An $O(k^2n^2)$ Algorithm to Find a k-Partition in a k-Connected Graph[1]

Ma Jun (马军) and Ma Shaohan (马绍汉)

Department of Computer Science, Shandong University, Jinan 250100

Received December 16, 1991; revised August 14, 1993.

Abstract

Although there are polynomial algorithms of finding a 2-partition or a 3-partition for a simple undirected 2-connected or 3-connected graph respectively, there is no general algorithm of finding a k-partition for a k-connected graph $G = (V, E)$, where k is the vertex connectivity of G. In this paper, an $O(k^2n^2)$ general algorithm of finding a k-partition for a k-connected graph is proposed, where $n = |V|$.

Keywords: Graph algorithm, graph vertex connectivity, k-partition of a graph.

1 Introduction

Let $G = (V, E)$ be a simple undirected graph. V and E represent the vertex set and the edge set of G respectively, $n = |V|$ and $m = |E|$. Two vertices u, v are said to be adjacent if the edge $e = (u, v) \in E$. The number of edges incident to a vertex $v \in V$ is called the degree of v. G is connected if for any $u, v \in V$, there is a path in G joining u and v; otherwise G is unconnected. A vertex cut of a connected graph G is a subset V' of V such that $G - V'$ becomes disconnected, where $G - V'$ is the subgraph of G gained by deleting all vertices of V' and those edges incident to the vertices of V' from G. G is called k-connected graph if the minimum vertex cut of G consists of k vertices of G, while k is called the connectivity of G and noted as $k(G)$. If G is a complete graph, $k(G)$ is defined as $n - 1$. Throughout this paper we assume that $V = \{1, 2, \cdots, n\}$, and N, R represent the natural number set and the real number set respectively.

Let G' be a subgraph of G. We use the symbols $V(G')$ and $E(G')$ to denote the vertex set and edge set of G'. A subtree of G is a connected subgraph of G without cycles in it. Let V' be a subset of V. The subgraph of G whose vertex set is V' and whose edge set is the set of those edges of E incident to the vertex of V' is called the subgraph of G induced by V' and is denoted by $G\lceil V' \rceil$. The problem of finding a k-partition of G is described as follows.

Input (1) $G = (V, E)$: a simple undirected graph;
(2) $a_1, a_2, \cdots, a_k$: k different vertices of G;
(3) $n_1, n_2, \cdots, n_k$: k positive integers such that $n_1 + n_2 + \cdots + n_k = n$.

Output $(V_1, V_2, \cdots, V_k)$, for all i, $1 \le i \le k$, such that
(1) $a_i \in V_i$;
(2) $G\lceil V_i \rceil$ is a connected subgraph of G;
(3) $|V_i| = n_i$ and $V_i \cap V_j = \phi$, for $i \ne j$.

[1]The research was supported by National Natural Science Foundation of China.

An example of 3-partition of a 3-connected graph is given in Fig.1, and the applications of the k-partition of a graph G are given in [1].

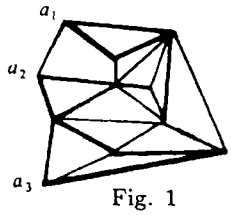

Fig. 1

Obviously, it is not always the case that for any graph G and for a $k \in N$ $(k > 1)$, a k-partition of G exists and it has been proved that for a general graph G, to find a k-partition of G is an NP complete problem[2]. In this paper, we limit G to be a k-connected graph. Györi[3] and Lovász[4] proved that if G is a k-connected graph, a k-partition of G exists. But from their proofs, only a polynomial algorithm for $k = 2$ can be induced. Suzuki *et al.* presented an $O(n^2)$ algorithm for 3-connected graph in 1990[5]. But the general polynomial algorithm for finding a k-partition for a k-connected graph has not been found. In this paper, we propose an $O(k^2n^2)$ general algorithm to find a k-partition in a k-connected graph.

