第 5 卷第 4 期
1994 年 4 月

软 件 学 报
JOURNAL OF SOFTWARE

Vol. 5, No. 4
Apr. 1994

一类排污问题在树图上的线性算法*

朱大铭 马绍汉

（山东大学计算机科学系，济南 250100）

摘要 MEGIDDO 等人证明了图搜索问题的 NP 完全性并给出一个树图上的算法，可在 O(n) 时间内求解树的搜索数，在 O(nlog(n)) 时间内求解树搜索方案. 本文通过引入搜索方案边序表示法给出一个线性算法，可在 O(n) 时间内同时求得树的搜索数和搜索方案.

关键词 算法，NP 完全性，树，无向连通图.

1 问题简述

给定无向连通图 G=(V,E). 设想该图表示一个街道系统，系统中藏有一名罪犯. 一组搜索者正在追踪罪犯. 假设罪犯熟悉街道布局，可以通过无人把守的道口由一条街道进入另一街道；假设罪犯行速较搜索者快得多，只能在一条街道上两端夹击将罪犯抓住. 问：怎样搜索该系统才能用最少的搜索者保证抓住罪犯？这是城市街道系统排污的一类问题模型.

搜索一个图 G 有三个基本动作：(1)将一个搜索者置于某顶点 v；(2)一个搜索者从顶点 v_1 沿边 v_1v_2 搜索至 v_2；(3)从顶点 v 撤回搜索者. 搜索图 G 所需最少的搜索者个数称为图 G 的搜索数，记为 s(G). 这个问题首先由 PARSONS 提出[2]，MEGIDDO，HAKIMI，GAREY，JOHNSON 等人证明该问题为完全问题并给出一个限制 G 为树 T 时的多项式时间算法[1]，对于 n 个顶点的树 T，可在 O(n) 时间内计算搜索数 s(T)，O(nlog(n)) 时间内计算搜索方案 P(T). 该问题称为图搜索问题. 本文首先由 LaPaugh 的结论[3]提出一种用边序列表示搜索方案的方法，然后给出一个线性时间算法，可在 O(n) 时间内同时求得树的搜索数和搜索方案.

2 已有的一些结果

首先简述 MEGIDDO 等人的算法 compute－info. 采用分而治之的思想求解树的搜索数 s(T). 其算法主要基于如下结论：若 T 的搜索数为 k，则(1)T 有一个中心顶点 v，其分支的搜索数均小于 k；或(2)T 有一条林荫路，该路径上顶点最多有两个分支搜索数为 k. 于是

---

* 本文 1991 年 7 月 1 日收到，1992 年 1 月 29 日定稿

作者**朱大铭**，30 岁，助教，主要研究领域为人工智能，算法分析与设计. **马绍汉**，56 岁，教授，主要研究领域为算法分析与设计，人工智能.

本文通讯联系人：朱大铭，济南 250100，山东大学计算机科学系

可按树根位置将树分为四类：(1)H 型，树根在中心顶点；(2)E 型，树根在林荫路 $v_1,v_2,\cdots,v_r$ 的头顶点 v_1,v_r 的分支上或 v_1,v_r 本身；(3)I 型，树根为某个林荫路中间顶点；(4)M 型，树根在林荫路中间顶点的搜索数小于 s(T)的分支上，根所在的分支称为 T 的 M－树.

算法递归计算 T 的信息记录 info(T)＝(type,s,M－info). type 为树的类型；s 为搜索数；M－info 为 T 的 M－树的信息记录. 设树根为 r,d(r)为其度.

(1)若 d(r)＝1，有两种情况. 若 T 仅是一条边，则 info(T)＝(E,1,nil)；否则将 T 的根移到 r 的唯一邻顶 r′得 T′，info(T)＝re_root(info(T′)).

(2)若 d(r)＞1，将 T 分裂成子树 T_1,T_2，info(T)＝merge(info(T_1),info(T_2)).

re_root 过程由(type′,s′,M－info′)计算(type,s,M_info)：

情况(1)若 type′＝E，则 type＝E；s＝s′.

情况(2)若 type′＝H，则 type＝E；s＝s′.

情况(3)若 type′＝I；s′＝1，则 type＝E；s＝s′. 若 type′＝I；s′＞1，则 type＝M；s＝s′；M_info＝(E,1,nil).

情况(4)若 type′＝M，则 type＝M；M_info＝re_root(M_info′)；s＝s′.

合并过程 merge 由(type1,s1,M－info1)和(type2,s2,M－info2)计算(type,s,M_info). 不失一般性设 s1≥s2.

情况(1)若 s1＝s2；type1＝type2＝H，则 info(T)＝(H,s1,nil).

情况(2)若 s1＝s2；type1＝H；type2＝E 或相反，则 info(T)＝(E,s1,nil).

