linear-time algorithms for edge domination in block-cactus graphs

Yancai Zhao[1,2]
[1] Wuxi City College of Vocational Technology, Jiangsu 214153, China
[2] Wuxi Environmental Science and Engineering Research Center, Jiangsu 214153, China

Abstract

In a graph $G = (V, E)$, a subset $F \subseteq E$ is an *edge dominating set* if every edge $e \in E \setminus F$ is adjacent to at least one edge in F. The *edge domination problem* is to find a minimum edge dominating set of G. This paper studies edge domination problem from an algorithmic point of view. Our final goal is to present a linear-time algorithm solving the edge domination problem in block-cactus graphs, which is a superclass of both block graphs and cacti. Our algorithm is based on the using of edge-label method and the handling of a series of configurations in block-cactus graphs, so as by-products, we also design different algorithms and procedures for these configurations, including , tree, cycle, sun, cactus, part tree, end sun and end clique, etc.

Keywords: edge domination; algorithm; edge-label method; block-cactus graph.

MSC: 05C85; 05C69

1 Introduction

In this paper we in general follow [7] for notation and graph theory terminology. All graphs considered here are simple graphs, that is, finite, undirected, without loops or multiple edges, and without isolated vertices. Let $G = (V, E)$ be a simple graph, and let v be a vertex in V. The *neighborhood* $N(v)$ of v is defined as the set of vertices adjacent to v. The *closed neighborhood* of v is $N[v] = N(v) \cup \{v\}$. Similarly, let e be an edge in E. The *neighborhood* $N(e)$ of e is defined as the set of edges adjacent to e. The *closed neighborhood* of e is $N[e] = N(e) \cup \{e\}$. The *degree* of a vertex v is $d_G(v) = |N_G(v)|$ and the *degree* of an edge e is $d_G(e) = |N_G(e)|$. If no ambiguity occurs, $d_G(v), N_G[v], d_G(e), N_G[e]$ are replaced by $d(v), N[v], d(e), N[e]$, respectively. The set of edges incident to vertex v is denoted $E(v)$ in our paper.

Given a non-empty graph $G = (V, E)$, an *edge dominating set* of $G = (V, E)$ is an edge subset $F \subseteq E$ such that every edge not in F is adjacent to an edge in F. The *edge domination number* $\gamma'(G)$ of G is the minimum cardinality among all edge dominating sets of G. In [14] Mitchell and Hedetniemi introduced this concept and it has been extensively studied in, for example, [8, 12, 15, 16].

The edge domination problem is known to be NP-complete, even when restricted to planar cubic graphs, planar bipartite graphs and subdivision graphs of planar bipartite graphs [8, 16]. Linear-time algorithms are available for trees [14, 16] and block graphs [6]. A $O(n^2)$-time algorithm for cacti is mentioned in [8]. The method used by Mitchell and Hedetniemi in [14] and Yannakakis and Garvil in [16] are based on a relationship between edge dominating sets in a graph and independent sets in its total graph; the method used by Hwang and Chang in [6] is a labeling approach which labels the vertices of a graph; the method used by Horton and Kilakos in [8] is based on relationships between edge domination, stable set and matching.

This paper continues the study the algorithmic aspects of edge domination problem. Our purpose is to present a linear-time algorithm for the edge domination problem in block-cactus graphs whose blocks are edges, cycles or cliques, which includes both block graphs and cacti. To this purpose, we need to handle a series of configurations, designing different algorithms or procedures for different configurations. Note that, our algorithm labels the edges of a graph, different from the existing methods to label the vertices of a graph.

2 Preliminary definitions

The *subgraph of* G *induced* by $S \subseteq V$ is the graph $G[S]$ with vertex set S and edge set $\{uv \in E|\ u, v \in S\}$. A *clique* is defined to be a maximal complete induced subgraph with at least three vertices. In a graph $G = (V, E)$, the *deletion* of $S \subseteq V$ from G, denoted by $G - S$, is the induced subgraph $G[V \setminus S]$. The deletion of $F \subseteq E$ from G, denoted by $G - F$, is the graph $(V, E \setminus F)$. For an element ε in G, we write $G - \varepsilon$ for $G - \{\varepsilon\}$. The *union* of two graphs G_1 and G_2 is the graph $G_1 \cup G_2$ with vertex set $V(G_1) \cup V(G_2)$ and edge set $E(G_1) \cup E(G_2)$. The *length* of a path is the number of edges in the path. The distance $d(u, v)$ is the minimum length of a path from u to v.

