

基于Matlab的有向图及无向图所有生成树搜索的算法与实现

配电网运行时的拓扑结构图为树状网络，配电网无向图的生成树与网络重构的候选解一一对应。无向图所有生成树的快速搜索算法对于配电网网络重构具有重要的理论与实际应用价值。以Matlab为平台具体实现了一种基于深度搜索与由原路返回的无向图及有向图所有生成树的搜索算法，算法包括邻接函数、节点类、双链表类、支路扩张函数和主函数。对于有向图，该算法的时间复杂度为 $O(V+E+EN)$ ，空间复杂度为 $O(V+E)$ ，其中 V 为顶点数， E 为边数， N 为生成树的个数。对于无向图，该算法的时间复杂度为 $O(V+E+VN)$ ，空间复杂度为 $O(V+E)$ 。算例表明，所建立的算法可以快速找到无向图及有向图的所有生成树，具有十分优越的性能。

张 剑/合肥工业大学电气与自动化工程学院

配电网运行时的拓扑结构图为树状网络，配电网网络重构的候选解与配电网无向图的生成树一一对应。无向图所有生成树的快速搜索算法对于网络重构具有十分重要的理论与实际应用价值。

许多算法被提出以解决图的所有生成树问题^[1-6]，主要分为以下几类。

第一类为基于由原路返回技术的搜索算法。这类算法能够列出每棵生成树的所有边。对于有向图，该算法的时间复杂度为 $O(V+E+EN)$ ，空间复杂度为 $O(V+E)$ ，其中 V 为顶点数， E 为边数， N 为生成树的个数。对于无向图，该算法的时间复杂度为 $O(V+E+VN)$ ，空间复杂度为 $O(V+E)$ 。

第二类算法是通过交换一对边来搜

国家自然科学基金：基于V2G的主动配电网电压协调优化控制（5150070099），高校博士点专项基金项目：光伏并网系统及有源滤波器建模与仿真研究（JZ2015HGBZ0095）。

寻新的生成树。这种方法只列出每棵生成树相对改变的边，通过压缩输出的边的个数，可以减少总的时间复杂度。文献[2]提出了一种采用这种压缩输出的方法，其时间复杂度是最优的，为 $O(V+E+N)$ ，但是空间复杂度处于劣势为 $O(VE)$ ；文献[7]提出了一种对已经搜索到的生成树进行边交换以得到新的生成树的搜索算法，并且压缩输出边的个数，其时间复杂度为 $O(V+E+N)$ ，空间复杂度为 $O(V+E)$ 。

第三类算法是基于逆向搜索的技术。文献[8]提出了一种基于逆向搜索的算法，其时间复杂度为 $O(V+E+N)$ ，空间复杂度为 $O(VE)$ 。反向搜索是枚举问题的一般性方法^[9-10]。

自从Jack Little和Cleve Moler于1984年创立MathWorks公司以来，经过国内外专家30年的努力，Matlab已经成为工程项目与科学研究的基本工具，在全世界拥有众多的用户，在国内外高等院校、科研、设计和制造部门都有广泛的应用。Matlab具有库函数丰富、规模大、功能强和运算速度快等特点，并且具有良好的二次开发功能，为用户进行程序开发提供了良好的便利性。

目前，在Matlab中还没有建立有向图及无向图所有生成树的搜索算法。本文基于第一类方法，以Matlab为平台，实现了有向图及无向图所有生成树的搜索算法，算法包括邻接矩阵函数、节点类、双链表类、支路扩张函数和主函数，并进行了试算。算例表明，实现



