基于"断弦护枝"思想的MST构造算法的设计与分析

袁关伟[1,2] 赵家刚[1]
(西南林业大学计算机与信息学院 昆明 650224)[1]
(西南林业大学云南高校林业3S技术工程研究中心 昆明 650224)[2]

摘 要 为弥补MST经典算法的局限性,开创性地提出了构造MST的"断弦护枝"思想,并基于此思想创造性地设计与实现了一种全新的MST构造算法。实验结果和数学证明表明新算法是正确的;实验和分析结果表明新算法在某些实际应用领域能显著弥补经典算法的不足,具有重要的理论和应用价值。

关键词 断弦护枝,最小生成树,构造算法
中图法分类号 TP311.12,TP301.6 **文献标识码** A

Design and Analysis of MST Constructing Algorithm Based on the Idea that is Named as Pruning Bowstrings and Protecting Branches

YUAN Guan-wei[1,2] ZHAO Jia-gang[1]
(College of Computer and Information, Southwest Forestry University, Kunming 650224, China)[1]
(Research Center of Engineering on 3S Technology in Yunnan, Southwest Forestry University, Kunming 650224, China)[2]

Abstract In order to recuperate localization of classical algorithms, it is advanced for the first time that the idea that is named as "pruning bowstring and protecting branches", and a kind of downright new constructing algorithm of MST is designed and achieved creatively base on the idea. The result of experiment and math proving indicates that the new algorithm is right. The result of experiment and analysis indicates that the new algorithm can prominently recuperate the deficiency of classical algorithm in some actual application field and possesses important value on theory and application.

Keywords Pruning bowstrings and preserving branches, MST, Constructing algorithm

1 引言

最小生成树(MST)在现实世界中有着极其广泛的应用[4,7,8,12,14-19,23],普里姆算法(Prim)和克鲁斯卡尔算法(Kruskal)是构造MST的经典算法,在MST基础算法领域一直居于领袖地位[18-25]。但是,任何算法都有其局限性,即不存在放之四海而皆准的算法。例如,以上两种典算法在输出结果、边较少或悬挂边较多的稀疏图中的时间复杂度等都表现得不尽完美。因此,提出一种全新算法以弥补它们在这些方面的不足,无论对MST基础理论还是应用领域,都意义重大而深远。于是,本文算法应运而生。

2 基本概念

定义1[1,2,16,24,25] 设$G=\langle V,E\rangle$和$G'=\langle V',E'\rangle$为两个图,若V'是V的子集,且E'是E的子集,则称G'是G的子图。若G'是G的子图且$V'=V$,则称G'是G的生成子图。

定义2[1,2,16,24,25] 除始点和终点外,顶点各异的回路称为基本回路,也称为圈。

定义3[1,2,16,24,25] 连通无圈的无向图称为树。

定义4[1,2,16,24,25] 设T是无向图G的子图且为树,则称T为G的树。若T是G的树且为生成子图,则称T是G的生成树。设T是G的生成树,对任意的$e\in E(G)$,若$e\in E(T)$,则称e为T的树枝,否则称e为T的弦。

定义5[1,2,16,24,25] 设无向连通带权图$G=\langle V,E,W\rangle$,T是G的一棵生成树。T的各边权之和称为T的权,记作$W(T)$。G的所有生成树中权最小的生成树称为G的最小生成树。

定义6[1,2,16,24,25] 度为1的顶点称为悬挂点,与悬挂点关联的边称为悬挂边。

为叙述方便,设$G=\langle V,E,W\rangle$是一个无向连通带权图,T是G的一棵MST,E是存储G的所有边的辅助临时数组。不作特殊说明,以后文字中用到的G、T、E都是指此处的假设。另外,本文研究的图是简单图,其概念和文中的数学符号可参照文献[1,2]。