2 k-Partition Algorithm for a k-Connected Graph

Lemma 1[6]. *A graph $G = (V, E)(|V| \geq k + 1)$ is a simple undirected graph and $k \in N$. G is a k-connected graph iff for any vertices $w, v \in V$, there are k paths in G joining w, v without intersecting inner vertex.*

Corollary 1. *If G is a k-connected graph and $a_1, a_2, \cdots, a_k$ are k different vertices of G, for any vertex $v \in V - \{a_1, a_2, \cdots, a_k\}$ and $a_i (1 \leq i \leq k)$, there is at least a path P in G joining a_i and v with no inner vertex of P belonging to $\{a_1, \cdots, a_k\}$.*

It is obvious that for given k different vertices $a_1, a_2, \cdots, a_k$ and k positive integers $n_1, n_2, \cdots, n_k$ such that $n_1 + n_2 + \cdots + n_k = n$, if we can find k subtrees $T_1, T_2, \cdots, T_k$ of G such that,

$$a_i \in V(T_i) \quad i = 1, 2, \cdots, k; \tag{1}$$

$$|V(T_i)| = n_i \text{ and } V(T_i) \cap V(T_j) = \phi, \text{ for } i \neq j \quad 1 \leq i, j \leq k. \tag{2}$$

the k vertex sets of $V(T_1), V(T_2), \cdots, V(T_k)$ form a k-partition of G. The idea of our algorithm is to find such k subtrees in a k-connected graph.

In the following discussion, if a vertex $v \in V(T_i), 1 \leq i \leq k$, v is called a tree vertex; otherwise, it is called an untree vertex. The value of function P at v is called the P value of v for short. Two subtrees T_i and T_j are adjacent if there exists an edge $e = (u, v)$ such that $u \in V(T_i)$ and $v \in V(T_j), i \neq j$.

The technique used in our algorithm are similar to these used in the algorithm of finding the maximum flow in a network[7] with little change. We use a vertex flow function P to control the process of expanding k subtrees, and we define the vertex flow function $P: V \rightarrow R$ as follows:

$$P(v) = 0, \text{ if } v \text{ is an untree vertex;} \tag{3}$$

$$P(v) = n_i / |V(T_i)|, \text{ if } v \in V(T_i). \tag{4}$$

Clearly, the P value of every vertex of T_i is the same, $1 \leq i \leq k$. It decreases only if more vertices are added to T_i.

Our algorithm is a dynamic one. First it lets T_i consist of a_i, and the P value of a_i be n_i. Then it expands the subtree $T_i (1 \leq i \leq k)$ in cycles. T_i satisfies that:

- there is untree vertices adjacent to the vertices of T_i, or
- there is subtree T_j adjacent to T_i and the P value of vertices of T_j is smaller than that of T_i.

T_i is called the subtree which can be expanded. The process will not terminate until the P value of every tree vertex becomes 1. In fact for every vertex $v \in V$, v becomes a tree vertex. If this process ends, clearly, $T_1, T_2, \cdots, T_k$ must satisfy Eqs. (1) and (2). The formal algorithm is as follows.

Algorithm. k-partition

Input: (1) $G = (V, E)$: the adjacency matrix of a k-partition graph;
(2) $k : k = k(G)$;
(3) $a_1, a_2, \cdots, a_k$: k different vertices of G;
(4) $n_1, n_2, \cdots, n_k$: k positive integers such that $n_1 + n_2 + \cdots + n_k = n$

Output: A k-partition of G.

The auxiliary data structure are:

(1) Tree-node[1..n]: array of set of $\{a_1, a_2, \cdots, a_k\}$. If $a_j \in$ Tree-node[i], it means that the vertex i is or has been a vertex of T_j.

(2) $P\lceil 1..n \rceil$: array of real, $0 \leq P\lceil i \rceil \leq n, 0 \leq i \leq n$. $P\lceil i \rceil$ is the value of function P at vertex i.

In the process of expanding k subtrees, if $v \in V(T_i)$, we use $Td(v, i)$ to represent the degree of v in T_i.

