Annals of Pure and Applied Mathematics
Vol. 2, No. 1, 2012, 45-59
ISSN: 2279-087X (P), 2279-0888(online)
Published on 18 December 2012
www.researchmathsci.org

Annals of
Pure and Applied Mathematics

An Optimal Algorithm to Find a Minimum 2-neighbourhood Covering Set on Cactus Graphs

Kalyani Das[1] and Madhumangal Pal[2]

[2]Department of Applied Mathematics with Oceanology and Computer Programming, Vidyasagar University, Midnapore – 721102, India.
e-mail: mmpalvu@gmail.com

[1]Department of Mathematics, Ramnagar College,
Depal, Purba Medinipur – 721453, India.

Received 2 Semptember2012; accepted 16 December 2012

Absttact. A cactus graph is a connected graph in which every block is either an edge or a cycle. An optimal algorithm is presented here to find minimum cardinality 2-neighbourhood-covering set on cactus graphs in $O(n)$ time, where n is the total number of vertices of the graph. The cactus graph has many applications in real life problems, specially in radio communication system.

Keywords: *Design of algorithms, analysis of algorithms, 2-neighbourhood-covering set, cactus graph.*

AMS Mathematics Subject Classifications (2010): 05C85

1. Introduction

1.1. Cactus graph

Let $G = (V, E)$ be a finite, connected, undirected, simple graph of n vertices and m edges, where V is the set of vertices and E is the set of edges. A vertex v is called a *cut-vertex* if removal of v and all edges incident on v disconnect the graph. A *non-separable graph* is a connected graph which has no cut-vertex and a *block* means a maximum non-separable sub-graph. A block is a *cyclic block* or simply a *cycle* in which every vertex is of degree two.
A *cactus graph* is a connected graph in which every block is either an edge or a cycle.

Cactus graph has many applications. These graphs can be used to model physical setting where a tree would be inappropriate. Examples of such setting arise in telecommunications when considering feeder for rural, suburban and light urban regions [8] and in material handling network when automated guided vehicles are used [9]. Moreover, ring and bus structures are often used in local area networks. The combination of local area network forms a cactus graph. Design a suitable algorithms is a very important task. Several algorithms have been design for different graphs, see [16-27].

1.2. The k-neighbourhood-covering set

The k-neighbourhood-covering (k-NC) problem is a variant of the domination problem. Domination is natural model for location problems in operations research, networking, etc.

The graph considered in this paper are simple *i.e*, finite, undirected and having no self-loop or parallel edges. In a graph $G=(V,E)$, the *length* of a path is the number of the edges in the path. The *distance* $d(x,y)$ from the vertex x to the vertex y is the minimumn length of a path from x to y, and if there is no path from x to y then $d(x,y)$ is taken as ∞.

A vertex *x*, k-dominates another vertex y if $d(x,y)\le k$. A vertex z is k-NC of an edge (x,y) if $d(x,z)\le k$ and $d(y,z)\le k$ *i.e*, the vertex z k-dominates both x and y. Conversely if $d(x,z)\le k$ and $d(y,z)\le k$ then the edge (x,y) is said to be k-neighbourhood covered by the vertex z. A set of vertices $C\subseteq V$ is a k-NC set if every edge in E is k-NC by some vertices in *C*. The k-NC number $\rho(G,k)$ of G is the minimum cardinality of all k-NC sets.

1.3. Review of previous works

For $k=1$, Lehel et al. [10] have presented a linear time algorithm for computing $\rho(G,1)$ for an inteval graph G. Chang et al. [1] and Hwang et al. [7], have presented linear time algorithms for computing $\rho(G,1)$ for a strongly chordal graph provided that strong elimination ordering is known. Hwang et al. [7] also proved that k-NC problem is NP-complete for chordal graphs. Mondal et al. [12] have presented a linear time algorithm for computing 2-NC problem for an interval graph. Also a linear time algorithm for trapezoid graph has presented by Ghosh et al. [4].

1.4. Our result

In this paper we consider a cactus graph G. Here we design an algorithm which finds the 2-neighbourhood covering set of the graph G in $O(n)$ time. The algorithm also takes $O(n)$ space.

2. Computation of blocks and cutvertices of G

As described in [13] the blocks as well as cut vertices of a graph G can be determined by applying DFS technique. Using this technique we obtain all blocks and cut vertices of the cactus graph $G=(V,E)$. Let the blocks be B_1, B_2, B_3,..., B_N and the cut vertices be c_1, c_2, $c_3,\ldots,c_R$ where N is the total number of blocks and R is the total number of cut vertices.

Two blocks are said to be *adjacent* if they have at least one common vertex of the graph G. Define edge blocks as $e_i = \{B_j : |B_j| = 2\}$, and cycle blocks as $C_i = \{B_j : |B_j| > 2\}$, where $|B_j|$ is the cardinality of B_j. Let the number of cycle blocks be N' and number of edge blocks be N''. Number of vertices of each cycle is denoted by $|C_i|$, $i = 1,2,\ldots,N'$.