3 算法设计

为叙述方便,作如下规定:图G未质变成T前,图G及其非T生成子图统称为图G。

3.1 算法思想

本文算法的基本思想可概括为:断弦护枝,即(在图G中)剪断T的弦,(在图G中)保护T的树枝。权值最大但会破坏图G连通性的边必为T的树枝,权值最大且不会破坏图

袁关伟(1983—),男,硕士生,工程师,主要研究方向为GIS算法、设计与开发,E-mail:yuanweilinkhappy@163.com。

G 连通性的边选为 T 的弦。具体为：重复从图 G 中删除一条不会破坏其连通性的权值最大边，直至图 G 中只剩下 $|V(G)|-1$ 条边为止。

如果从图 G 中删除第 1 条边到第 $|E(G)|-|V(G)|$ 条边的过程中，图 G 一直在发生量变，那么当从图 G 中删除第 $|E(G)|-|V(G)|+1$ 条边时，图 G 就发生了质变，即变成了一棵最小生成树 T。

3.2 算法实施策略

算法思想十分明确，但要按这一思想实现算法，显然存在一个比较棘手的问题，即如何寻找出不会破坏图 G 连通性的权值最大边？这一大问题又可细分为两个子问题，即：(1)如何找出图 G 的权值最大边？(2)如何判断该边是否会破坏图 G 的连通性？下面就分别阐述解决这两个子问题的策略：

(1)寻找图 G 的权值最大边：将图 G 的所有边复制到一个辅助临时数组 $E[]$ 中，然后通过适当的办法选出 $E[]$ 中的权值最大边，该边在图 G 中的映射即为图 G 的权值最大边。

(2)判断权值最大边是否会破坏图 G 的连通性：将 $E[]$ 中的权值最大边在图 G 中的映射鲁莽地从图 G 中删除，然后再检查被删除边的两顶点是否仍属于同一个连通分量，若属于，说明被删除边不会破坏图 G 的连通性；否则，说明被删除边会破坏图 G 的连通性。

显然，要解决问题(2)，即必须解决怎样判断两个顶点是否属于同一连通分量的问题，不妨将此问题称为问题(3)，阐述如下：

(3)判断两个顶点是否属于同一连通分量：以一个顶点为访问开始点，在其所在的连通分量中采用广(深)度优先搜索策略来查找另一个顶点，一旦查找到，则立即主动结束搜索，因为此时已足以说明两顶点属于同一连通分量；若搜索操作自动结束时(遍历完访问始点所在连通分量)，还没查找到另一顶点，则说明两顶点不属于同一连通分量。

在解决问题(1)和(2)时，还会产生相应的副问题，分别称之为(4)和(5)，阐述如下：

(4)在解决问题(1)时，因为图 G 是无向图，所以被复制到数组 $E[]$ 中的边必然有一半是重复的[1,3-8,16]。为了防止同一条边被选择两次进而导致被删除两次的错误发生，可以采取只保留始点小于终点的边的策略，具体为将终点大于始点的边的权值根据实际置为一个极小值，让它们永远不可能被选到。

(5)在解决问题(2)时，鲁莽地将权值最大边在图 G 中的映射从图 G 中删除("先斩后奏"策略)，若通过检验后，该边会破坏图 G 的连通性，那么该次删除操作显然是失误的。为弥补这种失误，可采取反悔操作("知错就改"策略)，即将失误删除的权值最大边重新添加到其在图 G 中的原位置上。

上述那个棘手的大问题最终被分解成 5 个小问题，通过论证，5 个小问题都是可解的，所以整个算法必然可以实现。

3.3 算法优化策略

(1)在算法实施策略的(2)、(3)中，阐述了判断权值最大边是否会破坏图 G 连通性的通用策略。然而，对于权值最大的悬挂边，该类边显然一定会破坏图 G 的连通性，而要判断一条边是否为悬挂边检查其两顶点度的情况即可。因此，为方便检查顶点的度，可以在定义顶点时添加一个表示其度的数据项，这项工作在邻接矩阵和邻接表存储结构下，都非常容易实现，而且几乎不产生任何额外负担。这样一来，对该类边的判断，完全可以在常数阶时间复杂度下完成。该类边在原始图 G 及其非 T 生成子图中都会出现，利用此策略可显著降低算法的误删率和提高算法的护边效率，对算法性能影响明显。

(2)如果一条边会破坏图 G 的连通性，那么将其删除后，图 G 必被分成两个连通分量，一般地，被删除边的度较小的顶点所在的连通分量的顶点数较少，所以调用搜点函数时，以被删除边的度较小的顶点为参数能降低算法的时间复杂度。