情况(3)若 s1＝s2；type1＝type2＝E，则 info(T)＝(I,s1,nil).

情况(4)若 s1＝s2；type1＝I；type2＝H 或相反，则 info(T)＝(I,s1,nil).

情况(5)对于 s1＝s2 的其他情况，info(T)＝(H,s1＋1,nil).

情况(6)若 s1＞s2；type1＝H,E 或 I，则 info(T)＝(type1,s1,nil).

情况(7)若 s1＞s2；type1＝M，则：info(T′)＝(type′,s′,M－info′)＝merge(M－info1,(type2, s2, M － info2)). 若 s′＜s1，则 info(T)＝(M, s1, info(T′))，否则 info(T)＝(H,s1＋1,nil).

由[1]知该算法得到树 T 的搜索数 s(T). 设计算 s(T)时间复杂度为 S(T)，则：$S(T)\leqslant R(T)+S(T_1)+S(T_2)+M(T_1,T_2)$. R(T)为执行一次 re－root 的时间；$M(T_1,T_2)$为执行过程 merge 的时间. 递归调用一次 re－root，搜索数至少减 1，故 R(T)＝O(s(T)). $M(T_1,T_2)$可由 $m(s(T_1),s(T_2))$估计：$m(s(T_1),s(T_2))\leqslant m(s(T_1)-1,s(T_2))+O(1)$，从而 $S(T)\leqslant S(T_1)+S(T_2)+O(s(T))$. 由此得 S(T)＝O(ns(T))＝O(nlog(n)).

3　树搜索方案构造算法

LaPaugh 指出[3]：对于无向图 G，允许边重新污染不减少图 G 的搜索数 s(G). 因此图的搜索方案可由该图的边被搜索的顺序表示. 对于 G 的边序列 P，s(P)表示按 P 搜索图 G 所需最少搜索者数.

定义 1. 给定 G＝(V,E)，v∈V. 若 v 邻接的边全是干净边，则称 v 是干净顶点；若 v 邻接的全是污染边，则称 v 是原始顶点；其他情况称 v 是混合顶点. 设 E1⊆E，s(P,E1)表示按

照 P 搜索图 G 并形成干净边集 E1 所需最少的搜索者数.

对于图 G 不难得到：$s(P,E1) \leqslant |V(E1) \cap V(E-E1)|$.

引理 1. 给定图 $G=(V,E)$，$P_1=e_1,e_2,\cdots,e_n$，$P_2=e_n,\cdots,e_2,e_1$，$n=|E|$，则 $s(P_1)=s(P_2)$.

证明：设 P_1 在搜索边 $e_i(1 \leqslant i \leqslant n)$ 时所需搜索者个数首次达到最大. 即：$E_i=\{e_1,e_2,\cdots,e_{i-1}\}$，$s(P_1,E_i)<s(P_1,E_i \cup \{e_i\})=s(P_1,E)=s(P)$. 分三种情况：

情况 1：$e_i=vv'$ 两顶点均为原始顶点(图 1)，$v_1,v_2,\cdots,v_m$ 是混合顶点，$m=|V(E1) \cap V(E-E_1)|$. (1)若 $d(v)>1$，$d(v')=1$，则 e_i 干净后 v 为混合顶点，v' 为干净顶点，且：$s(P_1,E_i \cup \{e_i\})=s(P_1,E_1)+1=m+1=s(P_1)$. 在按 P_2 搜索图 G 且形成干净边集 $E_2=\{e_n,\cdots,e_{i+1}\}$ 时，混合顶点为 $v_1,v_2,\cdots,v_m,v$. 故 $s(P_2,E_2) \geqslant s(P_1)$. (2)若 $d(v)>1$，$d(v')>1$，则 e_i 干净后 v,v' 均成为混合顶点，且 $s(P_1,E_1 \cup \{e_i\})=m+2=s(P_1)$. 在按 P_2 搜索图 G 且形成干净边集 $E_2=\{e_n,\cdots,e_{i+1}\}$ 时，混合顶点必为：$v_1,v_2,\cdots,v_m,v,v'$. 故：$s(P_2,E_2) \geqslant s(P_1)$. 故对于情况 1 总满足 $s(P_2) \geqslant s(P_1)$. 类似地可以证明如下情况 2,3 均满足 $s(P_2) \geqslant s(P_1)$.

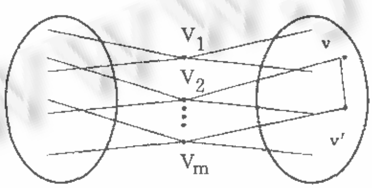

图1

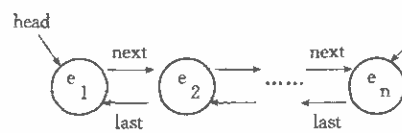

图2

情况 2：e_i 两顶点为原始顶点与混合顶点.