A *matching* in a graph is a set of pairwise nonadjacent edges. If M is a matching, then each vertex incident with an edge of M is said to be covered by M. A *perfect matching* is one which covers all vertices of the graph, a *maximum matching* is one which covers the maximum vertices as among all matchings.

A *forest* is a graph without cycles. A *tree* is a connected forest. A *pendant vertex* in a graph is a vertex with degree one. The pendant vertex is also called a *leaf* in a tree. The unique neighbor of a pendant vertex is called its *support vertex*. A *pending edge* in a graph is an edge incident to a pendant vertex. In a graph G, a vertex x is a *cut-vertex* if deleting x increases the number of connected components of G. A *block* of G is a maximal connected subgraph without a cut-vertex. If G has no cut-vertex, G itself is a block. The intersection of two blocks contains at most one vertex and a vertex is a cut-vertex if and only if it is the intersection of two or more blocks. A block B of G is called an *end block* if B contains at most one cut-vertex of G. A *block graph* is a graph whose blocks are edges or cliques. A *cactus* is a connected graph whose blocks are either an edge or a cycle. Alternatively, a cactus is a connected graph in which two cycles have at most one vertex (cut-vertex) in common. A more general graph class than both block graphs and cacti is the *block-cactus graph* which is defined as a graph whose blocks are edges, cycles or cliques, which includes block graphs and cacti.

In general, the blocks of a connected graph fit together in a treelike structure. We will base on this property, deeply seek and take use of different configurations in bloc-cactus graphs to design our algorithms and procedures in the next sections. In our paper, we also define the following concepts. For a vertex v of G, we use $E(v)$ to denote the set of edges incident v. A *sun* is a cycle C together with all pending edges attached to some vertices of C, where, C is called the *base cycle* of the sun. A *pseudo clique* is a clique Q together with some pending edges attached to some vertices of Q, where, Q is called the *base clique* of the pseudo clique. When an end block of G is a cycle or a clique, respectively, we call it an *end cycle* or an *end clique*, respectively. An *end sun* in a graph G is a sun S respect to a cycle C such that there is exactly one common vertex between C and all blocks outside S. An *end pseudo clique* can be defined similarly.

3 Linear-time algorithms in trees, cycles and suns

To obtain a linear-time algorithm for edge domination in trees, we use the well-know labeling method, whose different variations are widely used in the literature for solving the domination-related problems [1, 2, 3, 4, 9, 10, 11, 17, 18]. Noticing that all the former labeling methods are labeling vertices of a graph. In this paper we use the labeling method to the edges of a graph.

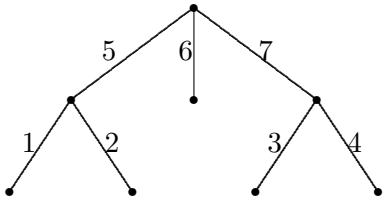
We partition the edge set E into $F \cup B \cup R$. More specifically, Given a graph $G = (V, E)$, an FBR assignment is a mapping that assigns each edge a label in $\{F, B, R\}$. For simplicity, the terms F, B, R represent sets and labels interchangeably. Since every edge has a label and the labels of some edges maybe change in the iterations of our algorithm, so we further give a generalization of the concept of edge domination. In an edge-labeled graph G, an *optional edge dominating set* of G is a subset $D \subseteq E$ such that $R \subseteq D$ and D dominates B. The *optional edge domination number* of G, denoted by $\gamma'_{opt}(T)$, is the cardinality of a minimum optional edge dominating set of G. A minimum optional edge dominating set is also called a $\gamma'_{opt}(T)$-set. $e \in F$ is called a *free edge*, which means that e needs not to be dominated by D (but can be used in D to dominate edges in B); $e \in B$ is called a *bound edge*, which means that e needs to be dominated by D; $e \in R$ is called a *required edge*, which means that e must be in D. Note that an algorithm to find a minimum optional edge dominating set suffices to find a minimum edge dominating set when specifying $B = V$ and $F = R = \emptyset$ at the beginning of the algorithm.

We first give a FBR labeling algorithm to find a minimum optional edge dominating set of a tree, which will be used as a subroutine in our main algorithms. Before our algorithm, we need an edge data structure as follows.