(3)在辅助临时数组 $E[]$ 中选择权值最大边的方法有多种，为了最大限度地降低算法的时间复杂度，可以采用数据结构堆来实施权值最大边的选择。若采用堆来选择权值最大边，除第一次需要 $O(e)$ 的时间开销外，其余每次仅仅需要 $O(\log e)$ 的时间开销[4,9]。若利用排序算法来选择权值最大边，那么即使性能较好的快速排序和堆排序，也要 $O(e\log e)$ 的时间开销(当然这是选择权值最大边的总时间开销)，并且，算法的目的只是为了选择权值最大边，绝非为了排序。例如，在最好情况下，只需执行 $|E(G)|-|V(G)|+1$ 次选择，后面排好序的 $|V(G)|-1$ 条边根本用不上。

3.4 算法描述

为能清楚表述算法主要思想，又不局限于程序设计语言的细枝末节，算法用自然语言描述如下(未指明转向的步骤按自然顺序执行)：

算法："断弦护枝"构造 MST 算法

输入：无向连通带权图 G；

输出：G 的最小生成树 T；

过程：

(1)将图 G 的所有边复制到辅助临时数组 E[]中；

(2)删除 E[]中的重复边；

(3)建立初始大根堆，E[0]即是权值最大边；

(4)若 E[0]是悬挂边，转(10)；否则，执行(5)；

(5)鲁莽地将 E[0]的映射从图 G 中删除；

(6)若 E[0]的两顶点同属一个连通分量，执行(7)；否则，转(8)；

(7)图 G 的边数|E(G)|减 1；转(9)；

(8)实施反悔操作，即将刚刚被删除的边重新添加到其在图 G 中的原位置上。转(10)；

(9)若|E(G)|等于|V(G)|-1，结束循环；否则，执行(10)；

(10)将 E[0]从数组 E[]中彻底删除；

(11)调整大根堆，E[0]即为权值最大边；转(4)。

3.5 算法模拟

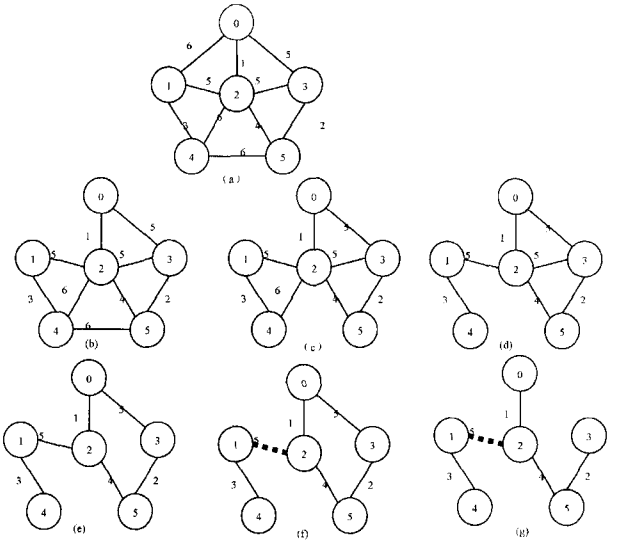

图 1 本文算法构造最小生成树的过程图

图 1(a)[1,4,6,10]是用来构造 MST 的无向连通带权图 G，(b)至(f)分别表示每次在 G 中删除边的情况，图中(红)虚线边表示被鲁莽误删除后实施反悔操作恢复了的边。

4 算法实现

限于篇幅，本文只给出邻接表存储结构下实现方案的关键代码，邻接矩阵存储结构下的实现方案留给读者自己思考。需要指出的是，邻接矩阵存储结构下的实现方案相对更容易，因为删除边和添加边的操作要简单得多。

