $i=1$，step _ size $(i)=0$，$j=0$。

step 2：While $i<n-1$

$j:=j+1$

$i:=i+1$

step _ size $(i)=j$

$i:=i+1$

step _ size $(i)=j$

end while

step 3：if $i=n-1$

$j:=j+1$

$i:=i+1$

step _ size $(i)=j$

end if

若 $n=8$，则由上述过程容易得到：step _ size $=(0, 1, 1, 2, 2, 3, 3, 4)$。

注意到，对于不同的两个正整数 n_1，n_2，若 $n_1<n_2$，则它们所对应的两个向量 step _ size 的前面 n_1 个分量对应相等；下面将会看到：算法 3 执行时，所引入的向量 step _ size 并不改变。因此，从这个意思上看，完全可以在算法运行之前预先给出向量 step _ size，并保证其维数不小于所考虑的网络的顶点数 n，这样可以达到在计算过程中节省计算 step _ size 的时间。

在得到向量 step _ size 的基础上，可以通过下述过程 2 实现算法 3 中其 step 2 中插入一步，即将 $l(v)$ 重新插入到数表 l 的位子 low_i 和 $high_i^v$ 之间，使数表 l 按升序排列。

过程 2

step 1：初始化。令 bottom $=low_i$，top $=high_i^v-1$；向量 entry $\in R^n$，且

for $i=low_i$ to $high_i^v$

entry $(i)=l(i)$,

end for

step 2：寻找 $l(v)$ 在数表 l 中的合适位子。

if entry $(high_i^v)\geqslant$entry (top)

转到 step 4

else

length = top − bottom

while length > 1

mid: = bottom + step _ size (length)

if $l(v)<$entry (mid)

top: = mid

else

bottom: = mid

end if

length：＝top－bottom
end while
if l（v）＜entry（bottom）
location＝bottom
else
location＝top
end if
end if

step 3：将 l（v）插入到数表的位子 location 处（为了保证数表中插入 l（v）的能空出来，必须先将数表 l 中的一些元素下移）。

for j＝high$_i{}^v$ to location＋1
entry（j）：＝entry（j－1）
end for
entry（location）＝l（v）
for i＝location to high$_i{}^v$
l（i）＝entry（i）
end for

step 4：输出已经排序的数表 l

值得说明的是，过程 2 用了类似二分搜索的技巧来确定 l（v）的插入位置。。从过程 2 中，可以看到引入向量 step _ size 的目的在于避免通常的二分搜索中的除法运算，并保证过程 2 需要的比较次数与通常的二分搜索几乎一样。

3.2 算法分析

在给出上述算法实现技巧的基础上，可以给出与算法 3 有关的几个分析结果。

命题 1 利用过程 1 和过程 2 将 l（v）插入到数表 l 的位子 low$_i$ 和 high$_i{}^v$ 之间，使数表 l 按升序排列所需要的加法次数和比较次数为 O（log（high$_i{}^v$－low$_i$））。

证明：

由过程 1 知道，过程 2 中的变量 mid＝bottom＋step _ size（top－bottom）差不多是以 bottom 为起始位子，top 为终点位子的数表的中间位子，因此，通过一次比较就可以将 l（v）的位子范围缩小几乎一半。类似二分搜索方法，得到将 l（v）插入到数表 l 的位子 low$_i$ 和 high$_i{}^v$ 之间，使数表 l 按升序排列所需要的加法次数和比较次数为 O（log（high$_i{}^v$－low$_i$））。

命题 2 假设 D 为网络中与顶点相关联边数的最大值，那么算法 3 或 2 在利用过程 1 和过程 2 的基础上求得从顶点出发到网络其它各顶点的最短路所需加法和比较的次数为 O（m＋Dlog（n!））。

证明：

由命题 1 及算法 2 和算法 3 容易得到从顶点 s 出发到网络其他各顶点的最短路所需加法和比较的次数为

$$O\left(m+\sum_{i=0}^{n-1}\sum_{v\in\bar{S}\cap\text{neighbor}(u_i)}O\left(\log\left(\text{high}_i{}^v-\text{low}_i\right)\right)\right) \quad (1)$$

利用

$$\log\left(\text{high}_i{}^v-\text{low}_i\right)\leqslant\log\left(n-i\right) \quad (2)$$

可以得到

$$\sum_{i=0}^{n-1}\sum_{v\in\bar{S}\cap\text{neighbor}(u_i)}\log\left(\text{high}_i{}^v-\text{low}_i\right)\leqslant$$

$$\sum_{i=0}^{n-1}D\max_{v\in\bar{S}\cap\text{neighbor}(u_i)}\left\{\log\left(\text{high}_i{}^v-\text{low}_i\right)\right\}\leqslant$$

$$\sum_{i=0}^{n-1}D\log\left(n-i\right)=D\log\left(n!\right) \quad (3)$$

将（3）式代入（1）式即可得到命题的结论。

值得说明的是：由于算法 3 是在算法 2 的基础上，将不需要调整排列次序的元素没有放到其数表 l 中，因此，对于算法 3 而言，上述证明过程中用到的不等式（2）是很保守的，对于大型问题，算法 3 在执行过程中多数情形满足 log（high$_i{}^v$－low$_i$）远小于 log（n－i）。