Step 1. //Initialization//
1 input the adjacency matrix of G;
2 $T_1 = \{a_1\}$; $T_2 = \{a_2\}$; $\cdots$; $T_k = \{a_k\}$; //subtree T_i has one vertex a_i as its root//
3 Tree-node$\lceil a_i \rceil := \{a_i\}$, for $1 \leq i \leq k$;
4 Tree-node$\lceil j \rceil := \phi$ for $j \in V - \{a_1, a_2, \cdots, a_k\}$;
5 $P\lceil a_i \rceil := n_i$, for $1 \leq i \leq k$;
6 $P\lceil j \rceil := 0$ for $j \in V - \{a_1, a_2, \cdots, a_k\}$;
7 $i := 1$; //i is the index of subtree T_i to be expanded//
Step 2. //Calculation//
8 while $\sum_{j=1}^{k} |V(T_j)| < n$ do
9 if $P\lceil a_i \rceil > 1$ then
10 $auxv := \{v | v$ does not belong to $V(T_i) \cup \{a_1, a_2, \cdots, a_k\}$ and v is adjacent to a vertex of $T_i\}$;
11 $NEW := \{v | v \in auxv$ and Tree-node$[v] = \phi\}$;
12 $OLD := \{v | v \in auxv$ and Tree-node$[v] \neq \phi\}$;
13 if $NEW \neq \phi$ then
14 if $NEW > (n_i - |V(T_i)|)$ then
15 choose $(n_i - |V(T_i)|)$ vertices of NEW randomly, add those vertices to T_i;
16 for every vertex v of T_i do
17 (1) Tree-node$[v]$:=Tree-node$[v] \cup \{a_i\}$;
18 (2) $P[v] := 1$;
19 endfor
20 else
21 add all vertices of NEW to T_i;
22 for every vertex v of T_i do

```
23              (1) Tree-node[v]:=Tree-node[v] ∪ {a_i};
24              (2) P[v] := n_i/|V(T_i)|;
25            endfor
26          endif
27        else
28          OLD1 := {v|v ∈ OLD and a_i does not belong to Tree-node[v]};
29          if OLD1 = φ then goto line 54; endif;
30          OLD2 := {v|v ∈ OLD1 and P[v] = the minimum value of p for
              the vertices in OLD1};
31          if p[v] ≥ p[a_i], for a v ∈ OLD2 then goto line 54; endif; //there is no subtree
              adjacent to T_i whose vertex function P value is less than that of T_i//
32          for t := 1 to k − 1 do
33              j := (i + t) mod (k + 1); //j is an index of a_j//
34              if there is a vertex v ∈ OLD2 and a_j ∈ Tree-node[v] then
35                exit the for loop; endif;
36          endfor;
37          OLD3 := {v|v ∈ OLD2, a_j ∈ Tree-node[v]}; // OLD3 is the vertex set of the
              subtree T_j adjacent to T_i with the minimum vertex flow function value//
38          find a vertex w ∈ OLD3 and Td(w, j) = min_t ∈OLD3 Td(t, j);
39          if Td(w, j) = 1 then
40            delete w from T_j and add w to T_i;
41            Tree-node[w] := Tree-node[w] ∪ {a_i};
42          else
43            cut off the subtree T' rooted at w in T_j;
44            Tree-node[w] := Tree-node[w] ∪ {a_i};
45            add w to T_i;
46            for all v, v ∈ T' − {w}
47                Tree-node[v] := φ; p[v] := 0;
48            endfor
49            for every v ∈ T_i do P[v] := n_i/|V(T_i)| endfor;
50            for every v ∈ T_j do P[v] := n_j/|V(T_j)| endfor;
51          endif
52        endif
53      endif
54  i := i mod k + 1; //expanding next subtree//
55  endwhile
Step 3. Output V(T_1), V(T_2), ···, V(T_k);
Step 4. Stop
```

An implementation of k-algorithm is shown in Fig. 2.