算法用 C 语言实现(图 G 的顶点是从 0 开始编号)。

```
int search_Vertex(AdjLGraph *g,int sv,int ev)
{/*搜点函数,从一点 sv 出发沿其所在的连通分量广度优先搜索另一点 ev,若找到,返回 1;否则,返回 0*/
  EdgeNode *p;
  int queue[GVN],front=0,rear=0;//定义队列
  visited[sv]=1;//访问"访问始点"
  rear=(rear+1)%GVN;//队尾指针加 1
  queue[rear]=sv;//"访问始点"入队
  while(front! =rear)//队不空时循环
  {
    front=(front+1)%GVN;//队头指针加 1
    p=g->V[queue[front]].firstEdge;//第一条边
    while(p)//边存在时循环
    {
      if(visited[p->ev]==0)//顶点没访问过
      {//搜索到另一顶点,主动结束搜索
        if(p->ev==ev) return 1;
        visited[p->ev]=1;
        rear=(rear+1)%GVN;
        queue[rear]=p->ev;//入队
      }
      p=p->nextEdge;//取下一条边
    }//搜索完一条单链表
  }//搜索自动结束
  return 0;
}
void construct_MST(AdjLGraph *g)
{/*限于篇幅,调整函数、删边函数、添边函数在本文中未作定义,使用时望读者自己添加。在 copy_Edge(g,E)过程中,k 统计边数*/
    int i,k=0,vst_sv,vst_ev;
    EdgeNode *p;
    Edge E[2*|E(G)|];//数组大小为图 G 的边数|E(G)|的 2 倍
    copy_Edge(g,E);//将图 G 的所有边拷贝到数组 E[]中
    for(i=0;i<k;i++)//删除数组 E[]中的重复边
      if(E[i].sv>E[i].ev)//详阅算法实施策略(4)
        E[i].w=-∞;//根据实际需要置为一个极小值
    for(i=(k-2)/2;i>=0;i--)//建立初始大根堆
      adjust_BT_to_Heap(E,k,i);//调用调整函数
while(1)
{//顶点度数据项在定义顶点时体现,详阅算法优化策略(1)
    if(g->V[E[0].sv].degree>1 &&
    g->V[E[0].ev].degree>1)//对非悬挂边的处理
    {//顶点度数的减小在删边函数内实现
      delete_Edge(g,E[0].sv,E[0].ev);
      delete_Edge(g,E[0].ev,E[0].sv);
      for(i=0;i<g->n;i++)//数组 visited[]元素置 0
        visited[i]=0;//全局数组,搜点函数中将用到
      vst_sv=(g->V[E[0].sv].degree)<
      (g->V[E[0].ev].degree)? E[0].sv:E[0].ev;
      vst_ev=(vst_sv==E[0].sv)
      ? E[0].ev:E[0].sv;//详阅算法优化策略(2)
      if(! search_Vertex(g,vst_sv,vst_ev))
      {//顶点度数的增加在添边函数内实现
        add_Edge(g,E[0].sv,E[0].ev,E[0].w);
        add_Edge(g,E[0].ev,E[0].sv,E[0].w);
      }//被删除边的两顶点不属于同一连通分量
      else
      {
        (g->e)--;
        if(g->e==(g->n)-1) break;
      }
    }
    E[0]=E[--k];//删除权值最大边,同时参选个数减 1
    adjust_BT_to_Heap(E,k,0);//调整大根堆
  }//end-while
}//end-construct_MST
```

5 算法测试

以北京市为例，假设要在其各辖区间修建地铁，使得各辖区的市民可通过地铁互相通行，并实现修建的代价最小[17]。通过相关专家实地考察，依据两两城区间修建地铁的可能性和必要性原则，得到了如图 2 所示的"北京市地铁规划原型图"，两城区连线上的数字表示距离(在此工程中，修建代价用距离来衡量)。以图 2 为例，在 VS 2008 中用 C 语言设计测试程序，程序的输出结果是邻接表状的"地铁规划原型图 G"和"地铁规划确认图 T"及其各自修建里程，以"地铁规划确认图 T"为最终的规划方案，结果如图 3 所示。