3. Construction of tree blocks and the tree T_{BC}

Suppose the set $S' = \{e_1, e_2, e_3, \ldots, e_{N''}\}$. A tree block T_i is a maximal subgraph of G such that T_i is a tree. Let $T_1, T_2, \ldots, T_L$ be the tree blocks of *G*. The tree blocks T_i's are formed by the members of S' *i.e.,* $T_i \subset S'$, $i = 1,2,\ldots,L$. Now we have in a position to construct the tree T_{BC} using tree blocks T_i's, $i = 1,2,\ldots,L$ and C_j's, $j = 1,2,\ldots,N'$. Before construction of the tree T_{BC} we define *an intermediate graph* G' whose vertices are the blocks of G.

Thus $G' = (V', E')$ where the vertices are blocks of the graph G *i.e.,* $V' = \{T_1, T_2, \ldots T_L, C_1, C_2, C_3, \ldots, C_{N'}\}$.

If two blocks are adjacent they are connected by an edges. Thus $E' = \{(C_i, C_j)$ or (C_i, T_k) or $(T_k, T_l) : i \neq j; i, j = 1,2,\ldots,N'$ and $k \neq l; l, k = 1,2,\ldots,L$, C_i, C_j, T_l and T_k are adjacent blocks $\}$.

Now the tree T_{BC} is constructed from G' as follows:

We discard some suitable edges from G' in such a way that the resultant graph becomes a tree. The procedure for such reduction is given below:

Let us take any arbitrary vertex of G', containing at least two cut-vertices of G, as root of the tree T_{BC} and mark it. All the adjacent vertices of this root are taken as children of level one and mark them. If there are edges between the vertices of this level, then discard these edges. Each vertices of level one is considered one by one to find the vertices which are adjacent to them but unmarked. These vertices are taken as children of the corresponding vertices of level one and put them at level two. These children at level two are marked and if there be any edge between them then remove them. This process is continued until all the vertices are marked.

Thus the tree $T_{BC} = (V', E'')$ where $V' = \{T_1, T_2, \ldots T_L, C_1, C_2, C_3, \ldots, C_{N'}\}$ and $E'' \subset E'$ is obtained. For convenience, we refer the vertices of T_{BC} as nodes. We note that each node of this tree is a block (cycle block or tree block) of the graph $G = (V, E)$.

The parent of the node C_i in the tree T_{BC} will be denoted by *Parent(*C_i*)*.

4. Euler Tour

Euler tour produces an array of nodes. The tour proceeds with a visit to the root and there after visits to the children of the root one by one from left to right returning each time to the root using tree edges in both directions. Algorithm GEN-COMP-NEXT of Chen et al. [2] implements this Euler tour on a tree starting from the root. The input to the algorithm is the tree represented by a `parent of' relation with explicit ordering of the children. The output of the algorithm is the tour starting from the root of the tree and ending also at the root. The tour is represented by an array $S(1:2(N'+L)-1)$ that stores information connected to the visits during the tour. The element $S(i)$ of the array S is a record consisting of two fields, one of which, denoted by $S(i).node$, is the node visited during the ith visit while the other, denoted by $S(i).subscript$ is the number of times the node $S(i).node$ is visited d using the first i visits of the tour. Two fields of an element of S are written together using the notation $(node)_{subscript}$.

Also, we consider an array $f(j)$ and $f'(k)$ which stores the total number of occurrence of the block C_j, $j=1,2,3,\dots,N'$ and T_k, $k=1,2,3,\dots,L$ in the array $S(i), i=1,2,3,\dots,2(N'+L)-1$. Thus $f(j)$ and $f'(k)$ represents the total number of visits of the block C_j and T_k in the Euler tour, *i.e.*, $f(j)$ and $f'(k)$ are the maximum subscript of C_j and T_k in the array $S(i)$.

For each j and k, j=1,2,3,...,N and k=1,2,3,...,L, $(C_j)_{f(j)}$ and $(T_k)_{f'(k)}$ occurs only once in th array S(i) and before $(C_j)_{f(j)}$ and $(T_k)_{f'(k)}$ all of $(C_j)_1$, $(C_j)_2$,...$(C_j)_{f(j)-1}$ and $(T_k)_1$, $(T_k)_2$,...$(T_k)_{f'(k)-1}$ occur respectively in order of increasing subscripts of C_j and T_k.

The following important lemma is proved in [11].

Lemma 1. *If* $S(i).subscript=1$ *and* $S(i+1).subscript \neq 1$, *then* $S(i).node$ *is a leaf node of the tree.*

5. Determination of 2-NC set from cycles and paths

Lemma 2. *For 2-NC problem a vertex in a cactus graph can cover at least 4 edges.*

Proof: Let $G=(V,E)$ be a cactus graph and $u \in V$. Now degree of u may be two or more.

Case 1: Let the degree of u be two. Then there exist two vertices v and w so that $d(u,r)=1$, $r=v,w$. Now if v,w are of degree two then there also exist two vertices x,y so that $d(u,x)$ and $d(u,y)=2$. Thus u covers four edges $(u,v),(u,w),(w,x),(v,y)$ where v,w are adjacent to u and x,y are adjacent to v and w respectively.

But if any one of v and w are of degree more than two then u cover all edges incident on v or w or both as $d(u,r)=1$, $r=v,w$. In this case u covers more than four edges.

Case 2: Suppose the degree of u is more than two. In this case u is adjacent to more than two vertices so that the inequality $d(u,v) \leq 2$ is satisfied for more than four vertices *i.e.,* u covers more than four edges.