First root the tree T at any vertex, say, r. The *height* of T is the maximum distance between r and all other vertices. Let h be the height of T. The i-th *level* A_i $(0 \leq i \leq h)$ be the set of vertices of T which are at distance i from the root.

For such a rooted tree T with order n and edge number m, we can number the edges of T with $1, 2, \ldots, m-1$ as follows. We go on every level starting with level h to level 1. For each level i $(1 \leq i \leq h)$, we traverse the edges connecting the vertices on level i and level $i-1$ in arbitrary order, say from left to right. Finally, we list out the father edge of every edge of T.

Since the edges incident to the root r having no father edge, we write their father edge 0 for convenience. Now we can represent T by a data structure called an *edge father array*. Fig. 1 shows an example of a tree and its edge father array.



edge	1	2	3	4	5	6	7
edge father	5	5	7	7	0	0	0

Figure 1: A rooted tree with its edge father array

Algorithm 1. OptEdgDomTree (Finding a minimum optional edge dominating set of a tree).
Input: a tree T with its edge father array and an arbitrary FBR assignment L.
Output: a minimum optional edge dominating set of T, consisting of edges with label R.

Initiate
$T' \leftarrow T$;
$D \leftarrow \emptyset$;
do while the height of T' is at least two
  for $e = 1$ to $m - 2$ **do**
    let $e = uv$ be a pending edge of T' such that v is a leaf of T';
    if $e \in F$, **then** $T' \leftarrow T' - v$;
    if $e \in B$, **then**
      if $N[e] \cap R \neq \emptyset$ **then** $T' \leftarrow T' - v$;
      else $L(father(e)) \leftarrow R$; $T' \leftarrow T' - v$;
    if $e \in R$ **then**
      $L(x) \leftarrow F$ for every edge $x \in N(e) \cap B$;
      $D \leftarrow D \cup \{e\}$;
      $T' \leftarrow T' - v$;
  end for
end while
Now the height of T' is one;
if there is an edge with label R **then**
  $D \leftarrow D \cup \{e \in T' |\ L(e) = R\}$ and stop;
else
  if there is an edge $e \in B$ **then**
  $L(e) \leftarrow R$;
  $D \leftarrow D \cup \{e\}$ and stop;
  else stop.

Theorem 1. *Algorithm* OptEdgDomTree *produces a minimum optional edge dominating set*

of a tree T in linear time.

Proof. It is easy to see that the running time of algorithm OptEdgDomTree is $O(m)$, since it visits each edge $e = uv$ of T once, and all of the statements within which can be executed in a constant time. For the correctness of the algorithm, it is sufficient to consider T with height at least two, since the algorithm obviously produces a minimum optional edge dominating set of a tree with height one correctly. Then the proof of Theorem 1 relies on the following two claims.

Claim 1. *If $e \in F$ is a pending edge of T, then there exists a minimum edge dominating set of T not containing e.*

Let D be a minimum edge dominating set of T. Since $e \in F$, e needs not to be edge dominated by D. If $e \in D$, then $D \setminus \{e\} \cup \{father(e)\}$ is another minimum optional edge dominating set of T, since $father(e)$ can dominates more edges than e.

Claim 2. *If $e \in B$ is a pending edge of T, then there exists a minimum edge dominating set of T containing $father(e)$.*

Let D be a minimum edge dominating set of T. Since $e \in B$, there should be an edge $e' \in N[e] \cap D$ to dominate e. Then $D \setminus \{e'\} \cup \{father(e)\}$ is another minimum optional edge dominating set of T, since $father(e)$ can dominates more edges than e'.

Claim 3. *If $e = uv \in R$ and T' is the tree which results from T by deleting v and relabelling the edges in $N[e]$ as the statements in the corresponding step of the algorithm, then $\gamma'_{opt}(T) = \gamma'_{opt}(T') + 1$.*

Let D be a γ'_{opt}-set of T. Since $e \in R$, $e \in D$. Noticing all the vertices in $N[e]$ have been relabeled R or F in T' by step 3, there is no edge in $N[e]$ with label B needing to be edge dominated by e, so e is not needed to be in a optional edge dominating set of T', and thus $D \setminus \{e\}$ is a optional edge dominating set of T'. Therefore $\gamma'_{opt}(T') \leq |D \setminus \{e\}| = \gamma'_{opt}(T) - 1$.