张 剑/讲师

关键词/Keywords

有向图·
无向图·
所有生成树·
深度搜索·
复杂度·

的算法可以快速找到无向图及有向图的所有生成树，具有十分优越的性能。

所有生成树搜索算法

1. 基本概念

在一个连通的无向图 G 中，一棵生成树是 G 的一个子图，在这个子图中， G 中的任意两个顶点之间有且仅有一条简单路径。一个桥是一条边 e ，图 $G - e$ 是非连通的，即 G 的每棵生成树都包含 e 。在一个有向图 G 中，一棵生成树（ r 为根节点）是 G 的一个子图，在这个子图中，从 r 到 G 中的任意节点有且仅有一条有向路径。一个桥（对于 r ）是一条边 e ，图 $G - e$ 不是以 r 为根节点，即 G 的每棵以 r 为根节点的生成树都包含 e 。图1为一个简单的有向图，其根节点为节点（1），由有向边1、5和6组成的集合为图1的一棵生成树。在图1中没有桥，因为去掉任何一条边都不可能使得图1是非连通的。图2是一个无向图，由无向边1、3和4组成的集合为图2的一棵生成树。在图2中没有桥，因为去掉任何一条边都不可能使得图2是非连通的。

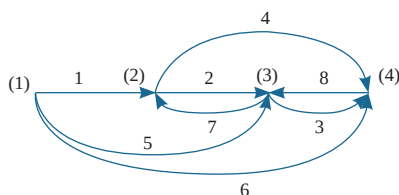


图1 一个简单的有向图

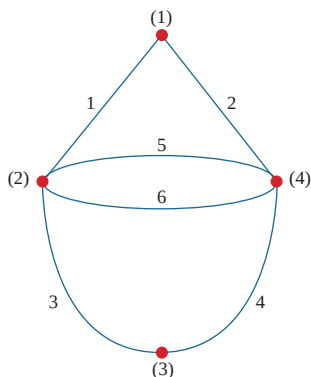


图2 一个简单的无向图

2. 搜索算法

本节将列出以 r 为根节点的有向图所有生成树的搜索算法，无向图的所有生成树的搜索算法在有向图的所有生成树的搜索算法的基础上稍作修

改即可得到。

为了找出以 r 为根节点的所有生成树，首先找出以 r 为根节点的包含子树 T 的所有生成树，步骤如下。首先选择一条边 e_1 ， e_1 从 T 指向 T 的外部，找出所有包含 $T \cup e_1$ 的生成树，从图 G 中删除 e_1 ，得到 G_1 ；然后选择一条边 e_2 ， e_2 从 T 指向 T 的外部。找出所有包含 $T \cup e_2$ 的生成树，从图 G_1 中删掉 e_2 ，得到 G_2 。重复以上过程，选择一条边 e_i ， e_i 从 T 指向 T 的外部，找出所有包含 $T \cup e_i$ 的生成树，从图 G_{i-1} 中删掉 e_i 得到 G_i 。停止此过程，直到边 e_k 是修改后的图的一个桥。由此，每一棵包含 T 的生成树都被找到一次且仅被找到一次。因为一棵生成树如果不包含任意 e_j ， $j < k$ ，那么它必定包含桥 e_k 。

本文选择边 e 使得 T 总是从深度优先方向扩张，即将 e 增加至 T 使得 T 沿着深度优先方向遍历 G 中的节点。假设所有包含 $T \cup e$ 的生成树已经找到，要确定 e 是否为修改后的图的一个桥。假设 L 是包含 $T \cup e$ 的最后一棵生成树，假设 $e = (u, v)$ ，直观上，在 L 中，相比于其他包含 $T \cup e$ 的生成树，顶点 v 的后代节点最少。因此在当前的图中，当且仅当不存在一条在 L 中从 v 派生出来的节点指向 v 的边时， e 是一个桥，这种方法可以作为 e 是否为桥的高效率的判定方法。

以下的算法采用了一个双向链表 F ， F 按照堆栈的模式进行操作，即遵循后进先出的原则， F 中的元素为从 T 指向其外部的支路。当 T 扩张时，支路 e 从 F 中弹出并添加至 T 。新的从 v 指向 $T \cup e$ 外部的边被压至 F 中。当 e 添加至 T 中时，某些边从 F 删除，当将 e 从 T 中删除时，这些从 F 中删除的边重新恢复其在 F 中的位置。删除与恢复操作必须保证这些边在 F 中的位置不变。否则， T 将无法按照深度优先扩张，从而导致程序失败。