Thus it is evident from the above cases that u covers at least four edges. Hence the proof.

Lemma 3. *A cycle of* $4m$ *vertices contribute at least* m *vertices in the 2-NC set* X.

Proof: The degree of all vertices of a cycle are two. Let u, v, w, x, y be the five consecutive vertices of a cycle. Hence from Lemma 2 it is evident that a vertex u can cover four edges $(u,v),(u,w),(w,x),(v,y)$ where v,w are adjacent to u and x,y are adjacent to w and v respectively as $d(u,r) \le 2$, $r = v,w,x,y$. Now a cycle of $4m$ vertices contain $4m$ edges. Thus to cover $4m$ edges at least m vertices are needed.

Lemma 4. *A cycle containing* $4m+1$, $4m+2$ *and* $4m+3$ *vertices contribute at least* $m+1$ *vertices in the 2-NC set* X.

Lemma 5. *A path containing* $4m$ *edges i.e., containing* $4m+1$ *vertices contribute at least* m *vertices in the 2-NC set* X.

Proof: In a path every vertex except the end vertices are of degree two. So, if u is a vertex on the path which is not an end vertex or its adjacent then it must cover four edges because end vertices of a path cover at most two edges where as the adjacent vertex of an end vertex can cover at most three edges, one from the side where end vertex lies and two edges from the other side. Hence to cover all the edges on this path at least m vertices are needed.

Lemma 6. *A path containing* $4m+1, 4m+2$ *or* $4m+3$ *edges contribute at least* $m+1$ *vertices in 2-NC set* X.

Proof: As a path containing $4m$ edges contribute at least m vertices in X then for the rest 1 or 2 or 3 edges one vertex is required to cover them. Thus $m+1$ vertices are necessary to cover all the edges of these paths.

Lemma 7. *Between two vertices* $u_i, u_j \in X$ *and also* $u_i, u_j \in C_i$ *(or path), there exist at most* $4r+4$ *edges, i.e.,* $4r+3$ *vertices,* r *being the number of vertices between* u_i *and* u_j *included in both* X *and the cycle* C_i *(respectively path).*

Proof: Suppose u_i and u_j are any two vertices of X belong to the same cycle (path). Let there exists r vertices in X between u_i and u_j. Now u_i covers two edges and u_j covers two edges between the edges of u_i and u_j. Also r vertices of X cover $4r$ edges. Hence there are $4r+4$ edges between u_i and u_j *i.e.,* $4r+3$ vertices.

Thus if $r = 0$, then there exists 4 edges between u_i and u_j *i.e.,* 3

vertices, if $r = 1$, then there exists 8 edges between u_i and u_j *i.e.,* 7 vertices, if $r = 2$, then there exists 12 edges between u_i and u_j *i.e.,* 11 vertices, and so on. Hence the proof.

6. Determination of 2-NC set from the tree blocks

For a tree block T_i consider the vertex common to T_i and the adjacent cycle block of T_i as the root say, u_0^*. Here the adjacent cycle block of T_i is the node which is consider after T_i in the Euler sequence. Then the adjacent vertices of the root are placed at level 1 and the adjacent vertices of the vertices of level 1 are placed at level 2 and so on. Thus the height of the tree h is defined as $h = max\{d(u_0^*, v)$, u_0^* being the root and $v \in T_i\}$. The vertices for which maximum level is obtained, one of that is denoted by u_h^* and the path between u_0^* and u_h^* on which h occurs is treated as the *main path* of the tree. The vertices on the main path are denoted as $u_0^*, u_1^*, u_2^*, \ldots, u_h^*$, where subscripts denote the level of vertices.

For every vertex u_i^*, $i = 1,2,\ldots,h$ there exists one or more subtrees rooted at u_i^*. These are denoted by $B_1(u_i^*), B_2(u_i^*), B_3(u_i^*)\ldots B_M(u_i^*)$, M being the total number of subtrees rooted at u_i^*.

Clearly, the height of $B_k(u_i^*)$ is less than or equal to $h-i$ for all $k = 1,2,\ldots,M$. Some of the subtrees are paths rooted at u_i^*. Thus for each such path $|B_k(u_i^*)| \le h-i$ for some k, as the maximum number of edges and vertices excluding u_i^* on that path is $h-i$.

Lemma 8. *The vertex u_{h-2}^* is the first member of X in T and $d(u_{h-2}^*, v) \le 2$, for all $v \in B_k(u_{h-i}^*)$, $i = 1,2$.*

Proof: Since the height of the tree is h, there is no subtree (path) rooted at u_h^*. If there is a subtree (path) rooted at u_{h-1}^*, height of that subtree is one. Similarly, u_{h-2}^* may has subtrees of height ≤ 2. Otherwise, they all exceed the height of the tree. Also $d(u_{h-2}^*, u_h^*) = 2$. Hence for all $v \in B_k(u_{h-i}^*)$, $i = 1,2$, $d(v, u_h^*) \le 2$.

Lemma 9. *If the path rooted at the vertex u_i^*, $i = 0,1,2,\ldots,h-2$ with $|B_k(u_i^*)| = 4m+2$ for at least one k, then u_i^* is a member of X.*

Proof: From Lemma 2, we have seen that one vertex cover at least four edges. For any path rooted at u_i^*, if we start from its leaf, m vertices are included in X to cover $4m$ edges. For the rest two edges either adjacent to u_i^* on that path or u_i^* is the vertex to cover them. But u_i^* is the most suitable vertex to cover them because it also covers more edges on the main path as well as on the other subtrees rooted at u_i^*. Hence the proof.