Conversely, let D' be a γ'_{opt}-set of T'. Obviously $D' \cup \{e\}$ is a optional edge dominating set of T. This means that $\gamma'_{opt}(T) \leq |D' \cup \{e\}| = \gamma'_{opt}(T') + 1$. □

Based on algorithm OptEdgDomTree, we can easily design an algorithm to find a minimum optional edge dominating set of a cycle. The idea is to cut the cycle into a path and then use algorithm OptEdgDomTree.

Algorithm 2. OptEdgDomCyc (Finding a minimum optional edge dominating set of a cycle).
Input: a cycle C with an arbitrary FBR assignment L to every edge.
Output: a minimum optional edge dominating set of C.

Initiate
$D \leftarrow \emptyset$;
if $\exists$ an edge $e \in R$ **then**
  $L(e') \leftarrow F$ for each $e' \in N[e] \cap B$;
  $D \leftarrow \{e\} \cup$ OptEdgDomTree$(C - e)$;

else
  if $\exists$ an edge $e \in B$ **then**
    $|U_{min}| \leftarrow \infty$;
    for each $e_i \in N[e]$ **do**
      $L(e_i) \leftarrow R$;
      $L(x) \leftarrow F$ for every edge $x \in N(e_i) \cap B$;
      $U_i \leftarrow \{e_i\} \cup$ OptEdgDomTree$(C - e_i)$;
      if $|U_i| < |U_{min}|$ **then** $U_{min} \leftarrow U_i$;
    end for
    $D \leftarrow U_{min}$.

The construction and correctness of OptEdgDomCYC is straightforward, and the proof is omitted. For the time complexity, since OptEdgDomTree is linear and OptEdgDomCyc makes at most three calls to OptEdgDomTree (the number of closed neighbor of an edge in a cycle), it is clear that OptEdgDomCyc is linear.

Based on algorithm OptEdgDomCyc, we can further design an algorithm to find a minimum optional edge dominating set of a sun.

For convenience, we will always let the edges of the input graph unlabeled (or in other words, all edges are labeled B) in the following algorithms and procedures. For the former algorithms OptEdgDomTree and OptEdgDomCyc, we will write them with EdgDomTree and EdgDomCyc respectively when the input graph is unlabeled.

Algorithm 3. EdgDomSun.
Input: a sun S with respect to a cycle C.
Output: a minimum edge dominating set of S.
let uv be an arbitrary pending edge such that u is on C;
let $N_C(u) = \{s, t\}$;
$D_1 = \{us\} \cup$EdgDomTree$(S - \{u, s\})$;
$D_2 = \{ut\} \cup$EdgDomTree$(S - \{u, t\})$;
if $|D_1| \leq |D_2|$ **then**
  $D \leftarrow D_1$;
else
  $D \leftarrow D_2$.

The construction and correctness of EdgDomSun is straightforward. For the time complexity, since EdgDomTree is linear and EdgDomSun makes at most two calls to EdgDomTree, it is clear that EdgDomSun is linear.

4 An algorithm for edge domination on cacti

To obtain an algorithm to find a minimum edge dominating set of a cactus, we also need the following three procedures, to handle the configuration of a maximal induced subtree with height at least two and the configuration of an end cycle and the configuration of an sun with exact one cut-vertex.

For a tree T_x rooted at vertex x and with height at least two, if the algorithm EdgDomTree only visits the edges in $T_x - x$, then we call this procedure EdgDomPartTree $(T_x, T_x - x)$. It is easy to see that EdgDomPartTree $(T_x, T_x - x)$ is a procedure which uses minimum number of edges in T_x to dominate all edges in $T_x - x$.

Procedure EdgDomPartTree $(G, T_x, T_x - x)$.
let T_x be a tree rooted at x with height at least two in graph G;
construct the edge father array of T_x $[1, 2, \ldots, |E(T_x)|]$;
$T'_x \leftarrow T_x$;
for $e = 1$ to the last number such that e is in $T_x - x$ **do**
  let $e = uv$ be a pending edge of T'_x such that v is a leaf of T'_x;
  if $e \in F$, **then** $T'_x \leftarrow T'_x - v$;
  if $e \in B$, **then**
    if $N[e] \cap R \neq \emptyset$ **then** $T'_x \leftarrow T'_x - v$;
    else $L(father(e)) \leftarrow R$; $T'_x \leftarrow T'_x - v$;
  if $e \in R$ **then**
    $L(x) \leftarrow F$ for every edge $x \in N(e) \cap B$;
    $D \leftarrow D \cup \{e\}$;
    $T'_x \leftarrow T'_x - v$;
end for
if there is an edge in T'_x with label R, **then**
  $D \leftarrow D \cup (E(T'_x) \cap R)$;
  $G \leftarrow G - V(T_x)$ and stop;
elif there is an edge in T'_x with label B, **then**
  $G \leftarrow G - V(T_x - x) + E(T'_x) \cap B$ and stop;
else
  $G \leftarrow G - V(T_x - x)$ and stop.

