第 29 卷 第 2 期　西南交通大学学报　Vol. 29 No. 2
1994 年 4 月　JOURNAL OF SOUTHWEST JIAOTONG UNIVERSITY　Apr. 1994

关于最短路径的 SPFA 快速算法

段凡丁
(西南交通大学 计算中心 成都 610031)

【摘 要】 本文提出了关于最短路径问题的一种新的快速算法——SPFA(Shortest Path Faster Algorithm)算法. SPFA 算法采用动态优化逼近的方法,用邻接表作为有向图的存储结构,用了一个先进先出的队列 Queue 来作为待优化点的存储池。算法的时间复杂性为 $O(e)$,在绝大多数情况下,图的边数 e 和顶点数 n 的关系是 $e < n^2$,因此,SPFA 算法比经典的 Dijkstra 算法在时间复杂性方面更优越。

【关键词】 有向图;最短路径;算法;时间复杂性
【分类号】 TP312

在计算机网络、交通运输工程、通信工程等各种应用中,常常需要计算图中从某个源点到其余各顶点的最短路径或各顶点之间的最短路径。很多的优化问题在数学上讲,是相当于找寻一个图中的最短路径。因此,在图论中,最短路径算法比其它算法研究得更为透彻。到 1974 年为止,大约有 100 多篇论文和几十种算法已经提出来了[1]。关于单源最短路径的算法,目前最流行的经典算法是 Dijkstra 算法,它的时间复杂性是 $O(n^2)$[2]。1974 年,Wagner 采用桶分类得出一个 $O(e)$ 的算法,但必须是边的权值是小的整数才适用[3]。关于每一对顶点之间的最短路径算法,公认 Floyed 算法最佳,其时间复杂性为 $O(n^3)$[4]。本文对最短路径问题的这两个方面进行研究,提出了一种新的快速算法——SPFA,SPFA 算法的时间复杂性为 $O(e)$,当 $e \ll n^2$ 时,SPFA 算法表现出时间复杂性上的优越。SPFA 算法对边的权值没有特殊的限制,适合于任意有向图。

1 单源最短路径的 SPFA 算法

1.1 Dijkstra 算法分析

为了清楚起见和便于比较,现将 Dijkstra 算法简述如下:

(1) begin
(2) $S \leftarrow \{v_0\}$;
(3) $D[v_0] \leftarrow 0$;
(4) for each v in $V - \{v_0\}$ do $D[v] \leftarrow l(v_0, v)$;
(5) while $S \neq V$ do
　　begin

本文于 1993 年 9 月 20 收到。

(6) choose a vertex w in $V - S$ such that $D[w]$ is a minimum;

(7) add w to S;

(8) for each v in $V - S$ do

(9) $D[v] \leftarrow \mathrm{MIN}\ (D[v], D[w] + l(w,v))$;

end

(10) end

Dijkstra 算法采用了两个集合这样的数据结构来安排图的顶点。算法的主要思想是:从有向图的 V-S 集合中选取具有最小 $D[w]$ 的点 w,尔后把 w 点放入 S 集合中,那么 S 集合就是已经计算出具有最短路径的点的集合。然后再从 V-S 集合中将所有经过点 w 而与源点相通的 v 点的路径值 $D[v]$ 统统都作调整(如果存在 $D[v] > D[w] + l(w,v)$ 的话)。重复此过程直到所有的点全部进入 S 集合。

从以上分析可以看出,Dijkstra 算法属于探新法的一种,即每次都是从 V-S 集合中选一个具有最小 $D[w]$ 的点并放入 S 集合,进入了 S 集合的点的路径值就保证是该点的最短路径。

算法是一步一个脚印地向前搜索,即循环的每一步都能正确地算出从源点到 w 点的最短路径 $D[w]$,并且将所有的经过 w 点的 V-S 集合的点的路径值都作相应的优化。从时间上来分析,若有 n 个顶点的图,则第(6)句为 $O(n)$,第(8)句为 $O(n)$,这两项是并列在 while 循环里的,而 while 循环也是 $O(n)$,故总的时间为 $O(2n^2)$,简算为 $O(n^2)$ 数量级。