Lemma 10. *The path rooted at* u_i^*, $i = 0,1,2,\ldots,h-2$ *with* $|B_k(u_i^*)| = 4m+3$, *for some* k, $adj(u_i^*) \in B_k(u_i^*)$ *is a member of* X.

Proof: A path with $|B_k(u_i^*)| = 4m+3$ contains $4m+3$ vertices and $4m+3$ edges. Here also m vertices covered $4m$ edges if we start from the leaf of the path. Three edges are left uncovered below u_i^*. Thus if, the vertex $adj(u_i^*) \in B_k(u_i^*)$ is selected, then it covers those edges. Also $adj(u_i^*)$ can cover all the edges incident on u_i^*.

Lemma 11. *The path rooted at* u_i^*, $i = 0,1,2,\ldots,h-2$ *with* $|B_k(u_i^*)| = 4m+1$, *for some* k, u_i^* *or* $adj(u_i^*)$ *is a member of* X, *where* $adj(u_i^*)$ *is not a member of* $B_k(u_i^*)$.

Lemma 12. *The path rooted at* u_i^* *with* $|B_k(u_i^*)| = 4m$ *or* $4m+1$ *or* $4m+2$, *for some* k *contribute* m *vertices and with* $|B_k(u_i^*)| = 4m+3$, *contribute* $m+1$ *vertices.*

Proof: The path containing *4m,* $4m+1$and $4m+2$ vertices contain $4m$, $4m+1$, $4m+2$ edges. By Lemma 5, m vertices on the path are selected to cover $4m$ edges. Also by Lemma 9 and Lemma 10, m vertices on the path are selected and u_i^* or $adj(u_i^*) \notin B_k(u_i^*)$ are selected to cover the $4m+1$ and $4m+2$ edges. Hence $B_k(u_i^*)$ with $4m, 4m+1, 4m+2$ vertices contribute m vertices in X.

From Lemma 10, for $|B_k(u_i^*)| = 4m+3$, m vertices are selected for $4m$ edges and for the other three edges $adj(u_i^*) \in B_k(u_i^*)$ is selected. Hence $B_k(u_i^*)$ contribute $m+1$ vertices in X.

Procedure to determine the 2-NC set from the tree

Using the above lemmas the procedure for selecting covering vertices from the tree T_{BC} is described below.

Step-1: Start from u_h^*. The first member of X is u_{h-2}^*.

Step-2: Go to the vertex u_{h-i}^*, $i = 3,4,\ldots,h$ one by one. Two cases arise here for each i.

Case 1: There exist no subtrees rooted at u^*_{h-i}.

In this case we proceed to the vertex u^*_{h-i-1}, $i \neq h$ and apply Case 1 and Case 2 of Step 2.

Case 2: There exist some subtrees rooted at u^*_{h-i}.

Here also two cases arise.

Subcase 2.1: If some subtrees are paths then consider the leaf vertex of the path at the first position and consider the following situations.

(i) If the paths are of length $4m+2$, then select u^*_{h-i} and the vertices at $(4k-1)$th position, $k=1,2,\ldots,m$ are selected in X.

(ii) If the paths are of length $4m+3$, then select the vertices at $(4k-1)$th position, $k=1,2,\ldots,m+1$ in X.

(iii) If the paths are of length $4m$, then select the verties at $(4k-1)$th position, $k=1,2,3,\ldots,m$ in X.

(iv) If the paths are of length $4m+1$, then select the vertices at $(4k-1)$th position, $k=1,2,\ldots,m$ in X.

In this case one edge incident on u^*_i is left uncovered. It is covered by the vertex u^*_{h-i} or $adj(u^*_{h-i})$ which is either belongs to the main path or another subtrees (paths) rooted at the vertex u^*_{h-i} *i.e.,* they are situated at level $h-i+1$ or $h-i-1$. Now if the vertex $adj(u^*_{h-i})$ of level $h-i+1$ or u^*_{h-i} is not already selected for X then the vertex u^*_{h-i-1} must be a member of X.

Subcase 2.2: If the subtrees are another trees then find their height and following the same procedure as described in Step 1 and Step 2 find the covering vertices from those subtrees.

Thus applying the Step 1 and Step 2 repeatedly for the vertices one by one on the main path from level h to the level 0, the 2-NC set of the tree will be obtained.

At the time of consideration of u^*_0 if some edges incident on the vertex u^*_0 *i.e.,* the root of the tree or tree block is uncovered then select u^*_0 or $adj(u^*_0)$ from the tree or from cycle containing the vertex u^*_0. If $adj(u^*_0)$ from the tree or tree block is not selected then it is selected from the cycle when we consider the cycle.

7. Determination of 2-NC set from the nodes of the tree T_{BC}

Here we consider the nodes of the tree T_{BC} one by one from the sequence obtained from Euler tour. The nodes C_j or T_k for which $f(j)$ or $f'(k)$ is 1, are leaf nodes. Otherwise, the nodes are interior nodes.

