



## AN EFFICIENT ALGORITHM FOR MIXED DOMINATION ON GENERALIZED SERIES-PARALLEL GRAPHS

M. RAJAATI, M. R. HOOSHMANDASL\*, A. SHAKIBA, P. SHARIFANI AND M. J. DINNEEN

Communicated by S. Alikhani

ABSTRACT. A mixed dominating set  $S$  of a graph  $G = (V, E)$  is a subset of vertices and edges like  $S \subseteq V \cup E$  such that each element  $v \in (V \cup E) \setminus S$  is adjacent or incident to at least one element in  $S$ . The mixed domination number  $\gamma_m(G)$  of a graph  $G$  is the minimum cardinality among all mixed dominating sets in  $G$ . The problem of finding  $\gamma_m(G)$  is known to be NP-complete. In this paper, we present an explicit polynomial-time algorithm using the parse tree to construct a mixed dominating set of size  $\gamma_m(G)$  where  $G$  is a generalized series-parallel graph.

### 1. INTRODUCTION

A subset of vertices and edges  $S \subseteq V \cup E$  in a graph  $G = (V, E)$  is a mixed dominating set if for every  $v \in (V \cup E) \setminus S$ ,  $v$  is either adjacent or incident to at least one element in  $S$ . The mixed domination problem, also known as the total cover problem, is a variant of the classical dominating set problem and was introduced by Alavi et. al in 1977 [2]. Placing phase measurement units (PMUs) in an electric power system is one of its known applications [15]. The mixed domination number of a graph  $G$  is the minimum cardinality among all mixed dominating sets of  $G$  and is denoted by  $\gamma_m(G)$ .

MSC(2010): Primary:05C85

Keywords: Mixed Dominating Set; Generalized Series-Parallel; Parse Tree; Tree-width.

Received: 20 April 2018, Accepted: 01 September 2018.

\*Corresponding author

In [2], Alavi et. al. showed that for a connected graph  $G$  of order  $n$ , the value of  $\gamma_m(G)$  is bounded from above by  $\lceil n/2 \rceil$ . In [3], they have also illustrated some extremal cases and gave some properties for connected graphs, which have a total covering number equal to  $\lceil n/2 \rceil$ . In [11], Majumdar showed that the problem of finding  $\gamma_m(G)$  is NP-complete for general graphs. Also, it is shown that this problem remains NP-complete even if it is restricted to chordal graphs [7], planar bipartite graphs [12], and split graphs [10, 15]. Finding a mixed dominating set of minimum cardinality is tractable for some families of graphs such as trees [1, 15, 10], cactus graphs [10] and graphs with bounded tree-width [13].

Rajaati et. al. proposed a dynamic programming algorithm to solve the mixed domination problem on graphs with bounded tree-width using the tree decomposition of graphs in [13]. In this paper, we use the parse tree of graphs to present an explicit polynomial-time algorithm to construct a mixed dominating set for generalized series-parallel graphs in linear time. Moreover, we enumerate the number of  $\gamma_m$ -sets of  $G$ . The rest of the paper is organized as follows: In Section 2, we review some basic definitions and set our notions. In Section 3, we present a linear time algorithm to find a  $\gamma_m$ -set and determine the number of  $\gamma_m$ -sets for  $G$  using the parse tree of a generalized series-parallel graph  $G$ . Then, we analyze the correctness and computational complexity of the proposed algorithms.

## 2. PRELIMINARIES

In this section, we review some basic requirements on graph theory and set our notations. For notation and terminology which are not listed here, an interested reader is advised to consult [14].

Let  $G = (V, E)$  be a graph with vertex set  $V$  and edge set  $E$ . The (open) neighborhood of a vertex  $v \in V$  in  $G$  is the set of all vertices adjacent to  $v$  and is denoted by  $N_G(v)$ . The closed neighborhood of a vertex  $v$  in  $G$  is defined as  $N_G[v] = N_G(v) \cup \{v\}$ . The mixed neighborhood of  $r$  in  $G$ , for an element  $r \in V \cup E$ , is denoted by  $N_G^{md}(r)$  and is defined as  $N_G^{md}(r) = \{s \in V \cup E \mid s \text{ is adjacent or incident to } r\}$ . Similarly, the closed mixed neighborhood of  $r$  is denoted by  $N_G^{md}[r]$  and is equal to  $N_G^{md}[r] = N_G^{md}(r) \cup \{r\}$ .

The problem of domination and its variations are well-studied topics in the literature of graph theory [5, 6]. The mixed domination problem is one of these variants. A subset  $S \subseteq V \cup E$  is a mixed dominating set if for every  $r \in V \cup E$ , it is the case that  $|N_G^{md}[r] \cap S| \geq 1$ . The minimum cardinality among such sets is denoted by  $\gamma_m(G)$ . Moreover, a  $\gamma_m$ -set for  $G$  is a mixed dominating set of size  $\gamma_m(G)$ .