情况 3：e_i 两顶点均为混合顶点.

因 P_1 与 P_2 互为逆序，故 $s(P_1)=s(P_2)$.

由该引理我们引入双向链表构成搜索方案(图 2). 利用 next 指针和 last 指针链接连续搜索的边. 于是将树 T 的信息记录扩展为 4 元组(type,s,plan,MI－info)，仍记为 info(T). type,s 与前相同；plan 是二元组(head,tail)，head,tail 为指向树 T 搜索方案链表头和尾的指针；MI－info 当 type＝E,H,M 时意义与前相同，当 type ＝I时 MI－info＝(φ,0,(headI,tailI),nil). 因 I 型树是由两棵 E 型子树合并而成，因此 headI,tailI 指向两个链表的连接处.

(1)若 $d(r)=1$ 且 T 仅为一条边，则 info(T)＝(E,1,(head,tail),nil)，head,tail 均指向链表的唯一边. 否则将树根移至 r 的唯一邻顶 r′得 T′，将 T′分裂为 r′r 和 T′－r′r，info(T)＝re_root(merge(info(T′－r′r),info(r′r))).

(2)若 $d(r)>1$，计算方法同前.

re_ root 过程的步骤(1),(2),(4)同前. 步骤(3)修改如下：设 info(T′)＝(I,s′,plan′,(φ,0,(headI′,tailI′),nil)). 若 s′＝1，则 info(T)＝(E,1,plan′,nil)；若 s′>1，则 info(T)＝(M,s′,plan′,(E,1,(headI′,headI′),nil)).

下面给出算法 merge，我们重点描述两棵子树搜索方案如何合并.

(1)对于情况(1)：s1＝s2,type1＝type2＝H；情况(2)：s1＝s2,type1＝E,type2＝H；情况(5)：s1＝s2,type1＝M 或 type1＝type2＝I 或 type1＝I,type2＝E；情况(6.1)：s1>s2, type1=H,E,两个链表首尾相接,即做如下动作：

tail1. next=head2;head2. last=tail1;plan=(head1,tail2).

(2)对于情况(3):s1=s2,type1=type2=E,颠倒链表 2 再与链表 1 链接：

tail2. next=tail2. last;head2. last=head2. next;head2. next=nil;

tail1. next=tail2;tail2. last=tail1;plan=(head1,head2);

MI−info=(φ,0,(tail1,tail2),nil).

(3)对于情况(4):s1=s2,type1=I,type2=H; 情况(6.2):s1>s2,type1=I,断开链表 1 再与链表 2 链接：

headI1. next=head2;head2. last=headI1;tail2. next=tailI1;

tailI1. last=tail2;plan=(head1,tail1);MI−info=(φ,0,(tail2,tailI1),nil).

(4)对于情况(7):s1>s2,type1=M,设 T_1 的 M−树搜索方案的头尾指针为 headM1, tailM1. 引入指针 headQ = headM1. last; tailQ = tailM1. next. info (T′) =(type′,s′,(head′,tail′),MI−info′)=merge(MI−info1,info(T_2)). 若 s′=s1,则只需将 T_1 和 T_2 的搜索方案首尾相接. 若 s′<s1,如下计算：

headQ. next=head′;head′. last=headQ;tail′. next=tailQ;

tailQ. last=tail′;MI−info=info(T′).

引理 2. 对于树 T,算法同时得到搜索数 s(T)和搜索方案 P(T),且 s(P(T))=s(T).

证明:由文献[1]知该算法得到的搜索数恰为 s(T),只需证按 P(T)搜索树 T 恰需要 s(T)个搜索者. 首先考虑算法 re−root,若 T′为 H,E,M 型,正确性显然;若 T′为 I 型,显然 info (T′)是由 r′r 和 T′−r′r 合并而来,易知 T′−r′r 也为 I 型且 s(T′−r′r)>s(r′r). 由 merge 情况(6.2)知 headI′恰指向边 r′r. 再考虑算法 merge. 对于情况(1),(5)两个链表首尾相接,正确性显然. 对于情况(2),算法步骤(1)保证根 r 首次成为混合顶点时,只有 r 的搜索数小于 s1 的分支未搜索,因此搜索 T_1 只需 s1 个搜索者,再搜索 T_2 所需搜索者数不超过 s2. 情况(6.1)的正确性由情况(1),(2)保证. 对于情况(3),由情况(2)分析知先搜索 T_1 需 s1 个搜索者. 由引理 1 知再搜索 T_2 仍需 s1=s2 个搜索者. 由情况(2),(3)的分析易得情况(4),(6.2)得到搜索方案的正确性. 对于情况(7),由 T_1 去掉其 M−树,则是一棵 I 型树. 若 s1>s′,则需将 T_1 的 M−树与 T_2 合并的结果 T′插入 I 型树中间搜索,需 s1 个搜索者,若 s1=s′,则先搜索 T_1,再搜索 T_2,需 s1+1 个搜索者. 该算法同时计算树的搜索数和搜索方案,每一步仅比算法 compute−info 多用 O(1)时间. 因此时间复杂度仍为 O(nlog(n)).