7.1. Finding covering vertices from a leaf node of T_{BC}

If the leaf node is a cycle C_i we apply the following procedure. Suppose v is the cutvertex of the leaf node C_i and consider the cutvertex v in the first position of the cycle. Now

(1) For the leaf node C_i having $4m$ vertices mark the cutvertex v first and thereafter mark the vertices at $(4k+1)$ th position, $k = 1,2,\dots,m$.

(2) For the leaf node C_i having $4m+1$ vertices, one of the incident edges on the cutvertex v is left uncovered and marked the vertices at $(4k-1)$ th position, $k = 1,2,\dots,m$ (considering the cutvertex v in the first position).

(3) For the leaf node C_i having $4m+2$ vertices, both the edges incident on the cutvertex v are left uncovered and marked the vertices at $4k$ th position, $k = 1,2,\dots,m$.

(4) For the leaf node C_i having $4m+3$ vertices mark the cutvertex v first, then mark vertices at $(4k+1)$ th position, $k = 1,2,\dots,m$.

If the leaf node be a tree block T_i, then find the covering vertices from C_i for X by applying the procedure described in Section 6.

After selecting the vertices for the covering set from leaf nodes mark all edges which are covered by those vertices.

7.2. Finding covering vertices from an interior node of T_{BC}

After marking the covered edges from leaves or children nodes (C_j or T_j) the respective Parent(C_j) or Parent(T_j) which is another cycle C_i or tree T_k have the following situations.

(1) None of the edges of the $Parent(C_j)$ or $Parent(T_j)$ is covered. Also there may some uncovered edges incident on the some cutvertices with its children nodes.

(2) One or more edges of the $Parent(C_j)$ or $Parent(T_j)$ are covered from its children node. Here also may arise some uncovered edges incident on the cutvertices with its children nodes.

Case 1: Here (i) if the node is a tree block (T_k) then the uncovered edges of its children nodes increase its height as well as length of some subtrees of that tree block. Therefore applying the method described in Section 6 we find the covering vertices from that improved tree block.

(ii) If the node be a cycle block C_i, first we have to mark either the cutvertices of C_i and $C_{j'}s$ or any one of the adjacent vertices of the said cutvertices of the children nodes of C_i with C_j. It depends on the number of vertices lies between these cutvertices.

Suppose u_i are the cutvertices which have branch of length one. l_i be the path between u_i and u_{i+1}. $|l_i|$ be the number of vertices in l_i. Now we select u_i

or $adj(u_i)$ by using the following procedure.

From Lemma 7 we see that between two members of X of a cycle or a path there are $4r+3$ vertices, $r=1,2,\ldots,$. Using this lemma the following cases may arise.

Subcase 1: Let there be $4m+1$ vertices between u_i and u_{i+1}. In this case select the vertices $adj(u_i)\in l_i$ and $adj(u_{i+1})\in l_i$ or $adj(u_i)\in l_{i-1}$ and $adj(u_{i+1})\in l_{i+1}$. For the first pair there are $4m-1$ vertices between $adj(u_i)$ and $adj(u_{i+1})$ and for the second pair there are $4m+3$ vertices between $adj(u_i)$ and $adj(u_{i+1})$ which satisfy Lemma 7.

Subcase 2: Let there be $4m+2$ vertices between u_i and u_{i+1}. In this case select the vertices u_i and $adj(u_{i+1})\in l_i$ or $adj(u_i)\in l_{i-1}$ and u_{i+1}. For the first and for the second pair there are $4m+3$ vertices between u_i and $adj(u_{i+1})$ and between $adj(u_i)$ and u_{i+1} which satisfy Lemma 7.

Subcase 3: Let there be $4m+3$ vertices between u_i and u_{i+1}. In this case select the vertices $adj(u_i)\in l_i$ and $adj(u_{i+1})\in l_{i+1}$ or $adj(u_i)\in l_{i-1}$ and $adj(u_{i+1})\in l_i$ or u_i and u_{i+1}. There are $4m+3$ vertices between the vertices of each pair which satisfy Lemma 7.

Subcase 4: Let there be $4m$ vertices between u_i and u_{i+1}. In this case select the vertices u_i and $adj(u_{i+1})\in l_i$ or $adj(u_i)\in l_i$ and u_{i+1}. There are $4m-1$ vertices between the vertices of each pair which satisfy Lemma 7.

Lemma 13. *For* $|l_i|=4m$ *or* $4m+2$, $m=1,2,\ldots$

(i) u_{i+1} is the first member of X from C_i if l_{i-1} is of length $4m+1$.

(ii) u_i is the first member of X from C_i if l_{i+1} is of length $4m+1$.

(iii) u_i or u_{i+1} is the first member of X from C_i if both l_{i+1} and l_{i-1} are of length $4m+1$.

Proof: (a) For $|l_i|=4m$, select either the vertices u_i and $adj(u_{i+1})\in l_i$ or $adj(u_i)\in l_i$ and u_{i+1}.

(b) For $|l_i|=4m+2$, select either the vertices u_i and $adj(u_{i+1})\in l_{i+1}$ or $adj(u_i)\in l_{i-1}$ and u_{i+1}.

(c) For $|l_{i-1}|=4m+1$, select either the vertices $adj(u_i)\in l_{i-1}$ and $adj(u_{i-1})\in l_{i-1}$ or $adj(u_i)\in l_i$ and $adj(u_{i-1})\in l_{i-2}$.