Theorem 2. *The output D_T of procedure EdgDomPartTree $(G, T_x, T_x - x)$ is optimal for finding a minimum optional edge dominating set D_G of G.*

Proof. The optimality of the for statement in the procedure is ensured by the arguments in the proof of the algorithm OptEdgDomTree. Now we prove the optimality of the rest part of the procedure, when the height of T'_x is only one. If there is an edge in T'_x with label R, then this edge must be in D_G by the definition of a optional edge dominating set, and thus all the edges $E(x)$ in G will be edge dominated by this edge. So we can delete these edges from G. If no edge in T'_x with label R and there is an edge with label B, then this edge needs to be edge dominated by D_G, which is just the reason for us to leave this edge still in G, as shown in the elif statement of the procedure. Now there left the only case that all edges in T'_x with label F, then as the argument before in algorithm OptEdgDomTree, these edges can be deleted from G, as shown in the last statement of the procedure. □

For an end cycle in a cactus, we have the following procedure.

Procedure EdgDomEndCyc(G, C).
let C be an end cycle with the unique cut-vertex x of G;
let y be a neighbor of x in C;

$U_{xy} \leftarrow \{xy\} \cup \text{EdgDomTree}(C - \{x, y\})$;
$U_{\emptyset} \leftarrow \text{EdgDomTree}(C - x)$;
if $|U_{xy}| < |U_{\emptyset}| + 1$ **then**
  $D \leftarrow D \cup U_{xy}$;
  $G \leftarrow G - V(C)$;
else
  $D \leftarrow D \cup U_{\emptyset}$;
  $G \leftarrow G - V(C - x) + xy$;

The running time of procedure EdgDomEndCyc(G, C) is clearly linear. The construction and the correctness of procedure EdgDomEndCyc(G, C) is based on the following theorem.

Theorem 3. *For the graph G in procedure EdgDomEndCyc(G, C), there exists a minimum edge dominating set D of G such that D contains the output of EdgDomEndCyc(G, C).*

Proof. To dominate the edges in C, we should let either (i) edges only in C or (ii) edges in C together with some edges not in C, to be in D. For the case (i), we may assume $xy \in D$ by the symmetry of the edges in C, and thus $|U_{xy}|$ is the minimum number of edges to be in D. For the case (ii), only one edge not in C is needed to be in D to dominate xy and xz, and thus $|U_{\emptyset}| + 1$ is the minimum number of edges to be in D. Further, when $|U_{\emptyset}| + 1 = |U_{xy}|$, the set of the edges in $U_{\emptyset}$ together with some edge not in C is clearly the optimal choice to minimize $|D|$, since the edges dominated by this set contain the edges dominated by U_{xy}; also, a minimum edge dominating set of $G - V(C - x) + xy$ ensures that some edge not in C must be in D to dominate xy. □

We also need the following procedure to deal with an end sun.

Procedure EdgDomEndSun(G, S, C).
let S be an end sun in graph G with the base cycle C;
let x be the unique common vertex of C and other block(s) outside S, and y, z be neighbors of x in C;
$U_{xy} \leftarrow \{xy\} \cup \text{EdgDomTree}(S - \{x, y\})$;
$U_{xz} \leftarrow \{xz\} \cup \text{EdgDomTree}(S - \{x, z\})$;
$U_{\emptyset} \leftarrow \text{EdgDomTree}(S - x)$;
let $m = \min\{|U_{xy}|, |U_{xz}|, |U_{\emptyset}| + 1\}$;
if $|U_{\emptyset}| = m - 1$ **then**
  $D \leftarrow U_{\emptyset}$;
  $G \leftarrow G - (V(S) \setminus \{x\}) + E_s(x)$ and stop;
elif $|U_{xy}| = m$ **then**
  $D \leftarrow U_{xy}$;
  $G \leftarrow G - V(S)$ and stop;
else
  $D \leftarrow U_{xz}$;
  $G \leftarrow G - V(S)$ and stop.