Lemma 2. *After $2k$ loops of Step 2 in the implementation of k-partition algorithm, the number of tree vertices will increase.*

Proof. Let $S_i = \sum_{j=1}^{k} |V(T_j)|$, where $i \in N$ and S_i represents the numbers of k subtrees vertices at the beginning of the i-th loop in Step 2 ($i = 1, 2, \cdots$). To avoid confusion, let T_i always represent the subtree being expanded in the implementation of k-partition algorithm. Because $S_1 = k(< n)$ and Corollary 1, at the beginning of the i-th ($i = 1, 2, \cdots$) loop, $auxv \neq \phi$. If $NEW \neq \phi$ in line 13, all or a part of untree vertices of NEW are added to T_i, otherwise, because $auxv \neq \phi$, $OLD \neq \phi$. The vertices in OLD are the vertices of subtrees adjacent to T_i. We use the set $OLD1$ to store the vertices that are in OLD and have not been the vertices of T_i before. It is implemented by checking if a_i is in Tree-node[v] in line 23. The purpose of using array Tree-node is to prevent from more than two times

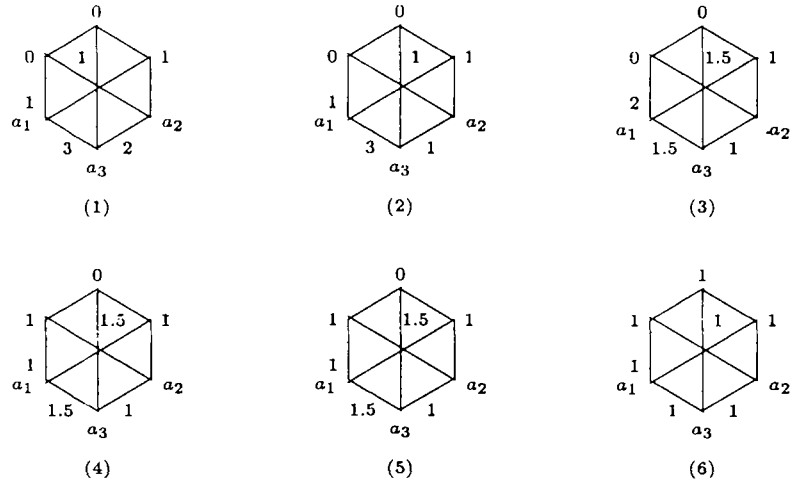

Fig. 2. The bold lines describe the process of expanding T_i, where $n_1 = 2$, $n_2 = 2$, $n_3 = 3$, the number near vertex v is the value of function P at v, the number i in the brackets represents the i-th cycle of Step 2 of the partition algorithm.

exchanging a vertex v between T_i and T_j, $i \neq j$. If $OLD1 = \phi$, T_i cannot be expanded, go to the $(i+1)$-th loop of Step 2 directly; otherwise we use $OLD2$ to store the vertices with the minimum P value in the set $OLD1$. If for a $v \in OLD2$, $P(v) > P(a_i)$, that is, the P value of the vertices of subtrees adjacent to T_i is greater than the P value of vertices of T_i, we give up expanding T_i, go to the $(i+1)$-th loop of Step 2. Because we expand subtrees in cycles, if $\sum_{j=1}^{k} |V(T_j)| < n$, at least one of the subtrees can be expanded will be found after $O(k)$ null loop of Step 2. Let us also use T_i to represent the subtree being expanded. If there are some untree vertices being added to T_i in line 13-26, the number of tree vertices increases, otherwise, because T_i is a subtree that can be expanded, there are some subtrees adjacent to T_i, the P values of the vertices of these subtrees are smaller than the P value of the vertices of T_i. We use the set $OLD3$ to store the vertices of T_j, where T_j is the subtree with the minimum P value in the subtrees adjacent to T_i and if there are more than two subtrees with the minimum value of P, the index of j is the first index in the order of $i+1, i+2, \cdots, i+k, 1, 2, \cdots, i-1$. This is because we want to expand k subtrees evenly. In line 38 we find a vertex w, such that, w is of the minimum degrees in the vertices of T_j which are adjacent to the vertices of T_i. If $T(w, j) = 1$, w is a leaf of T_j, in lines 39-41, we delete the w from T_j and add w to T_i; otherwise cut off the subtree T' rooted at w of T_j, add w to T_i, and let the P value of the rest of vertices of T' be zero. Clearly, only in the last case, the number of tree vertices decreases. But if we noted that in the next following k loops of Step 2, the rest vertices of T' can be added to T_j or to other subtrees, because T_j becomes a subtree which needs to be expanded and can be expanded. So we know that after $2k$ loops of Step 2, the number of tree vertices will not decrease. In addition, we expand k subtrees in cycles, and if $\sum_{j=1}^{k} |V(T_j)| < n$, there must be some untree vertices adjacent to a subtree T_b, such that, the P values of the vertices of $T_b > 1$. During $2k$ loops, T_b can be expanded, and when T_b is expanded, according to the implementation of our algorithm, some untree vertices will be added to T_b in lines 13-26. This means that the number of tree vertices increases truely after $2k$ loops of Step 2 in the implementation of k-partition algorithm, so Lemma 2 is correct. □

