经过指定节点集的最短路径

王超民、党妮、李晓杰

2016 年 5 月 1 日

1 问题定义

给定一个带权重的有向图$G=(V,E)$，V为顶点集，E为有向边集，每一条有向边均有一个权重。对于给定的顶点s、t，以及V的子集V_s，寻找从s到t的不成环有向路径P，使得P经过V_s中所有的顶点(对经过V_s中节点的顺序不做要求)。

若不存在这样的有向路径P，则输出无解，程序运行时间越短，则视为结果越优；若存在这样的有向路径P，则输出所得到的路径，路径的权重越小，则视为结果越优，在输出路径权重一样的前提下，程序运行时间越短，则视为结果越优。

说明：

1) 图中所有权重均为$[1,20]$内的整数；

2) 任一有向边的起点不等于终点；

3) 连接顶点A至顶点B的有向边可能超过一条，其权重可能一样，也可能不一样；

4) 该有向图的顶点不会超过600个，每个顶点出度(以该点为起点的有向边的数量)不超过8；

5) V_s中元素个数不超过50；

6) 从s到t的不成环有向路径P是指，P为由一系列有向边组成的从s至t的有向连通路径，且不允许重复经过任一节点；

7) 路径的权重是指所有组成该路径的所有有向边的权重之和。

2 算法描述

给定一个有向图$G=(V,A)$，其中节点集$V=\{v_1,\ldots,v_n\}$，边集$\{a_1,\ldots,a_m\}$。对于任意一条有向边$a_k=(v_i,vj),v_i,v_j\in V$，$v_i$为起点，$v_j$为终点。记$p=\langle s\equiv v_1,v_2,\ldots,v_k\equiv t\rangle$为从起点$s$出发至终点$t$的一条路径，其中$(v_i,vi+1)\in A,\forall i\in\{1,\ldots,k-1\}$，$k$是路径$p$中的节点数；记$V_p$为路径$p$的节点集，$A_p$为路径$p$的边集，$A_p=\cup_{\forall i\in\{1....,k-1\}}(v_i,v_{i+1})$。边$(v_i,v_j)\in A$的权重由函数$w(v_i,v_j)$给出，这里假设边的权重都为正数。一条路径的权重$D_p$为组成这条路径的所有有向边权重之和，$D_p=\sum_{v_i,v_j\in A_p}w(v_i,w_j)$。如果对于给定的两个节点之间的路径$p$不存在，则记为空集($\varnothing$)，对应的权重为无穷大。

2.1 SK66算法

Saksena和Kumar学者在1966年提出了一个基于最优准则的算法(SK66)。该算法通过选择具有最小路径权重的子路径串联而成的路径，来试图求解经过指定点集的最短路径，该算法的时间复杂度为$\mathcal{O}\left(\left|V_S+2\right|^2|A|\,log_2|V|+|V_S||V|\,log_2|V|\right)$ (在基于二叉堆实现的Dijkstra's算法的前提下)。

SK66算法的初始化步骤：

1) 计算属于V_S的所有节点对之间的最短路径，以及起点s到V_s中所有节点的最短路径；

2) 计算V_s中所有节点到终点t的最短路径。

记$D(v_i,v_l)$ 为v_i到v_l的最短路径权重，其中$v_i \in V_s \cup \{s\}$，$v_l \in V_s$；$f_{v_i}^0$为节点$v_i \in V_s$到终点t的最短路径权重。

那么算法可以不断地迭代计算：

$$f_{v_i}^{\eta} = min_{v_l \in V_s}\left[D(v_i,v_l)+f_{v_l}^{\eta-1}\right], \qquad v_l \neq v_i \tag{1}$$

$\eta = 1,2\ldots,|V_s|-1$，每次迭代意味着最短子路径中至少要新加入一个特定节点。换而言之，从v_i出发，经过一个特定节点v_l，到达终点t的最短子路径，只有在其包含了至少η个特定节点时(不包括v_i)，它的路径权重才为$D(v_i,v_l)+f_{v_l}^{\eta-1}$。这一点在算法的每一步中都需要验证。

算法的最后一步是计算

$$f_{v_0}^{|V_s|} = min_{v_l \in V_s}\left[D(s,v_l)+f_{v_l}^{|V_s|-1}\right] \tag{2}$$

以及对应的路径。SK66算法的伪代码如下：

SK66-Algorithm(G)

```
[] 1  run Dijkstra's algorithm to calculate D(v_i, v_j),  v_i, vj ∈ s ∪ V_s ∪ t
   2  let F = (f_{η,i}) be a new |V_s| × |V_s| matrix
   3  for i = 1 to |V_s|
   4        do F_{1,i} ← D(v_i, v_j)
   5  for η = 2 to |V_s|
   6        do for l = 1 to |V_s|
   7              if η == l
   8                 then continue
   9                 else  F_{η,l} = min(F_{η,l}, D(v_i, v_j) + F_{η-1,l})
```

SK66算法的缺陷在于无法保证最终得到的路径是无环的。

2.2 SK算法

SK算法是在SK66算法的基础上修改得到的，通过保存更多的中间子路径，从而确保求得的路径无环的。

为了保证路径的无环性，必须确保通过公式(13)获得的最终路径和公式(12)获得的子路径是无环的。

在SK66算法的迭代过程中，根据公式(12)，对于每一个节点$v_i \in V_s$，需要选择一个新节点$v_l \in V_s$，从而得到一条从v_i到t的路径。然而这个过程可能过早的排除掉了一些高权值的可行子路径；同样的，此时选择的相对低权值的子路径可能会因为后续节点的不可达而被舍弃。记

$$f^1_{v_i,v'_l} = D(v_i, v'_l) + f^0_{v'_l}, \quad v'_l \neq v_i \tag{3}$$

$$f^\eta_{v_i,v_l} = D(v_i, v_l) + min_{v_j} f^{\eta-1}_{v_l,v_j}, \quad v_l \neq v_i, v_j \wedge v_i \neq v_j \tag{4}$$

其中$\eta = 2, \ldots, |V_s|$。这里主要的改动是：在每次迭代过程中，保存了从节点v_i到终点t，并且经过所有可能的中间节点v_l的所有路径，而不是在公式(12)中直接选择其中最小权重的子路径。具体来说，在SK66算法中，每次迭代结束后，对于每一个v_i只选择了一条路径；而在SK算法中，则选择了$|V_s| - 1$条路径。