The running time of procedure EdgDomEndSun is clearly linear. The construction and the correctness of procedure EdgDomEndSun is based on the following theorem.

Theorem 4. *For the graph G in procedure EdgDomEndSun, there exists a minimum edge dominating set D of G such that D contains the output of* EdgDomEndSun.

Proof. To edge dominate the edges in S, one of the following possible cases holds:

(1) only edges in S are needed. In the case, $\min\{|D_{xy}|, |D_{xz}|\}$ edges should be in D;
(1.1) $xy \in D$. In the case, $|D_{xy}|$ edges should be in D;
(1.2) $xz \in D$. In the case, $|D_{xz}|$ edges should be in D;
(2) one edge not in S is needed. In the case, $|U_\emptyset| + 1$ edges (including one edge not in S) should be in D.

Then the correction is based on the fact that, if $|U_\emptyset| + 1 = |D_{xy}| = |D_{xz}| = m$ then the set of $U_\emptyset$ edges together with an edge not in S is optimal for finding a minimum edge dominating set of the whole graph G, since the set of edges in G dominated by $U_\emptyset$ together with an edge not in S contains the set of edges in G dominated by D_{xy} or D_{xz}. Also, when we choose $U_\emptyset$ edges, which corresponds to the first if statement in EdgDomEndSun, $G-(V(S)\setminus\{x\})+E_s(x)$ means that some edge not in S must be in D to edge dominate the edges in $E_s(x)$.

Since D_{xy} and D_{xz} dominates the same edges in G, so it is easy to see the correctness of the other two statements in EdgDomEndSun. □

We are now ready to present our algorithm, called EdgDomCAC, to determine a minimum edge dominating set in a cactus. Our algorithm takes algorithms EdgDomTree and EdgDomCyc and procedures EdgDomPartTree,EdgDomEndCyc and EdgDomEndSun as subroutines.

Algorithm 4. EdgDomCAC (Finding a minimum edge dominating set of a cactus).
Input: a cactus K.
Output: a minimum edge dominating set of K.

$D \leftarrow \emptyset$;
$K' \leftarrow K$;
if K' is a tree **then** $D \leftarrow D \cup$EdgDomTree(K') and stop;
if K' is a cycle **then** $D \leftarrow D \cup$EdgDomCyc(K') and stop;
if K' is a sun **then** $D \leftarrow D \cup$EdgDomSun(K') and stop;
else
  while there exists a maximal induced subtree T_x with height at least two
    such that every vertex in $V(T_x)\setminus\{x\}$ is not on any cycle **do**
    $D \leftarrow D \cup$EdgDomPartTree $(K', T_x, T_x - x)$;
  end while
  while there is an end sun S respect to cycle C such that $x \in V(C)$
    is the unique common vertex of C and other block(s) outside S **do**
    $D \leftarrow D \cup$EdgDomEndSun(K', S, C);
  end while
  while there is an end cycle C such that $x \in V(C)$ is a cut-vertex of K' **do**
    $D \leftarrow D \cup$EdgDomEndCyc (K', C);
  end while.

Theorem 5. *Algorithm EdgDomCAC produces a minimum edge dominating set of a cactus in*

linear-time.

Proof. In each step of algorithm EdgDomCAC, it calls to one of EdgDomTree, EdgDomCyc, EdgDomSun, EdgDomPartTree, EdgDomEndCyc and EdgDomEndSun. Since each of these subroutines is linear, algorithm EdgDomCAC is clearly linear.

As for the correctness of algorithm EdgDomCAC, the three if statements in the algorithm correspond three algorithm, whose correctness has been proved previously. So we only need to show the correctness of the three while statements in the algorithm. First by the property of a cactus graph, it is easy to see that the three configurations dealt with by the three while statements are the all possible configurations during the whole algorithm runs. Also the optimality of the procedures dealing with the three configurations has been proved before.

About the label aspect of the algorithm, we notice that in the input of EdgDomCAC, every edge has no label (or other words, all labeled B) initially. When we execute every procedure of EdgDomPartTree, EdgDomEndSun and EdgDomEndCyc in the three while statements, though some labels in the corresponding configuration are iterated, but finally all the edges with labels F or R will be deleted, and thus the whole graph G where the configuration locates will be renewed to a new graph, say G', with no edge labeled. This ensures that the algorithm still faces a graph with no label when it deals with the next configuration. □

5 An algorithm for block-cactus graphs

We need the following algorithm and procedure to deal with configurations related with cliques.