**Definition 2.1** (Generalized Series-Parallel Graphs [4]). A generalized series-parallel, or *GSP* for short, is a graph  $G = (V, E, s, t)$  with two distinguished vertices  $s, t \in V$ , which are called terminals, and is defined recursively as follows:

- (1)  $o_i$ : A graph  $G$  consisting of two vertices connected by a single edge is a *GSP*.

- (2)  $o_s$ : Given two *GSP* graphs  $G_1 = (V_1, E_1, s_1, t_1)$  and  $G_2 = (V_2, E_2, s_2, t_2)$ , the series operation of  $G_1$  and  $G_2$  is denoted by  $G_1 o_s G_2$  and is a new *GSP* graph  $G = (V, E, s_1, t_2)$  where

$$V = (V_1 \cup V_2) \setminus \{s_2\},$$

and

$$E = (E_1 \cup E_2 \cup \{\{t_1, v\} : v \in N_{G_2}(s_2)\}) \setminus \{\{s_2, v\} : v \in N_{G_2}(s_2)\}.$$

- (3)  $o_p$ : Given two *GSP* graphs  $G_1 = (V_1, E_1, s_1, t_1)$  and  $G_2 = (V_2, E_2, s_2, t_2)$ , the parallel operation of  $G_1$  and  $G_2$  is denoted by  $G_1 o_p G_2$  and is a new *GSP* graph  $G_1 o_p G_2 = (V, E, s_1, t_1)$  where

$$V = (V_1 \cup V_2) \setminus \{s_2, t_2\},$$

and

$$E = (E_1 \cup E_2 \cup \{\{s_1, v\} : v \in N_{G_2}(s_2)\} \cup \{\{t_1, v\} : v \in N_{G_2}(t_2)\}) \setminus (\{\{s_2, v\} : v \in N_{G_2}(s_2)\} \cup \{\{t_2, v\} : v \in N_{G_2}(t_2)\})$$

- (4)  $o_g$ : Given two *GSP* graphs  $G_1 = (V_1, E_1, s_1, t_1)$  and  $G_2 = (V_2, E_2, s_2, t_2)$ , the generalized series operation of  $G_1$  and  $G_2$  is denoted by  $G_1 o_g G_2$  and is a new *GSP* graph  $G_1 o_g G_2 = (V, E, s_1, t_1)$  where

$$V = (V_1 \cup V_2) \setminus \{s_2\},$$

and

$$E = (E_1 \cup E_2 \cup \{\{t_1, v\} : v \in N_{G_2}(s_2)\}) \setminus (\{\{s_2, v\} : v \in N_{G_2}(s_2)\}).$$

- (5) Any *GSP* graph is obtained by finite applications of rules (1) through (4).

If the rule (4) is removed, then we obtain a subclass of *GPS*s called series-parallel or *SP* graphs. These rules are illustrated in Figure 1. Note that in Figure 1, the graph  $(\widehat{G} o_g (G_1 o_s G_2)) o_s ((G_1 o_s G_2) o_p \widehat{G})$  is a *GSP*, however it is not an *SP*. We generalize the concept

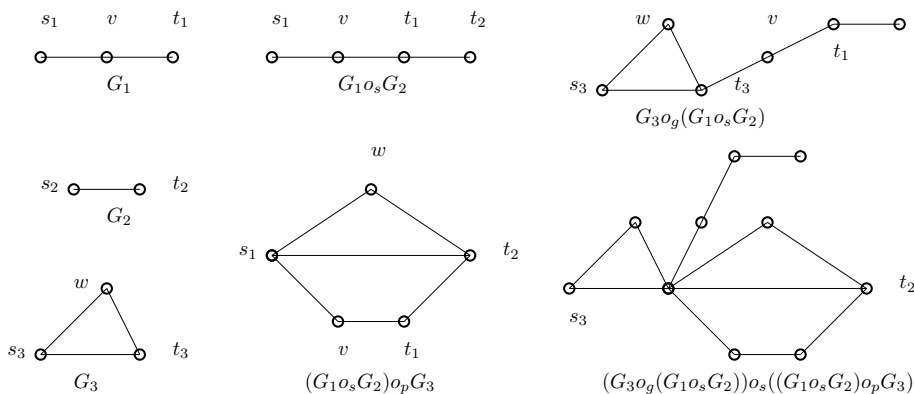


FIGURE 1. An illustration of applying *GSP* rules.

of  $p$ -graph, which are defined for  $SP$ s in [9], for  $GSP$  graphs in the following definition.