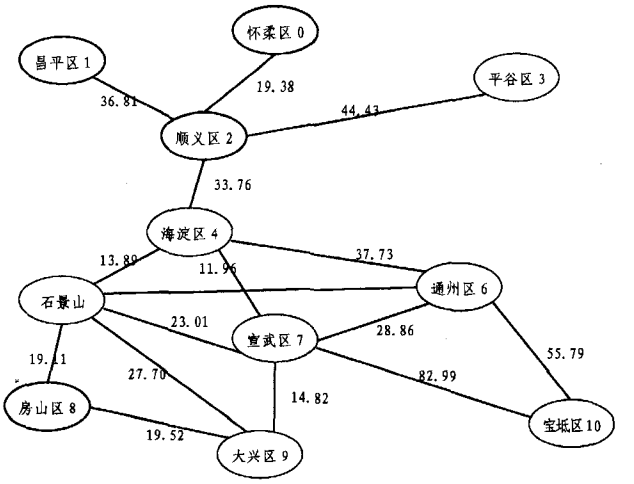

图 2 北京市地铁规划原型图(单位：km)[17]

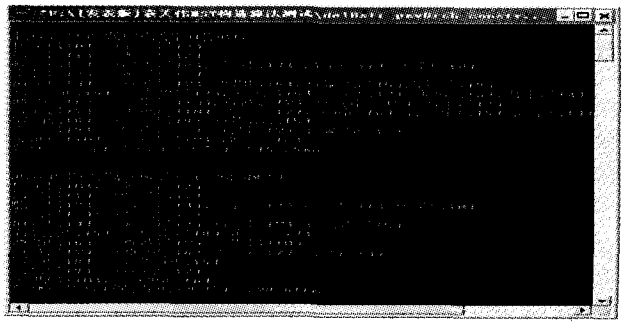

图 3 算法测试结果图

6 算法证明

定理 如果无向连通带权图 G、T 分别是“断弦护枝”构造 MST 算法的输入和输出，则 T 是 G 的最小生成树。

证明：(1)T 是 G 的生成树。

“断弦护枝”构造 MST 算法重复执行从图 G 和其生成子图中删除一条不会破坏它们连通性的(权值最大)边，直至 G 的某个生成子图中只剩下 $|V(G)|-1$ 条边。显然，当算法终止时，G 的生成子图仍然是连通图，且其中有 $|V(G)|$ 个顶点和 $|V(G)|-1$ 条边，G 的该生成子图即是 T，那么 T 必是 G 的一棵生成树。

(2)T 的权值不大于 G 的任何一棵生成树的权值(MST 可能不唯一)。

设 T^* 是 G 的一棵最小生成树。

①如果 T 的边和 T^* 的边完全相同，则 $W(T)=W(T^*)$，那么 T 的权值不大于 G 的任何一棵生成树的权值。

②如果 T 的边和 T^* 的边不完全相同，则在 T 中必有边 e_{i+1}，使得 e_{i+1} 不是 T^* 的边，而 $e_i,e_{i-1}\cdots e_3,e_2,e_1$ 是 T^* 的边，序列 $e_{i+1},e_i,e_{i-1}\cdots e_3,e_2,e_1$ 是按权值非递增序被保留在 T 中的边，换言之就是把不同边中的权值最小边定为第一条不同边。因为 T^* 是树，所以将 e_{i+1} 加入 T^* 中必然形成一个圈 α。同样，因为 T 是树，则在圈 α 中至少存在一条边 x 不在 T 中。同时，根据本文算法在执行过程中优先破除大边圈(优先删除权值最大边)的特点和上述对第一条不同边的规定，可推知 $\{e_1,e_2,e_3\cdots e_{i-1},e_i\}\cup\{x\}\cup\{e_{i+1}\}$ 一定含圈 α。

现将 T^* 中的边 x 换成 e_{i+1}，必然形成一棵新的生成树 T'，并有如下关系成立：$W(T')=W(T^*)+W(e_{i+1})-W(x)$。因为 T^* 是最小生成树，所以 $W(T*)\leqslant W(T')$，进而可推知：$W(e_{i+1})-W(x)\geqslant 0$，即 $W(e_{i+1})\geqslant W(x)$。

假设 $W(e_{i+1})>W(x)$ 成立。

根据本文算法的执行过程可知，一条边最终存于 T 中没被删除，有且只有两种原因：算法终止前该边没被选择过(原因一)；该边被选择过并被判定为悬挂边或会破坏图 G 的连通性(原因二)。