Theorem 1. *k-partition algorithm can find a k-partition in a k-connected graph correctly in $O(knm)$ time.*

Proof: In Step 1, T_i consists of one vertex a_i as its root, and $P[a_i] = n_i$; $P[v] = 0$, for $v \in V - \{a_1, a_2, \cdots, a_k\}$, $1 \leq i \leq k$. Clearly, Step 1 can be implemented in $O(n)$.

Because $S_1 = k$ and the number of untree vertices at the beginning of Step 2 is $n-k$, based on Lemma 2, it is obvious that after at most $2k(n-k)$ loops of Step 2, $\sum_{j=1}^{k} |V(T_j)| = n$. Because the inequatilies $|V(T_i)| \leq n_i$ $(1 \leq i \leq k)$, are unchanged in the implementation of k-partition algorithm, when $\sum |V(T_j)| = n$, it must be $|V(T_i)| = n_i (1 \leq i \leq k)$. Because in the implementation of our algorithm, subtree T_i keeps its connectivity and never a vertex v is a vertex of both T_i and T_j, $i \neq j$, so k subtrees $T_1, T_2, \cdots, T_k$ obtained in Step 3 satisfy (2.1), (2.2). So our algorithm is correct.

Because every sentence in Step 2 can be implemented in $O(m)$ and Step 3 can be implemented in $O(m)$, based on Lemma 2, the number of loops in Step 2 is less than or equal to $2k(n-k)$, so the time complexity of k-partition is $O(knm)$. □

3 Conclusion

We have coded k-partition algorithm in Pascal. The implementation of k-partition shows that the running time of k-partition relies on the distribution of $a_1, a_2, \cdots, a_k$ in G strongly and that in most cases the number of loops in Step 2 $\ll 2k(n-k)$. H. Nagamochi and T. Ibaraki gave an $O(m)$ algorithm which could find a sparse k-connected spanning subgraph $G' = (V, E')$, such that, $|E'| = O(kn)$ for any k-connected graph[8]. If we use this algorithm as the preprocessing procedure of the k-partition algorithm, that is, first we find G' by the algorithm of [8], and then let k-partition algorithm run on G'. It is obvious that the time complexity of the combined algorithm becomes $O(k^2n^2)$.

References

[1] Manabe I. Building networks with some fixed routing and the ability to resist some obstacles. Electronic Information Communication Research Materials (in Japanese), COMP 86-70, 1987: 95-105.

[2] Dyer M E, Frieze A M. On the complexity of partitioning graphs into connected subgraphs. Discrete Appl Math, 1985, 10: 139-153.

[3] Győri E. On division of graph to connected subgraphs, combinatorics. In: Proc Fifth Hungarian Combinatorial Coll, Keszthely, Bolyai: North-Holland, 1978, 485-494.

[4] Lovász L. A homology theory for spanning trees of a graph. Acta Math Acad, 1977, 30: 241-251.

[5] Hitoshi S, Naomi T, Takao N, Hiroshi M, Shuichi U. An algorithm for tripartitioning 3-connected graphs(in Japaneses). J of Infomation Processing, 1990, 30(5): 584-592.

[6] Swamy N N S, Thulasiraman K. Graphs, networks and algorithms. John Wiley & Sons Inc, 1981, 143.

[7] Even S. The max flow algorithm of Dinic and Karzanov, an exposition. MIT/LCS/TM-80, 1976.

[8] Nagamochi H, Ibaraki T. A linear-time algorithm for finding a Sparse k-connected spanning subgraph of a k-connected graph. Algorithmica, 1992, 7: 583-596.