(d) For $|l_{i+1}|=4m+1$ select either the vertices $adj(u_{i+1})\in l_{i+1}$ and $adj(u_{i+2})\in l_{i+1}$ or $adj(u_{i+1})\in l_i$ and $adj(u_{i+1})\in l_{i+2}$.

Case 1: Thus from (a) and (c) it is evident that if l_i is of length $4m$ and l_{i-1} is of length $4m+1$ then select u_{i+1}, $adj(u_i) \in l_i$, $adj(u_{i-1}) \in l_i$, for X. (1)

If l_i is of length $4m+2$ and l_{i+1} is of length $4m+1$ (b) and (d) give the covering vertices as $u_{i+1}, adj(u_{i-1}) \in l_{i-1}, adj(u_i) \in l_{i-1}$ in X. (2)

Thus (1) and (2) shows that u_{i+1} must be member of X. So it is the first member of X from that cycle C_i.

Case 2: Now if l_i is of length $4m$ and l_{i+1} is of length $4m+1$ then (a) and (d) give u_i, $adj(u_{i+1}) \in l_i$ and $adj(u_{i+2}) \in l_{i+2}$, (3)

and if l_i is of length $4m+2$ and l_{i+1} is of length $4m+1$, (b) and (d) give

$u_i, adj(u_{i+1}) \in l_{i+1}$ and $adj(u_{i+2}) \in l_{i+1}$. (4)

Thus (3) and (4) shows that u_i must be a member of X. So it is the first member of X from the cycle C_i.

Case 3: If both l_{i+1} and l_{i-1} are of length $4m+1$ and l_i is of length $4m$ or $4m+2$ select vertices from any one of (1) or (2) or (3) or (4), which shows that any one of u_i and u_{i+1} is the first member of X from C_i.

Lemma 14. *For l_i and l_{i+1} (l_{i-1}) are either both of length $4m$ or $4m+2$ or one is $4m$ and other is $4m+2$, u_{i+1} (u_i) is the first member of X from that cycle.*

Proof: (a) If l_i is of length $4m$, select either the vertices u_i and $adj(u_{i+1}) \in l_i$ or u_{i+1} and $adj(u_i) \in l_i$.

(b) For l_{i+1} is of length $4m$ select either u_{i+1} and $adj(u_{i+2}) \in l_{i+1}$ or u_{i+2} and $adj(u_{i+1}) \in l_{i+1}$.

(c) If l_i is of length $4m+2$, select either the vertices u_i and $adj(u_{i+1}) \in l_{i+1}$ or u_{i+1} and $adj(u_i) \in l_{i-1}$.

(d) For l_{i+1} is of length $4m+2$ select either the vertices u_{i+1} and $adj(u_{i+2}) \in l_{i+2}$ or u_{i+2} and $adj(u_{i+1}) \in l_i$.

Now if l_i and l_{i+1} both are of length $4m$ or $4m+2$, then from (a), (b), (c) and (d) the vertex u_{i+1} is the common member to be selected. Hence u_{i+1} must be a member of X. So it is the first member of X from C_i.

If l_i and l_{i+1} one of which is $4m$ and other is the length $4m+2$, then from (a), (c) and (b), (d) u_{i+1} is the common member to be selected. Hence u_{i+1} must be a member of X. So it is the first member of X from C_i.

For l_{i-1} instead of l_{i+1}, we can prove similarly that u_i is the first member of X. Hence the proof.

Lemma 15. *For all* l_i, $i = 1,2,\ldots,r-1$ *of length* $4m$ *or* $4m+2$, *any one of* u_i, $i = 1,2,\ldots,r$ *is the first member of* X *from the cycle* C_i.

Proof: (a) If l_i is of length $4m$, select either u_i, $adj(u_{i+1}) \in l_i$ or u_{i+1}, $adj(u_i) \in l_i$.

(b) If l_i is of length $4m+2$, select either u_i, $adj(u_{i+1}) \in l_{i+1}$ or u_{i+1}, $adj(u_i) \in l_{i-1}$.

From (a) and (b) it is evident that for each l_i, $i = 1,2,3,\ldots r-1$ of length $4m$ or $4m+2$, u_i or u_{i+1} is a common member of X. Thus for all l_i of length $4m$ or $4m+2$, any one of u_i, $i = 1,2,\ldots,r$ is the first member of X from the cycle C_i.

Lemma 16. *For all* l_i, $i = 1,2,\ldots,r-1$ *of length* $4m+1$ *or* $4m+3$, *any one of* $adj(u_i)$, $i = 1,2,\ldots,r$ *is the first member of* X *from the cycle* C_i.

Proof: (a) If l_i is of length $4m+1$ select either the vertices $adj(u_i) \in l_i$ and $adj(u_{i+1}) \in l_i$ or $adj(u_i) \in l_{i-1}$ and $adj(u_{i+1}) \in l_{i+1}$.

(b) If l_i is of length $4m+3$ select either the vertices $adj(u_i) \in l_i$ and $adj(u_{i+1}) \in l_{i+1}$ or $adj(u_i) \in l_{i-1}$ and $adj(u_{i+1}) \in l_i$ or u_i and u_{i+1}.