定理. 树的搜索数和搜索方案可在线性时间内同时得到.

证明:对于 M 型树,算法 re_root 和 merge 皆沿着其 MI_info 链递归计算. 我们引入双向链表表示该链,其中引入如下指针:(1)指向 T 的 M−树的信息记录的指针. 若 M−树的搜索数恰比 s(T)小 1,该指针标记为 t,否则标记为 l. 其反向指针标记为 $\bar{t}$,$\bar{l}$. (2)表示 t 和 l 闭包的指针记为 t^* 和 l^*. (3)由 T 指向 MI−info 链尾的指针标记为 m^*.

算法 re_root 由 m^*指针一步即可完成. 下面给出减少算法 merge 递归次数的方法. 设 T_1 为 M 型树,t^*指针指向树 T^*的信息记录. (1)若 s(T^*)≤s(T_2). 则在 T_1 与 T^*间存在树 T_3 满足 s(T_3)=s(T_2). 若 T_3,T_2 合并属于情况(1−4),则 T_3=T^*. 只需将 T^*,T^2 的链表合并后插入 T^*的位置. 若 T_3,T_2 合并属于情况(5),则 T_1,T_2 合并后是 H 型树,只需将 T_1,T_2

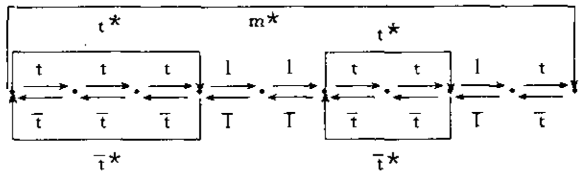

图3

的链表首尾相接即可.(2)若 $s(T^*)>s(T_2)$.首先利用 m^* 指针检查链尾信息记录,若其搜索数比 $s(T_2)$ 大,则只需将两树合并的链表插入原来位置,信息记录不变.不然则在 T_1 的 MI—info 链上确定两棵子树 T_3 和 $\overline{T}_3$. T_3 是第一个满足 $s(\overline{T}_3)\leqslant s(T_2)$ 的树,$\overline{T}_3$ 为最后一个满足 $s(T_3)>s(T_2)$ 且无 t 指针指向 $\overline{T}_3$ 的树.这样的树可首先沿 m^* 指针再沿 t 和 l 指针在 $O(s(T_2))$ 时间内得到.再调用 merge 将 T_3,T_2 同情况(1)那样合并.

于是执行 merge 所需时间 $m(s(T_1),s(T_2))$ 满足:$m(s(T_1),s(T_2))\leqslant m(s(T_2),s(T_2))+O(s(T_2))$.因 T3 在最后一种情况满足 $s(T3)\leqslant s(T_2)$.且有:$m(s(T_1),s(T_2))\leqslant m(s(T_1)-1,s(T_2))+O(1)$,因此可得:$m(s(T_1),s(T_2))\leqslant O(s(T_2))$,于是:$S(T)\leqslant S(T_1)+S(T_2)+O(\min\{s(T_1),s(T_2)\})\leqslant S(T_1)+S(T_2)+O(\log(\min\{|T_1|,|T_2|\})$.由此得 $S(T)=O(|T|)=O(n)$.

参考文献

1 Megiddo N, Hakimi S L, Garey M R *et al*. The complexity of searching a graph. J. ACM, 1988;**35**(**1**).

2 Parsons T D. The search of a connected graph. In: Proc. 9th Southeastern Conf. on Combinatorics, Graph Theory and Computing, 1978:549—554.

3 LaPaugh A S. Recontamination does not help. Manuscript, 1982.

A LINEAR ALGORITHM ON TREE FOR A CLASS OF CLEARING CONTAMINATION PROBLEMS

Zhu Daming and Ma Shaohan

(*Department of Computer Science*, *Shandong University*, *Jinan* 250100)

Abstract The Graph Search problem is proved to be NP—complete by MEGIDDO et al. An algorithm for tree is also proposed by them which computes the search number in O(n) time and the search plan in O(nlog(n)) time. This paper developes a linear algorithm through representing a search plan by edge sequence, which computes both the search number and the search plan in O(n) time.

Key words Algorithm, NP—complete, tree, undirected connected graph.