1.2 SPFA 算法描述

输入 设 L 是用来表示有向图 $G = (V,E)$ 的邻接表,邻接表元素 l 是有向图各边的权值,输入各边的权值 $l(v,k)$ 建立邻接表 L。

输出 设 D 数组是记录当前从源点到其余各点的最短路径的值,初始化时 D 数组的每个元素都为最大值,经过 SPFA 算法 D 数组输出各点的最短路径值。

方法 SPFA 算法采用图的存储结构是邻接表,方法是动态优化逼近法。算法中设立了一个先进先出的队列 Queue 用来保存待优化的顶点,优化时从此队列里顺序取出一个点 w,并且用 w 点的当前路径 $D[w]$ 去优化调整其它各点的路径值 $D[j]$,若有调整,即 $D[j]$ 的值改小了,就将 j 点放入 Queue 队列以待继续进一步优化。反复从 Queue 队列里取出点来对当前最短路径进行优化,直至队空不需要再优化为止,此时 D 数组里就保存了从源点到各点的最短路径值。SPFA 算法的形式算法描述及注释如下:

(1) begin

{算法开始}

(2) for each v in V do

begin

for each $k \in L[v]$ do read $(l(v,k))$;

{读入每条边的权值到邻接表}

QM $[v] := 0$;

{初始化每个顶点是否在队里的标志数组}

$D[v] := \mathrm{MAX}$;

```
{将最短路径数组初始化为最大值}
            end;
(3) Queue← v0;
    QM [v0] : = 1;
{源点 v0 入 Queue 队}
(4) D[v0] : =0;
{源点到源点本身的路径值赋值为零}
(5) while Queue not empty do
        begin
            w←Queue;
{从 Queue 队列里取出一个点 w}
            QM [w] : = 0;
{w 点出队后,其标志数组元素改为零,表示 w 点不在队列}
(6)                 for each j ∈ L[w] do
(7)                     if D[j] > D[w] + l(w,j) then
                        begin
(8)                         D[j] : = D[w] + l(w,j)
{判断经过 w 点到 j 点的路径是比原来的路径 D[j] 更短后,对 j 点的路径进行优化}
                            if QM [j] =0 then
                            begin
                                Queue ←j ;
                                QM [j] : = 1;
{当 j 点不在队列里,j 入队,并且将 QM [j] 标志置为 1 表示 j 已入队}
                            end
                        end
          end;
(9) for each v in V do
            begin
                write (D[v]);
            end;
{优化完成后,D 数组存放的就是从源点到各点的最短路径值,可以输出结果}
(10) end.
{算法结束}
```

2　SPFA 算法的正确性证明

对 SPFA 算法的正确性,要证明以下两个定理。

定理 1　SPFA 算法采用的动态优化方法,能够使得 D 数组的路径值变得越来越小,优化

过程是正确的。

证 算法中采用了一个 FIFO 的优化顺序队 Queue,用来存放待优化的点,算法中每一步都是从该队取出点来优化其它各点。现假定算法结束时计算的结果 D 不是最短路径,即从源点到某些点还存在着更短的路径,而算法规定:只要存在着比 $D[j]$ 更短的路径,即 $D[j] > D[w] + l(w,j)$,那么 $D[j]$ 就一定要被优化且 j 点就要入队,算法就不会结束(见算法第(6)句至第(9)句)。所以上述假定不成立,算法的执行会使 D 数组的值变得越来越小,逼进直至达到最短路径,优化过程正确。

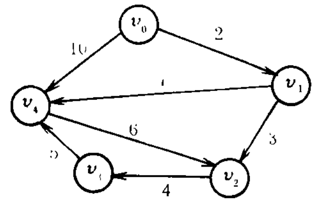
定理 2 SPFA 算法的优化过程是收敛的,不会形成无限循环,算法能够正常结束。