From (a) and (b) it is evident that for each l_i, $i = 1,2,3,\ldots r-1$ of length $4m+1$ or $4m+3$, $adj(u_i)$ is a common member for both the cases. So it is taken as the first member of X. Thus for all l_i of length $4m+1$ or $4m+3$, any one of $adj(u_i)$, $i = 1,2,\ldots,r$ is the first member of X from the cycle C_i.

Case 2: In this case also if the node be a tree block then height of the tree or length of some paths be decreased. Also if there be some uncovered edges incident on some vertex of the tree then the length of some path or height of the tree be increased.

If the node be a cycle there occur two or more than two trees.

Hence the steps to find the covering vertices from the interior node are:

Step 1: For the cycle arises in Case 1.

Select the first member of X from C_j and mark the edges covered by this vertex. Then the unmarked edges of the cycle C_j thus form two trees rooted at the cutvertex of C_j and *Parent(C_j)*. For every tree we apply the procedure as described in Section 6 to select the covering vertices from that tree.

Step 2: For the cycle arises in the Case 2.

Here the unmarked vertices occurs as two or more than two trees. For every tree we apply the procedure as described in Section 6 to select the covering vertices.

Step 3: For the tree block arises in Case 1 or Case 2 we apply the procedure as described in Section 6 to select the covering vertices from that tree block.

8. Algorithm and its complexity

In this section, we present an algorithm 2NBCOV to compute the 2-neighbourhood covering set on cactus graphs. The time and space complexities are also computed here. The proof of correctness of the algorithm is also presented in this section.

```
Algorithm 2NBCOV
Input: The cactus graph $G$.
Output: The 2-neighbourhood covered set $X$.
Step 1: Compute the blocks and cutvertices of $G$ as described in Section 3.
        //Let $S'$ be the set of edge blocks and form the tree blocks $T_i$,
        $i = 1,2,\ldots,L$. Also denote the cycle blocks as $C_j$, $j = 1,2,\ldots,N'$.//
Step 2: Construct a tree $T_{BC}$ whose nodes are the tree blocks and cycle blocks as described in Section 4.
Step 3: Apply Euler tour on $T_{BC}$ and store the output in the array $S(1:2(N'+L)-1)$, $N'+L$ is the total number of nodes of $T_{BC}$.
Step 4: Compute $f(j)$ and $f'(k)$, $j = 1,2,\ldots,N'$ and $k = 1,2,\ldots,L$, $N'+L$ which stores total number of occurrences of the node $C_j$ and $T_k$ in the array $S$.
Step 5: Note the order in which $(C_j)_{f(j)}$, $(T_k)_{f'(k)}$ $j = 1,2,\ldots,N$ and $k = 1,2,\ldots,L$ occurs in the array $S$.
Step 6: For each node of the resulting sequence, if
        (i) $f(j) = 1$ or $f'(k) = 1$, then find the vertices of $G$ using rule described in Section 7.1 and put them in the set $X$.
        (ii) $f(j) \neq 1$ or $f'(k) \neq 1$, then find the vertices of $G$ using the rule described in Section 7.2 and put them in the set $X$.
end 2NBCOV
```

Lemma 17. *The set X obtained from the algorithm 2NBCOV is a 2-neighbourhood covering set.*

Proof: Here the problem is to find 2-NC set. The set X is constructed in such a way that for every vertex $u \in X$, we find $v \in S'$ so that $d(u,v) \leq 2$. Now it is seen that $S \bigcup X = V$. Therefore all the edges connected with the vertices of X and vertices of S' are covered by the vertices of X, *i.e., E* is covered by the vertices of the set X. Thus X is the 2-NC covering set of the graph G.

Lemma 18. *The set X obtained from 2NBCOV is minimum among all the 2-NC covering set of the cactus graph G.*

Proof: From the Lemmas 2, 3, 4, 5, 6, 10 and 12, it is evident that the selection of covering vertices from cycles, paths and tree blocks is made in such a way that these contribute least number of vertices in the covering set. Also during consideration of the nodes of the tree T_{BC} we minimize the number of covering vertices for the cases where leaf nodes contain $4m+1$ and $4m+2$ vertices. Sometimes, these contribute m vertices in X instead of $m+1$ vertices as in Lemma 4. Similarly, for the tree block the paths containing $4m+1$ and $4m+2$ edges also contribute m vertices instead of $m+1$ vertices as in Lemma 12. Hence the lemmas and procedure are so designed that they find minimum number of vertices to 2-NC set X. Thus the set X is the minimum cardinality 2-NC set for the cactus graph G.

Theorem 1. *The minimum 2-neighbourhood covering set X obtained from the algorithm 2NBCOV can be computed in $O(n)$ time.*

Proof: The blocks and cutvertices of any graph can be computed in $O(m+n)$ time [13]. For the cactus graph $m=O(n)$, hence step 1 of Algorithm *2NBCOV* takes $O(n)$ time. Also formation of tree blocks T_i using the edge blocks of G takes $O(n)$ time. Hence step 2 can be computed in $O(n)$ time. In step 3, the construction of the tree T_{BC} using tree blocks and cycles, finding $f(j)$ and $f'(k)$ for each node and finding sequence of nodes using Euler Tour take $O(n)$ time. Hence steps 3, 4, 5 and 6 take $O(n)$ time. Step 7 can be performed by comparing $f(j)$ and $f'(k)$ with 1 for $j = 1,2,\ldots, N'$ and $k = 1, 2, \ldots, L$. So this step takes only $O(n)$ time. Hence the algorithm 2NBCOV computed the 2-NC set in $O(n)$ time.