Algorithm 5. EdgDomPsdClq (Finding a minimum edge dominating set of a pseudo clique).
Input: a pseudo clique $\widetilde{Q}$ with respect to a clique Q.
Output: a minimum edge dominating set of $\widetilde{Q}$.
let $V(Q) = \{u_1, u_2, \ldots, u_q\}$;
choose an arbitrary maximum matching M of Q;
$D \leftarrow M$.

Procedure EdgDomEndPsdClq($G, \widetilde{Q}, Q$).
let $\widetilde{Q}$ be an induced pseudo clique with respect to clique Q in graph G such that $|V(Q)| \geq 3$
let x be the unique common vertex of Q and other blocks outside $\widetilde{Q}$;
let $V(Q) = \{u_1, u_2, \ldots, u_q = x\}$;
if q is odd, **then**
    choose a perfect matching M of $Q - x$;
    $D \leftarrow M$;
    $G \leftarrow G - (V(\widetilde{Q}) \setminus \{x\})$;
else
    choose a maximum matching M' of Q;
    $D \leftarrow M'$;
    $G \leftarrow G - (V(\widetilde{Q}))$.

Theorem 6. *Algorithm EdgDomPsdClq is correct, and procedure EdgDomEndPsdClq is opti-*

mal. Both runs in linear time.

Proof. We first note a fact in the matching theory and edge domination theory, a collection of some edges is a minimum edge dominating set of a clique if and only if it is a maximum matching of the clique, which means that in order to edge dominate all the edges of a clique Q, a number of $\lceil \frac{q}{2} \rceil$ edges are needed. Secondly, it is easy to see that, in the case q odd, we use only $\lceil \frac{q}{2} \rceil - 1$ edges which is the number of a perfect matching of $Q - x$, and leave $E_{\widetilde{Q}}(x)$ being not edge dominated, which permits us to use an edge in $E_G(x) \setminus E_{\widetilde{Q}}(x)$ to dominate all the edges in $E_G(x)$. Such a method is obviously optimal for finding a minimum edge dominating set of the whole graph G.

For the time complexity, it is clear that the running time of above algorithm and procedure are all linear. □

Now, based on all the algorithms and procedures before, we are ready to provide a linear-time algorithm to find a minimum edge dominating set of a block-cactus graph.

Algorithm 6. EdgDomBLK-CAC (Finding a minimum edge dominating set of a block-cactus graph).
Input: a block-cactus graph G.
Output: a minimum edge dominating set of G.

$D \leftarrow \emptyset$;
$G' \leftarrow G$;
if G' is a tree **then** $D \leftarrow D \cup$EdgDomTree(G') and stop;
if G' is a cycle **then** $D \leftarrow D \cup$EdgDomCyc(G') and stop;
if G' is a sun **then** $D \leftarrow D \cup$EdgDomSun(G') and stop;
if G' is a clique or pseudo clique **then** $D \leftarrow D \cup$EdgDomPsdCLK(G') and stop;
else
  while there exists a maximal induced subtree T_x with height at least two
    such that every vertex in $V(T_x) \setminus \{x\}$ is not on any cycle or clique **do**
    $D \leftarrow D \cup$EdgDomPartTree $(G', T_x, T_x - x)$;
  end while
  while there is an end cycle C **do**
    $D \leftarrow D \cup$EdgDomEndCyc(K', C);
  end while
  while there is an end sun S respect to cycle C such that $x \in V(C)$
    is the unique common vertex of C and other block(s) outsider S **do**
    $D \leftarrow D \cup$EdgDomEndSun(K', S, C);
  end while
  while there is an end clique or end pseudo clique $\widetilde{Q}$ such that there is
    a unique common vertex of Q and other block(s) outsider $\widetilde{Q}$ **do**
    $D \leftarrow D \cup$EdgDomEndPsdClq $(G', \widetilde{Q}, Q)$;
  end while

Theorem 7. *Algorithm EdgDomBLK-CAC produces a minimum edge dominating set of a block-cactus graphs in linear time.*

Proof. In each step of algorithm EdgDomBLK-CAC, it calls to one of EdgDomTree, EdgDomCyc, EdgDomSun, EdgDomPsdClk, EdgDomPartTree, EdgDomEndCyc, EdgDomEndSun and EdgDomEndPsdCLQ. Since each of these subroutines is linear, algorithm EdgDomBLK-CAC is clearly linear.