证 SPFA 算法尽管不是从队列里去挑选出具有最小路径的点来进行优化,而是直接从队列里取出队首的点来进行优化,因此整个优化过程不一定循环 n 次就能完成,有些点入队次数也可能不止一次,那么是否存在不收敛的情况呢?众所周知,在有向图 $G = (V,E)$ 中,各边的权值都是已知的并且在整个算法中是不会改变的。可以证明,只要边的权值保持不变,那么优化过程就一定会收敛。

证明 在任意一个具有循环结构的算法或程序中,所谓不收敛就是死循环。在 SPFA 算法中,循环控制是由队列 Queue 是否为空为条件的,若队列不为空,继续循环;若队列为空,循环终止,算法结束。在 SPFA 算法的循环中,既有从队列取出点来进行优化的队减操作,也有被优化后的点入队的队增操作,则要注意,某任意顶点 j 入队是有先决条件的,亦即 $D[j] > D[w] + l(w,j)$,j 点入队的同时,$D[j]$ 也就被优化取值为较小的 $D[w] + l(w,j)$,这样使得各点的路径值 $D[j]$ 变得越来越小,直到最终优化成为最短路径,此时就不会再有任何点被继续优化的可能,因而就不会再有顶点入队,仅有队减操作直到队空循环终止,算法结束时 D 数组正好是各点的最短路径。(证毕)

这里通过一个例子来说明 SPFA 的执行情况以及验证算法的正确性。

设有一个有向图(见附图) $G = \{V,E\}$,其中,$V = \{v_0,v_1,v_2,v_3,v_4\}$,$E = \{<v_0,v_1>, <v_0,v_4>, <v_1,v_2>, <v_1,v_4>, <v_2,v_3>, <v_3,v_4>, <v_4,v_2>\} = \{2,10,3,7,4,5,6\}$,即画出附图。



附图 一个有向图实例

算法执行时各步的 Queue 队的值和 D 数组的值,由表 1 所示。

表 1 实例图 SPFA 算法执行的步骤及结果

初始		第一步		第二步		第三步		第四步		第五步	
Queue	D	Queue	D	Queue	D	Queue	D	Queue	D	Queue	D
v_0	0	v_1	0	v_4	0	v_2	0	v_3	0		0
	∞	v_4	2	v_2	2		2		2		2
	∞		∞		5		5		5		5
	∞		∞		∞		∞		9		9
	∞		10		9		9		9		9

算法执行到第五步后,队 Queue 空,算法结束。源点 v_0 到 v_1 的最短路径为 2,到 v_2 的最短路

径为 5,到 v_3 的最短路径为 9,到 v_4 的最短路径为 9,结果正确。

3 算法的时间复杂性分析

定理 3 对于任意的有向图 $G=(V,E)$,设边的总数为 e,顶点的总数为 n,SPFA 算法的时间复杂性为 $O(e)$。

证 SPFA 算法首先进行初始化,初始化主要是读入有向图的每一条边的权值,显然需用的时间为 O(e)。初始化后,SPFA 算法首先将源点入队,然后从队列里取出队首的一个顶点作为 w 点,这里没有像 Dijkstra 算法那样从 $V-S$ 集合中选一个具有最小 $D[w]$ 的 w 点,所以省去了选择所花费的时间,SPFA 算法的时间复杂性主要是由 while 循环所决定的。由于采用邻接表作为图的数据结构,第(6)句就是对一个点的表所连接的所有边进行优化,而每一个点的表长是与该点的出度有关。因为 n 个点的出度总和就是有向图的边数 e,那么对于一个点来说,其平均出度就是 e/n,所以第(6)句的执行时间平均为 $O(e/n)$。