除了 F ，算法采用了列表 FF ，是每一次递归调用的局部变量，用于重构最初的 F ， FF 采用堆栈的操作方式。算法的伪代码如下。

1) 程序 S （ S 找到根节点为 r 的有向图的所有生成树）begin

初始化 T ， T 含有根节点 r ，初始化 F 含有所有边 (r, v) ，即节点支路邻接函数输出结果矩阵 X 的

第 r 行所有非零元素。

```

Grow
end S
2) 程序Grow (Grow找到所有以 $r$ 为根节点的
包含 $T$ 的生成树) begin
    if  $T$ 含有 $V$ 个顶点 ( $V$ 为图的顶点个数), then
 $L \leftarrow T$ , 输出 $L$ , end
    else begin使 $FF$ 为一个空表, Grow的局部变量
    repeat
        从 $F$ 中弹出一条边 $e$ ,  $e$ 从 $T$ 指向 $v$ ,  $v \notin T$ 
        将 $e$ 添加到 $T$ 中
        向 $F$ 中压入每一条边 $(v, w)$ ,  $w \notin T$ 
        从 $F$ 中删去 $(w, v)$ ,  $(w, v)$ 
    Grow
        从 $F$ 中弹出每一条边 $(v, w)$ ,  $w \notin T$ 
        恢复从 $F$ 中删去的 $(w, v)$ ,  $w \in T$ 
        从 $T$ 、 $G$ 中删去 $e$ , 将 $e$ 添加至 $FF$ 中
        if 在当前的图中存在一条边 $(w, v)$ , 在 $L$ 中,  $w$ 不
        是 $v$ 的后代, then  $b \leftarrow \text{false}$ , else  $b \leftarrow \text{true}$ 
        until  $b$ 
        从 $FF$ 中弹出每一个 $e$ , 将 $e$ 压入 $F$ 中, 将 $e$ 添加
        到 $G$ 
    end end Grow

```

程序的具体实现过程

1. 节点支路关联矩阵

节点支路关联矩阵为一个 $M \times N$ 矩阵。其中, M 为节点数; N 为支路数。如果节点 i 是支路 j 的父节点,则矩阵的第 i 行第 j 列为1,否则为0。如图1所示的简单有向图,包含了4个节点、8条支路。节点支路关联矩阵 A 的表达式为

$$A = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \quad (1)$$

2. 节点支路邻接函数

节点支路邻接函数的输入为节点支路关联矩阵,输出为节点支路邻接矩阵。节点邻接支路的实现过程包括以下步骤。

1) 输入节点支路关联矩阵 A 。

2) 通过赋值语句生成一个与节点支路关联矩阵 A 维数相同的全零矩阵 X 。

3) 从左到右依次扫描节点支路关联矩阵 A 的每一行对应的元素,若元素不为零,则在全零矩阵 X 从左到右的位置一次记录该元素所处的列号;对于图1所示的有向图,节点支路关联矩阵函数的输出结果为

$$X = \begin{pmatrix} 1 & 5 & 6 & 0 & 0 & 0 & 0 & 0 \\ 2 & 4 & 0 & 0 & 0 & 0 & 0 & 0 \\ 3 & 7 & 0 & 0 & 0 & 0 & 0 & 0 \\ 8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \quad (2)$$

其含义为节点1是支路1、5和6的父节点,节点2是支路2、4的父节点,节点3是支路3、7的父节点,节点4是支路8的父节点。

3. 支路的父节点与子节点向量

支路的父节点与子节点在 T 扩张以及支路是否为桥判定时具有十分重要的作用。支路的父节点(子节点)向量存储的是每条支路的始端(末端)节点编号。如图1所示,支路的父节点向量为