假设算法终止前 e_{i+1} 没被选择过，因为 $W(e_{i+1})>W(x)$，算法又是按权值非递增序选边的，那么算法终止前 x 必然也没被选择过，而没被选择过的边是不可能被删除的，这与 x 最终被删除没存于 T 中的事实矛盾。那么，e_{i+1} 必然被选择过，同样因为 $W(e_{i+1})>W(x)$，算法又是按权值非递增序选边的，所以 e_{i+1} 被选择时 x 仍存于 G 的某非 T 生成子图中(待选择)，因为 $\{e_1,e_2,e_3\cdots e_{i-1},e_i\}\cup\{x\}\cup\{e_{i+1}\}$ 一定含圈 α，而 $\{e_1,e_2,e_3\cdots e_{i-1},e_i\}$ 是 $E(T)$ 的真子集，$E(T)$ 又是 G 的任何非 T 生成子图的边集合的真子集，所以 e_{i+1} 被选择时 G 的该非 T 生成子图中至少含一个圈 α，显然 e_{i+1} 存于 α 中，而圈中的边一定不是悬挂边，也一定不会破坏图 G 的连通性，所以算法对 e_{i+1} 判断后的处理措施必然是将其彻底删除，这与其最终没被删除而存于 T 中的事实矛盾。

综上所述，e_{i+1} 没被删除与算法的执行过程矛盾。所以 $W(e_{i+1})>W(x)$ 不成立。结合上面的结论，必然可推出：$W(e_{i+1})=W(x)$。

因为 $W(e_{i+1})=W(x)$，所以 $W(T')=W(T^*)$，即 T' 也是图 G 的一棵最小生成树。不难看出，经过一次变换，T^* 与 T 中的相同边多了 1 条。然后再把 T' 看成 T^*，用同样的变换方法反复进行，直至 T^* 中的边与 T 中的边完全相同。因为 T^* 与 T 的不同边的数量必然在区间 $[1,n-1]$ 中，即是有限的，那么经过有限次变换必然可使得 T^* 中的边与 T 中的边完全相同，而变换结束时的 T^* 与变换前的 T^* 的权值相等，从而证得 T 是 G 的最小生成树。证毕。

7 算法分析

7.1 性能分析

令 $n=|V(G)|$，$e=|E(G)|$，本文算法的时间开销主要来自 3 部分：权值最大边的选择、两顶点是否属于同一连通分量的判断和删除边的操作。选择权值最大边的时间开销是 $O(\log e)$(调整函数)；判断两顶点是否属于同一连通分量的时间开销在邻接表存储结构下最坏是 $O(e)$[3,4,7-9,13]，在邻接矩阵存储结构下最坏是 $O(n^2)$[4,9](搜点函数)；在邻接表存储结构下删除边的最大和最小时间开销分别是 $O(n)$($e\geqslant n$)和 $O(1)$，在邻接矩阵存储结构下删除边的时间开销是 $O(1)$(删边函数)。while 循环执行的最大次数是 $e-2$ 次(原始图 G 中存在由 3 条权值最小且相同的边构成的圈，并且权值最小且相同的边仅有 3 条，用反证法易证此结论)，最小次数是 $e-n+1$ 次(原始图 G 中按权值非递增序排列的前 $e-n+1$ 条边都是 T 的弦)，综上所述，本文算法在邻接表存储结构下的时间复杂度最好为 $O(e^2-en)$(当边比较少时，仅相当于线性级开销)，最坏为 $O(e^2)$；在邻接矩阵存储结构下的时间开销最好为 $O(en^2-n^3)$，最坏为 $O(en^2)$。因为实施本文算法需要一个辅助临时数组 $E[]$ 存储图 G 的边、一个访问标识数组和一个队列存储图 G 的顶点，所以其空间复杂度为 $O(e)$($e\geqslant n$)。由于搜点函数绝大多数时候都不需要访问完图 G 的所有顶点，删边函数绝大多数时候都趋近于常数阶时间复杂度，调整函数的参选个数每次循环时都在减小，加之悬挂边在原始图 G 及其非 T 生成子图中都会出现，而这些都可以显著影响算法性能，因此可推知本文算法的性能实际上很难接近最坏时间复杂度，平均性能较佳。不难看出，本文算法较适合在邻接表存储结构下构造 MST，而稀疏图一般采用邻接表作为存储结构[4,5,8-11]，故本文算法较适合于稀疏图。