设第(5)句 while 循环的次数为 m,即为顶点入队的次数,若平均每一个点入队一次,则 $m=n$;若平均每个点入队两次,则 $m=2n$。算法编程后实际运行试算情况表明,m 一般没有超过 $2n$。事实上,虽然顶点入队次数 m 是一个事先不易分析出的数,但它确是一个随图的不同而略有不同的常数,所谓常数,即与 e 无关,也与 n 无关,仅与边的权值分布有关,一旦图确定,各边的权值确定,源点确定,m 也就是一个确定的常数,所以,SPFA 算法的时间复杂性为

$$T=O(m\cdot\frac{e}{n})=O(\frac{m}{n}\cdot e)$$

令
$$K=\frac{m}{n}$$
则
$$T=O(k\cdot e)$$

因为 K 是一个常数,所以 SPFA 算法的时间复杂性为 $O(e)$。(证毕)。

对于一个边数为 e、顶点数为 n 的有向图,有:

$$e_{最少}=n-1$$

$$e_{最多}=(n-1)+(n-2)+\cdots+1=\sum_{i=1}^{n-1}i<n^2$$

在一般的应用中,总有 $e\ll n^2$,所以 SPFA 算法的时间复杂性 O(e)优于 Dijkstra 算法的时间复杂性 $O(n^2)$。SPFA 算法用 PASCAL 语言编程,在 IBM PC 机上通过,多次计算的结果表明,其时间复杂性是 $O(e)$ 数量级,即与图的边数 e 成比例,Dijkstra 算法用程序实现,其时间复杂性是 $O(n^2)$ 数量级,即与图的顶点数 n^2 成比例。当图的顶点越多,边较稀疏时,SPFA 算法比 Dijkstra 算法的改进效果是比较明显的。

在文献[2]中也已证明:对于单源最短路径问题的算法,其最小的时间复杂性代价是 $O(e)$。这是十分显然的,因为对任意一个单源路径的算法来说,要是它不"查完"全部边,那么这样的算法就不可能是正确的,所以不难相信,$O(e)$ 时间复杂性的单源最短路径算法应该是我们期望的最好的算法。

4 结束语

采用动态逼进优化的 SPFA 算法，对最短路径这一典型的问题，可以提高速度，使其时间复杂性由 $O(n^2)$ 降低成为 $O(e)$。对于要解决每一对顶点之间的最短路径问题，可以把 SPFA 算法推广，改变源点重复执行 n 次，这样可使得计算每一对顶点之间的最短路径的时间复杂性降低为 $O(n \cdot e)$，这比 Floyed 在 1962 年提出的 $O(n^3)$ 时间复杂性的算法要快。动态逼近优化方法，其收敛的速度随图及边的权值分布而略有不同，这跟 Quicksort 算法的分析有些类似，Quicksort 的时间复杂性也跟表的元素分布有关，但就其平均复杂性来说，Quicksort 为 $O(n \cdot \log_2 n)$。这里也有类似的情况，SPFA 算法的平均复杂性为 $O(e)$，是解决最短路径问题的一种好的算法。

参 考 文 献

1 吴文泷. 图论基础及应用. 北京：中国铁道出版社，1984：220—227

2 Aho A V, Hopcroft J E, Ullman J D. The design and analysis of computer algorithms. Addison-Wesley, 1976：207—222

3 Wagner R A. The shortest path algorithm for edge-sparse graphs. Dept of Systems and Information Science. Vanderbilt University. Nashville. Tennessee, 1974

4 严蔚敏，沈佩娟. 数据结构. 北京：国防工业出版社，1981：107—109

A Faster Algorithm for Shortest-Ptath——SPFA

Duan Fanding

(*Computer Center. Southwest Jiaotong University. Chengdu* 610031, *China*)

【Abstract】 This paper offers a new fast algorithm for shortest-path prolem——SPFA. The data structrue of SPFA algorithm uses adjacency list and a FIFO queue. Applying dynamic optimal approach, the time complexity of SPFA algorithm is $O(e)$, it is better than Dijkstra's typical algorithm when $e \ll n^2$. No particular limitation conditions are needed. So the SPFA algorithm can be adapted for all directed graphs.

【Keywords】 directed graph; shortest-path; algorithm; time complexity