$$B_p = (1 \quad 2 \quad 3 \quad 2 \quad 1 \quad 1 \quad 3 \quad 4) \quad (3)$$

支路的子节点向量为

$$B_s = (2 \quad 3 \quad 4 \quad 4 \quad 3 \quad 4 \quad 2 \quad 3) \quad (4)$$

其含义分别为支路1~8的父节点依次为节点1、2、3、2、1、1、3和4,支路1~8的子节点依次为2、3、4、4、3、4、2和3。

4. 节点类

节点类的数据属性包括节点数据值Data、后一个节点的数据值Next和前一个节点的数据值Prev。节点类的操作属性主要包括创建节点函数dlnode(Data)、在节点之后插入一个节点函数insertAfter(newNode, NodeBefore)、在节点之前插入一个节点函数insertBefore(newNode, NodeAfter)、从链表中移除节点disconnect(Node)和删除节点delete(Node)等函数。

5. 双向链表类

双向链表类的数据属性包括链表中元素的个数count,当前指针所在的位置currentpos,当前数据的值current。双向链表的操作属性主要包括函数

用于将链表当前位置指向第 p 个SetPosition(list, p)、用于在链表第 p 个位置后插入节点dlnode的函数InsertList(list, p , dlnode)、用于销毁链表的函数ClearList(list)、用于获取链表长度的函数size = ListSize(list)、用于读取链表中第 p 个数据的函数aNode = RetrieveList(list, p)、用于从链表中删除第 p 个数据的函数aNode = DeleteList(list, p)和用于将链表中的第 p 个数据替换为dlnode的函数aNode = ReplaceList(list, p , dlnode)。在进行扩张函数Grow时需要用到这些节点类以及链表类的函数。

6. 扩张函数Grow的实现过程

支路扩张函数Grow的实现过程是整个程序的关键部分, 采用了递归调用的方法, 在上文已经给出了伪代码。在判断删去的支路 e 是否为桥时, 向量Par中存储的是 L 中每个节点的父节点, N 为图 G 中总的支路数, 如果支路 k 被删除了, 那么 $G(k) = 0$, 代码如下。

```
flag = 1;
for g = 1 : N
    if (G(g)) ~ = 0
        if (Bs(G(g)) = Bs(e))
            w = Bp(G(g));
            u = Bs(e);
            while(Par(w) ~ = u)
                w = Par(w);
                if (w = 0)
                    flag = 0;
                    break;
            end
        end
        if (flag = 0)
            break;
        end
    end
    b = flag;
end
```

7. 主函数S的实现过程

主函数S的作用主要是进行相关全局变量的定

义以及对 T 、 F 的初始化。对于图1所示的有向图, 主函数的部分代码为

```
X = linjie(A);
T = zeros(1,M);
r = 1;
T(1) = r;
F = doubleLinkedList();
InsertList(F, ListSize(F), dlnode(1));
InsertList(F, ListSize(F), dlnode(6));
InsertList(F, ListSize(F), dlnode(5));
Grow();
```

无向图的搜索算法

无向图的搜索算法是将无向图转化为有向图, 按照有向图的搜索算法进行所有生成树的搜索。无向图转化为有向图的方法为: 将无向图的每条边分裂为两条带有箭头的方向相反的边, 其中一条边的编号等于原无向图中对应边的编号, 另一条边的编号等于原无向图对应边的编号加上无向图边的总数。图3为无向图经过转换成为有向图。在程序输出搜索结果时, 当边的编号大于无向图边的总数时, 减去边的总数即可输出无向图每棵生成树边的组成。

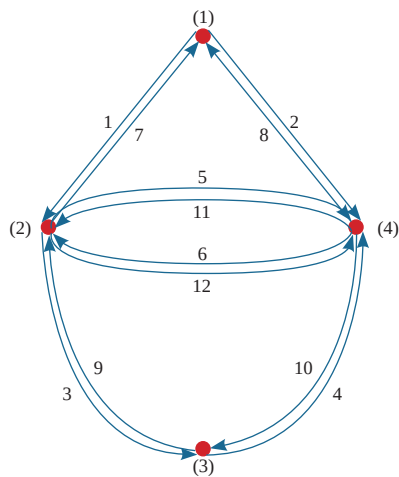


图3 无向图转换为有向图

仿真算例

算例1。图1为一个简单的有向图, 其所有生成树为5, 7, 4; 5, 7, 3; 5, 7, 6; 5, 3, 1;