4 数值试验

考虑到算法的运行时间不仅仅与计算机的性能有关，而且与所使用的计算机语言有关，因此，在数值试验报告中不准备报告算法运行的 CPU 时间，而根据算法工作量的特点，给出算法运行过程中所需要的加法次数与比较次数的和 T 与数 m＋Dlog（n!）的比值 Ratio。即

Ratio＝T/（m＋D log（n!））

其中，T 为算法运行过程中所需要的加法次数与比较次数的和（如前所考虑，这里不包括产生向量 step _ size 的计算量）。

显然 Ratio 的大小是一个直接反映算法好坏的重要指标。

例 1 设所考虑的网络的顶点有 n 个，分别以顶点 1，2，…，n 表示。对任意的 $1\leqslant i<n$，顶点 i 与顶点 i＋1 有边相连；同时顶点 i 与顶点 $i+j_i$ 有边相连，其中 j_i 在 2，3，…，λ 中随机取值且满足 $i+j_i\leqslant n$。在这个试验中，取 s＝1，λ＝30，所有边的长度在区间［0，500］上均匀地随机取值，D 的大小由所产生的网络确定。下面就 n＝10 000，11

000，…，21 000 分别进行试验，表 1 给出了试验报告。

表 1　边数较少情形的网络试验报告

Table 1　Network test report of fewer edges

网络的顶点数 n	网络的边数 m	Ratio
10 000	19 986	0.384 8
11 000	21 985	0.377 3
12 000	23 989	0.372 0
13 000	25 985	0.337 5
14 000	27 985	0.327 3
15 000	29 982	0.366 0
16 000	31 978	0.356 7
17 000	33 984	0.320 1
18 000	35 983	0.359 8
19 000	37 983	0.322 7
20 000	39 984	0.353 1
21 000	41 985	0.344 4

例 2　与例 1 类似，仍设所考虑的网络的顶点有 n 个，分别以顶点 1，2，…，n 表示。对任意的 $1\leqslant i<n$，顶点 i 与顶点 $i+1$ 有边相连；同时顶点 i 与顶点 $i+j_{i1}$ 有边相连，其中 j_{i1} 在 2，3，…，λ_1 中随机取值且满足 $i+j_{i1}\leqslant n$，并且顶点 i 与顶点 $i+j_{i1}+j_{i2}$ 也有边相连，其中 j_{i2} 在 1，2，…，λ_2 中随机取值且满足 $i+j_{i1}+j_{i2}\leqslant n$。在这个试验中，取 $s=1$，$\lambda_1=30$，$\lambda_2=20$ 所有边的长度在区间 [0，500] 上均匀地随机取值，D 的大小由所产生的网络确定。下面就 $n=10\ 000$，$11\ 000$，…，$21\ 000$ 分别进行试验，表 2 给出了试验报告。

表 2　边数较多情形的网络试验报告

Table 2　Network test report of more edges

网络的顶点数 n	网络的边数 m	Ratio
10 000	29 964	0.458 5
11 000	32 956	0.416 3
12 000	35 952	0.416 0
13 000	38 955	0.386 7
14 000	41 961	0.405 1
15 000	44 961	0.399 8
16 000	47 955	0.432 1
17 000	50 958	0.392 9
18 000	53 956	0.396 6
19 000	56 963	0.388 2
20 000	59 957	0.393 2
21 000	62 958	0.364 6

此外，进行多次模拟试验表明，算法 3 对于大型稀疏网络（D 不太大的情况）都有比较好的计算效果，能够比较好的处理大型稀疏网络的最短路问题。因此，算法 3 在一些应用问题上，如公路交通问题[7]等，有比较好的针对性。

参考文献：

[1] Dijkstra E W. A note on two problems in connection with graphs [J]. Numer Math，1959 (1)：269－271.

[2] Haldar S. An 'all pairs shortest paths' distributed algorithm using messages [J]. Journal of Algorithms，1997，24：20－36.

[3] Fredman M L，Tarjan R E. Fibonacci Heaps and their uses in improved network optimization algorithm [J]. Journal of the ACM，1987，34：596－616.

[4] 徐立华. 求解最短路问题的一种计算机算法 [J]. 系统工程，1989，7 (5)：46－51.

[5] Saunders S，Takaoka T. Improved Shortest Path Algorithms for Nearly Acyclic Graphs [J]. Electronic Notes in Theoretical Computer Science，2001，42：1－17.

[6] Xu M H，Liu Y Q，Huang Q L，et al. An improved Dijkstra's shortest path algorithm for sparse network [J]. Applied Mathematics and Computation，2007，185：247－254.

[7] Xu M H，William H K Lam，Shao H，et al. A heuristic algorithm for network equilibration [J]. Applied Mathematics and Computation，2006，174：430－446.