REFERENCES

1. Chang, G. J., Farber M. and Tuza, Z., Algorithmic aspects of neighbourhood numbers, *SIAM J. Discrete Math.*, 6 (1993) 24-29.
2. Chen, C. C. Y., Das, S. K., and Akl, S. G., A unified approach to parallel depth-first traversals of general trees, *Information Processing Letters*, 41 (1991) 49-55.
3. Golumbic, M. C., *Algorithmic Graph Theory and Prefect Graphs*, Academic Press, New York, 1980.
4. Ghosh, P. K. and Pal, M., An Optimal algorithm to solve 2-neighbourhood covering problem on trapezoid graph, *Advanced Modeling and Optimization*, 9 (1) (2007) 15-36.
5. Garey, M. R. and Johnson, D. S., *Computer and Intractability: A Guide to the theory of NP-Completeness*, Freeman, San Francisco, CA, 1978.
6. Gamst, A., Some lower bounds for a class of frequency assignment problems, *IEEE Transactions of Vehicular Technology*, 35(1) (1986) 8-14.
7. Hwang, S. F. and Chang, G. J., k-neighbourhood- covering and independence problems for chordal graphs, *SIAM J. Discrete Math.*, 11 (4) (1998) 633-643.
8. Koontz, W. L. G., Economic evaluation of loop feeder relief alternatives, *Bell System Technical J.*, 59 (1980) 277-281.
9. Kariv, O. and Hakimi, S. L., An algorithmic approach to network location problems, Part 1: The p-center, *SIAM J. Appl. Math*, 37 (1979) 513-537.
10. Lehel, J. and Tuza, Z., Neighbourhood perfect graphs, *Discrete Math.*, 61 (1986) 93-101.

11. Pal, M., and Bhattacharjee, G. P., An optimal parallel algorithm for all-pairs shortest paths on unweighted interval graphs, *Nordic Journal of Computing*, 4 (1997) 342-356.
12. Mondal, S., Pal, M. and Pal, T. K., An optimal algorithm to solve 2-neighbourhood covering problem on interval graphs, *Intern. J. Computer Math.*, 79 (2) (2002) 189-204.
13. Reingold, E. M., Nivergent, J and Deo, N., *Combinatorial Algorithms : Theory and Practice,* (Prentice Hall, Inc., Englewood Chiffs, New Jersy, 1977).
14. Sarrafgadeh, M. and Lee, D. T., A new approach to topological via minimization, *IEEE Trans. Computer Aided Design*, 8 (1989) 890-900.
15. Schaerf, A., A survey of automated timetabling, *Artificial Intelligence Review,* 13 (1999) 87-127.
16. Pal, M. and Bhattacharjee, G.P., Sequential and parallel algorithms for computing the center and the diameter of an interval graph, *Intern. J. Computer Mathematics*, 59(1+2) (1995) 1-13.
17. Pal, M. and Bhattacharjee, G.P., Parallel algorithms for determining edge-packing and efficient edge domination sets in an interval graph, *Parallel Algorithms and Applications*, 7 (1995) 193-207.
18. Pal, M. and Bhattacharjee, G.P., A sequential algorithm for finding a maximum weight k-independent set on interval graphs, *Intern. J. Computer Mathematics*, 60 (1996) 205-214.
19. Pal, M. and Bhattacharjee, G.P., A data structure on interval graphs and its applications, *Journal of Circuits, System and Computers*, 7(3) (1997) 165-175.
20. Saha, A. and Pal, M., An algorithm to find a minimum feedback vertex set of an interval graph, *Advanced Modeling and Optimization,* 7(1) (2005) 99--116.
21. Pal, M., Efficient algorithms to compute all articulation points of a permutation graph, *The Korean J. Comput. Appl. Math.*, 5(1) (1998) 141-152.
22. Bera, D., Pal, M. and Pal, T.K., An optimal parallel algorithm for computing cut vertices and blocks on permutation graphs, *Intern. J. Computer Mathematics*, 72(4) (1999) 449--462.
23. Hota, M., Pal, M. and Pal, T.K., An efficient algorithm for finding a maximum weight *k*-independent set on trapezoid graphs, *Computational Optimization and Applications*, 18 (2001) 49-62.
24. Bera, D., Pal, M. and Pal, T.K., An efficient algorithm for generate all maximal cliques on trapezoid graphs, *Intern. J. Computer Mathematics*, 79 (10) (2002) 1057--1065.
25. Bera, D., Pal, M. and Pal, T.K., An optimal PRAM algorithm for a spanning tree on trapezoid graphs, *J. Applied Mathematics and Computing,* 12(1-2) (2003) 21--29.
26. Barman, S.C., Mondal, S. and Pal, M., An efficient algorithm to find next-to-shortest path on trapezoidal graph, *Advances in Applied Mathematical Analysis,* 2(2) (2007) 97-107.
27. Das, K. and Pal, M. An optimal algorithm to find maximum and minimum height spanning trees on cactus graphs, *Advanced Modeling and Optimization*, 10 (1) (2008) 121-134.