5, 6, 1; 5, 1, 4, 6, 8, 7; 6, 8, 1; 6, 1, 2; 1, 4, 8; 1, 4, 2; 1, 2, 3共12棵生成树, 程序搜索时间为0.12 s。

算例2。图3为一个简单的无向图，其所有生成树为1, 6, 4; 1, 6, 3; 1, 5, 4; 1, 5, 3; 1, 3, 4; 1, 3, 2; 1, 2, 4; 2, 5, 3; 2, 5, 4; 2, 4, 3; 2, 4, 6; 2, 6, 3共12棵生成树，程序搜索时间为0.18 s。

算例3。图4为一个包含9节点13条支路的无向图,该无向图的所有生成树为463个,程序搜索时间为0.6 s。

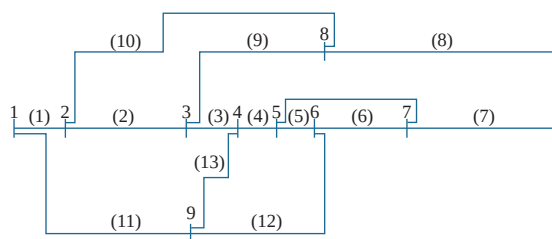


图4 9节点13支路无向图

结束语

基于Matlab建立了有向图及无向图所有生成树的搜索算法及其实现方法。对于有向图, 该算法的时间复杂度为 $O(V+E+EN)$, 空间复杂度为 $O(V+E)$, 其中 V 为顶点数, E 为边数, N 为生成树的个数; 对于无向图, 该算法的时间复杂度为 $O(V+E+VN)$, 空间复杂度为 $O(V+E)$ 。算例表明, 建立的算法可以快速地找到无向图及有向图的所有生成树, 具有十分优越的性能。

参考文献

- [1] Gabow H N, Myers E W. Finding all spanning trees of directed and undirected graphs [J]. Society for Industrial and Applied Mathematics, 1978, 7 (3) : 280-287.

- [2] Kapoor S, Ramesh H. Algorithms for generating all spanning trees of undirected, directed and weighted graphs [J]. Algorithms and Data Structures Lecture Notes in Computer Science, 1991, 51 (9) : 461-472.
- [3] Matsui T. An algorithm for finding all the spanning trees in undirected graphs [R]. Tokyo: University of Tokyo. 1993.
- [4] Minty G J. A simple algorithm for listing all the trees of a graph [J], IEEE Transactions on Circuit Theory, 1965, 12 (1) : 120-121.
- [5] Read R C, Tarjan R E. Bounds on backtracking algorithms for listing cycles, paths, and spanning trees [J]. American Mathematics Society, 1975, 5 (3) : 237-252.
- [6] Shioura A, Tamura A. Efficiently scanning all spanning trees of an undirected graph [J]. Journal of Operation Research Society, 1993, 7 (5) : 1-13.
- [7] Akiyoshi Shioura, Akihisa Tamura, Takeaki Uno. An optimal algorithm for scanning all spanning trees of undirected graphs [J]. Society for Industrial and Applied Mathematics, 1994, 26 (3) : 678-692.
- [8] Avis D, Fukuda K. Reverse search for enumeration [J]. Discrete Applied Mathematics, 1996, 65 (3) : 21-46.
- [9] Avis D, Fukuda K. A basis enumeration algorithm for linear systems with geometric applications [J]. Applied Mathematics Letters, 1991, 4 (5) : 39-42.
- [10] Avis D, Fukuda K. A pivoting algorithm for convex hulls and vertex enumeration of arrangements and polyhedral [J]. Discrete and Computational Geometry, 1992, 8 (1) : 295-313.

(收稿日期: 2014-05-20) EA

欢迎订阅 欢迎投稿 欢迎刊登广告