**Definition 2.2** ( $p$ -graph). Let  $G = (V, E, x, y)$  be a  $GSP$  and  $\widehat{G} = (\widehat{V}, \widehat{E}, \widehat{x}, \widehat{y})$  be a subgraph of  $G$  satisfying the following conditions:

- (1) either  $\widehat{x} = x$  or  $x \notin \widehat{V}$  and there exists an edge  $\{u, v\} \in E \setminus \widehat{E}$  such that  $v = \widehat{x} \in \widehat{V}$ .
- (2) either  $\widehat{y} = y$  or  $w \notin \widehat{V}$  and there exists an edge  $\{w, z\} \in E \setminus \widehat{E}$  such that  $z = \widehat{y} \in \widehat{V}$ .

Then,  $\widehat{G}$  is called a  $p$ -graph of  $G$ .

A generalized series-parallel graph  $G$  can be represented by a binary parse tree  $T$  which is defined as follows.

**Definition 2.3** (Binary Parse Tree for  $GSP$  Graphs[9]). A binary parse tree for  $GSP$  graph  $G$  is defined recursively as follows:

- (1) A tree consisted of a single vertex labeled  $(u, v)_i$  is a binary parse tree for primitive  $GSP$ ,  $G = (\{u, v\}, \{u, v\}, u, v)$ .
- (2) Let  $G = (V, E)$  be a  $GSP$  obtained by some composition of two other  $GSP$  graphs  $G_1$  and  $G_2$ , and  $T_1$  and  $T_2$  be their binary parse trees, respectively. Then, a binary parse tree for  $G$  is a tree with the root  $r$  labeled as either  $(u, v)_s$ ,  $(u, v)_p$  or  $(u, v)_g$  depending on which operation is used to generate  $G$ . The vertices  $u$  and  $v$  are terminals of  $G$  and the roots of  $T_1$  and  $T_2$  are the left and the right children of  $r$ , respectively.

It is obvious that in any binary parse tree for a  $GSP$  graph  $G$ , every internal vertex of the tree has exactly two children and there are  $|E|$  leaves.

**Remark 2.4.** Note that when we use a label  $(x, y)$ , we do not care about the label being either  $(x, y)_i$ ,  $(x, y)_s$ ,  $(x, y)_p$  and  $(x, y)_g$ .

Let  $t$  be an internal vertex of a binary parse tree  $T$ , and for  $GSP$  graph  $G$ ,  $\tau(t)$  denote the subtree of  $T$  rooted at  $t$ . Also, the left and the right subtrees of  $t$  are denoted by  $\tau_\ell(t)$  and  $\tau_r(t)$ , respectively. Then, the vertices of  $T$  are labeled as follows:

- (a) For each edge  $e = \{x, y\} \in E$ , there exists exactly one leaf which is labeled by  $(x, y)$  in  $T$ .
- (b) For each internal vertex  $t \in V_T$ , which is labeled by  $(x, y)_s$ , the root of  $\tau_\ell(t)$  is labeled by  $(x, z)$  and the root of  $\tau_r(t)$  is labeled by  $(z, y)$  where  $z$  is some vertex in  $V$ . These vertices are called  $s$ -vertices.
- (c) For each internal vertex  $t \in V_T$  labeled  $(x, y)_p$ , the root of  $\tau_\ell(t)$  and  $\tau_r(t)$  are labeled by  $(x, y)$ . These vertices are called  $p$ -vertices.
- (d) For each internal vertex  $t \in V_T$  labeled  $(x, z)_g$ , the root of  $\tau_\ell(t)$  is labeled by  $(x, z)$  and the root of  $\tau_r(t)$  is labeled by  $(z, y)$  where  $z$  is a vertex in  $V$ . These vertices are called  $g$ -vertices.

A binary parse tree for a *GSP* graph  $G$  is illustrated in Figure 2. Note that there may exist several binary parse trees for a *GSP* and the binary parse tree is not necessarily unique. Given a *GSP* graph  $G$ , a binary parse tree is computable by a linear time algorithm according to the following Lemma.

**Lemma 2.5** ([8]). *For a given GSP graph  $G$ , a binary parse tree can be found in linear time.*

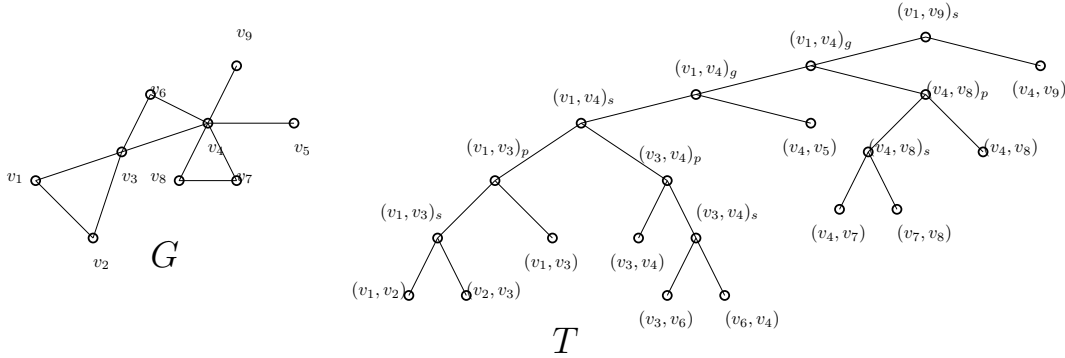


FIGURE 2. A binary parse tree for a *GSP* graph  $G$ .