As for the correctness of algorithm EdgDomBLK-CAC, the three if statements are clearly correct, so we only need to prove the correctness of the three while statements in the algorithm. First by the property of the block-cactus graphs, it is easy to see that the configurations dealt with by the four while statements are the all possible configurations during the whole algorithm runs. Also, the optimality of the procedures for dealing with all the configurations has been proved before.

About the label aspect of the algorithm, we notice that in the input of EdgDomBLK-CAC, every edge has no label (or other words, all labeled B) initially. When we execute every procedure of EdgDomPartTree, EdgDomEndCyc, EdgDomEndSun and EdgDomEndPsdClq in the three while statements, though some labels are iterated during this procedure runs, but finally all the edges with labels F or R will be deleted when this procedure stops, and thus the whole graph G where the configuration locates will be renewed to a new graph, say G', whose all edges remains no label. This ensures the algorithm always faces a graph with no label when it deals with the next configuration. □

6 Conflict of interest and data availability statement.

The authors have no relevant financial or non-financial interests to disclose. The authors have no competing interests to declare that are relevant to the content of this article. The authors have no conflict of interest in this article.

My data in this article is available and can be shared openly. My data supports my results and analyses in this article.

References

[1] S. Arumugam and S. Velammal, Edge domination in graphs, Taiwanese J. Math. 2 (2) (1998) 173–179.

[2] M.S. Chang, Effcient algorithms for the domination problems on interval and circular-arc graphs, SIAM J. Comput. 27 (1998) 1671–1694.

[3] G.J. Chang, G.L. Nemhauser, The k-domination and k-stability problems in sun-free chordal graphs, SIAM J. Algebraic Discrete Methods 5 (1984) 332–345.

[4] G.J. Chang, J. Wu, X. Zhu, Rainbow domination on trees, Discrete Appl. Math. 158 (1) (2010) 8–12.

[5] A.K. Dewdney, Fast turing reductions between problems in NP; chapter 4; reductions between NP-complete problems, Technical Report 71, Dept. Computer Science, Univ. Western Ontario, 1981.

[6] S.F. Hwang, G.J. Chang, The edge domination problem. Discuss. Math. Graph Theory 15 (1) (1995) 51–57.

[7] T.W. Haynes, S.T. Hedetniemi, P.J. Slater, Fundamentals of Domination in Graphs, Marcel Dekker, New York, 1998.

[8] J.D. Horton, K. Kilakos, Minimum edge dominating sets, SIAM J. Discrete Math. 6 (3) (1993) 375–387.

[9] S. Hedetniemi, R. Laskar, J. Pfaff, A linear algorithmfor finding a minimum dominating set in a cactus, Discrete Appl. Math. 13 (1986) 287–292.

[10] J.K. Lan, G.J. Chang, On the mixed domination problem in graphs, Theoret. Comput. Sci. 476 (2013) 84–93.

[11] R. Laskar, J. Pfaff, S.M. Hedetniemi, S.T. Hedetniemi, On the algorithmic complexity of total domination, SIAM J. Algebraic Discrete Methods 5 (1984) 420–425.

[12] K. Kilakos, On the complexity of edge domination, Master's thesis, University of New Brunswick, New Brunswick, Canada, 1988.

[13] V.R. Kulli, D.K. Patwari, On the edge total domination number of a graph, Proceedings of the Symposium on Graph Theory and Combinatorics (Cochin, 1991), 75–81, Publication, 21, Centre Math. Sci., Trivandrum, 1991.

[14] S. Mitchell and S. T. Hedetniemi, Edge domination in trees, Congr. Numer. 19 (1977), 489-509.

[15] B. Xu, Two classes of edge domination in graphs, Discrete Appl. Math. 154 (2006) 1541–1546.

[16] M. Yannakakis, F. Gavril, Edge dominating sets in graphs, SIAM J. Appl. Math. 38 (1980) 364–372.

[17] Y.C. Zhao, L.Y. Kang, M.Y. Sohn, The algorithmic complexity of mixed domination in graphs, Theoret. Comput. Sci. 412 (2011) (22), 2387–2392.

[18] Y.C. Zhao, L.Y. Miao, Z.H. Liao, A Linear-Time Algorithm for 2-Step Domination in Block Graphs, Journal of Mathematical Research with Applications 35 (3)(2015): 285–290.