7.2 对比同类经典算法

Prim 算法是利用 MST 性质构造最小生成树的典型算法[4]，Kruskal 算法是基于避圈思想构造最小生成树的典型算法[1,2,16]，它也利用了 MST 性质[4]。Prim 算法的时间复杂度为 $O(n^2)$[3-16]，Kruskal 算法的时间复杂度为 $O(e^2)$[6]，利用堆排序算法和并查集改进后的时间复杂度为 $O(e\log e)$[4]。以输入为图 G，输出为 MST 的边集合实现算法，Prim 算法和 Kruskal 算法的空间复杂度分别为 $O(n)$ 和 $O(e)$。Prim 算法和 Kruskal 算法都利用了贪心算法设计策略[3,4]，最终都是选出 MST 的边集合，并未真正构造出 MST。结合此两种经典算法的其他固有属性，可得如下结论：

(1)本文算法在邻接表存储结构下的时间复杂度与此两种经典算法相当，但随其问题规模的增长，其趋势将低于 Prim 算法，略高于 Kruskal 算法。因为 Prim 算法、本文算法、Kruskal 算法的大循环内存在的主要线性级开销的代码段分别为 2 个、1 个、1 个(且仅 1 个)。

(下转第 460 页)

化可信性进行分析。

参 考 文 献

[1] OWL-S Coalition: OWL-S: Semantic Markup for Web Service [EB/OL]. http://www.ai.sri.com/daml/services/owl-s/1.2/overview/,2006

[2] 朱俊,郭长国,吴泉源. 一种基于广义随机 Petri 网的 Web 服务组合性能预测模型[J]. 计算机科学,2011(8)

[3] Brogi A, Corfini S, Iardella S. From OWL-S Descriptions to Petri Nets[C]//ICSOC Workshops. 2007:427-438

[4] Norton B, Foster S, Hughes A. A Compositional Operational Semantics for OWL-S[J]. EPEW/WS-FM, 2005(3670):303-317

[5] 周敏,张为群,林已杰,等. 一种基于扩展 Owl-S 本体的 Web 服务质量度量及评价方法的研究[J]. 计算机科学,2010(5)

[6] Mitra P, Wiederhold G. An Ontology-Composition Algebra[M]. Handbook on Ontologies, 2004:93-116

[7] Moldt D, Ortmann J. DaGen: A Tool for Automatic Translation from DAML-S to High-Level Petri Nets [J]. FASE, 2004 (2984):209-213

[8] German R. Performance analysis of communication systems: Modeling with non-Markovian stochastic Petri nets[M]// John Wiley and Sons. Chichester, 2000

[9] http://www.daml.org/services/owl-s/1.1/CongoProcess.owl

(上接第 440 页)

(2)本文算法与两种经典算法的空间复杂度相当。

(3)本文算法的最坏、平均和最好时间复杂度区别最明显,Kruskal 算法次之,Prim 算法几乎不存在三者之分。根本原因是 Prim 算法几乎不受大循环内分支结构影响,而本文算法和 Kruskal 算法受大循环内分支结构影响明显。

(4)Prim 算法的时间复杂度与边无关,适合于稠密图。本文算法和 Kruskal 算法的时间复杂度都与边息息相关,适合于稀疏图。

(5)当 $n-1<e<2n-2(e-n+1<n-1)$时,邻接表存储结构下的性能优于 Kruskal 算法和 Prim 算法,这正如一袋含有较少砂的米,要将米沙分离,拈沙总比拈米快。

(6)本文算法性能受悬挂边的影响非常大,但能显著改善算法性能,但对 Kruskal 算法和 Prim 算法的性能改善几乎没影响。

(7)输出结果明显优于此两种经典算法,本文算法的输出结果是 MST,本质上是图,在其他函数或类中使用时可以调用图的一切算法;此两种经典算法的输出结果是 MST 的边集合,本质上是数组,MST 的所有边和顶点依然没被联系在一个数据结构中,在其他函数或类中使用不便。

综上所述,若图是稠密图($e\to n(n-1)/2$),则建议采用 Prim 算法并用邻接矩阵存储结构;若图是稀疏图($2n-2\leqslant e\ll n(n-1)/2$),则建议采用 Kruskal 算法并用邻接表存储结构;若图是稀疏图($n-1<e<2n-2$),则建议采用本文算法并用邻接表存储结构。这样的选择既可节省存储空间,又能提高算法性能。另外,在悬挂边天然较多和希望输出结果比较形象生动的情况下,优先选用本文算法。

除上述两个经典算法外,Sollin 算法也是构造 MST 的常用算法,其也是利用贪心算法设计策略,最终同样是选出 MST 的边集合,时间复杂度为 $O(n^2\log n)$[3,20,23];文献[15]指出管梅谷教授提出了破圈法来构造 MST 的思想,实现情况未知。本文提出的算法利用了贪心算法设计策略,同时也吸取了动态规划法和回溯法两种算法设计策略的精髓。有关贪心法、动态规划法和回溯法等设计策略,详见文献[7,10-13]。

结束语 本文阐明了一种全新 MST 构造算法的完整科学体系——思想、设计、实现、实验、分析、证明,填补了 MST 基础理论和应用领域的一项空白。

参 考 文 献

[1] 耿素云,屈婉玲. 离散数学(修订版)[M]. 北京:高等教育出版社,2004

[2] 尹宝林,何自强,许光汉,等. 离散数学(第三版)[M]. 北京:高等教育出版社,2011

[3] Horowitz E, Sahni S, Anderson-Freed S. 数据结构基础(第二版)[M]. 朱仲涛,译. 北京:清华大学出版社,2009

[4] 严蔚敏,吴伟民. 数据结构(C 语言版)[M]. 北京:清华大学出版社,1997

[5] 朱战立. 数据结构——使用 C 语言(第四版)[M]. 北京:电子工业出版社,2009

[6] 李春葆,尹为民,李蓉蓉,等. 数据结构教程(第二版)[M]. 北京:清华大学出版社,2007

[7] 廖明宏,郭福顺,张岩,等. 数据结构与算法(第四版)[M]. 北京:高等教育出版社,2007

[8] 陈媛,何波,卢玲,等. 算法与数据结构(第二版)[M]. 北京:清华大学出版社,2011

[9] 张铭,王桂龙,赵海燕. 数据结构与算法[M]. 北京:高等教育出版社,2008

[10] 陈平,褚华. 软件设计师教程(第二版)[M]. 北京:清华大学出版社,2006

[11] 张铭,赵海燕,王腾蛟,等. 数据结构与算法实验教程[M]. 北京:高等教育出版社,2011

[12] 王晓冬. 计算机算法设计与分析(第三版)[M]. 北京:电子工业出版社,2007

[13] Alsuwaiyel M H. 算法设计技巧与分析[M]. 吴伟昶,方世昌,等,译. 北京:电子工业出版社,2010

[14] 张宏,温永宁,刘爱利,等. 地理信息系统算法基础[M]. 北京:科学出版社,2006

[15] 马良,朱刚,宁爱兵. 蚁群优化算法[M]. 北京:科学出版社,2008

[16] 李明哲,金俊,石瑞银. 图论及其算法[M]. 北京:机械工业出版社,2010

[17] 滕国文. 数据结构课程设计[M]. 北京:清华大学出版社,2010

[18] 陈国龙,郭文忠,涂雪珠,等. 求解多目标最小生成树问题的改进算法[J]. 软件学报,2006,3(17):364-370

[19] 宁爱兵,熊小华,马良. 最小生成树灵敏度分析算法研究[J]. 小型微型计算机系统,2011,4(32):744-746

[20] 孙凌宇,冷明,谭云兰,等. 赋权有向图的最小生成树算法[J]. 计算机工程,2010,2(36):61-66

[21] 王立冬. 约束最小生成树算法研究[D]. 西安:西安电子科技大学,2009

[22] 王海涛. 内点带权的最小生成树[D]. 上海:复旦大学,2006

[23] 陈哗君. 基于最小生成树的上海板块股票网络分析[D]. 北京:北京工业大学,2010

[24] Grimaldi R P. 离散数学与组合数学(第五版)[M]. 林永钢,译. 北京:清华大学出版社,2007

[25] West D B. 图论导引(第二版)[M]. 李建中,骆吉洲,译. 北京:机械工业出版社,2005