### 3. A DYNAMIC PROGRAMMING ALGORITHM TO FIND A MINIMUM MIXED DOMINATING SET

In this section, we first set some necessary notations which are used throughout the section. Then, we present our proposed algorithm in details to find a  $\gamma_m$ -set, count them and computing  $\gamma_m(G)$  for a given *GSP* graph  $G$ .

Let  $t$  be a vertex in a parse tree  $T$ , for a *GSP* graph  $G$ , and  $\widehat{G}$  be a  $p$ -graph of subtree with root  $t$ . We define the sets  $ch(t)$  and  $\mathcal{MMD}_{i,j}(x, y)$  as follow:

- The set  $ch(t)$  consists of all children of  $t$ . In other words, in a parse tree  $T$ , if  $t$  is a leaf vertex, then  $ch(t)$  is an empty set and if it is an internal node, then  $ch(t)$  contains two elements.
- Let  $(x, y)$  be the label of  $t$  and  $i, j \in \{0, 1, 2, 3, 4, 5, 6\}$ . The set  $\mathcal{MMD}_{i,j}(x, y)$  is a  $\gamma_m$ -set for  $\widehat{G}$  where the label of the vertex  $t$  is  $(x, y)$ , and  $i, j$  satisfy one of the following conditions:
  - Case 0.:** If  $i = 0$ , then  $x \in \mathcal{MMD}_{i,j}(x, y)$  and at least one of its incident edges like  $e$  which belongs to  $\mathcal{MMD}_{i,j}(x, y)$ .
  - Case 1.:** If  $i = 1$ , then  $x \in \mathcal{MMD}_{i,j}(x, y)$  and none of its incident edges are in  $\mathcal{MMD}_{i,j}(x, y)$ .
  - Case 2.:** If  $i = 2$ , then  $x \notin \mathcal{MMD}_{i,j}(x, y)$  and at least one of its incident edges like  $e$  are in  $\mathcal{MMD}_{i,j}(x, y)$ .
  - Case 3.:** If  $i = 3$ , then  $x \notin \mathcal{MMD}_{i,j}(x, y)$  and none of its incident edges are in  $\mathcal{MMD}_{i,j}(x, y)$ , since all of them are dominated by an edge or a vertex in  $\mathcal{MMD}_{i,j}(x, y)$ . Moreover, there is a vertex like  $x'$  in  $\mathcal{MMD}_{i,j}(x, y)$  such that  $\{x, x'\} \in E(\widehat{G})$ .

**Case 4.:** If  $i = 4$ , then  $x \notin \mathcal{MMD}_{i,j}(x, y)$  and at least one of its incident edges are not dominated. Moreover, there is a vertex like  $x'$  in  $\mathcal{MMD}_{i,j}(x, y)$  such that  $\{x, x'\} \in E(\widehat{G})$ .

**Case 5.:** If  $i = 5$ , then  $x \notin \mathcal{MMD}_{i,j}(x, y)$  and none of its incident edges are in  $\mathcal{MMD}_{i,j}(x, y)$ , since all of them are dominated by an edge or a vertex in  $\mathcal{MMD}_{i,j}(x, y)$ . Moreover, there is no vertex like  $x'$  in  $\mathcal{MMD}_{i,j}(x, y)$  such that  $\{x, x'\} \in E(\widehat{G})$ .

**Case 6.:** If  $i = 6$ , then  $x \notin \mathcal{MMD}_{i,j}(x, y)$ , none of its incident edges are in  $\mathcal{MMD}_{i,j}(x, y)$ , and at least one of them is not dominated. Moreover, there is no vertex like  $x'$  in  $\mathcal{MMD}_{i,j}(x, y)$  such that  $\{x, x'\} \in E(\widehat{G})$ .

These cases can be defined for  $y$  based on  $j$ , similarly. We use  $\text{MINSIZE}(\dots)$  to denote a set with smallest cardinality among the input sets. With these definitions, our proposed algorithm constructs  $\mathcal{MMD}_{i,j}(x, y)$  as a mixed dominating set of minimum cardinality for graph  $G(x, y)$ . It also computes  $\mathcal{N}_{i,j}(x, y)$  as the number of minimal cardinality mixed dominating sets.

Now, we are ready to state our algorithm. The algorithm finds a binary parse tree like  $T$  for the input  $GSP$  graph  $G$  in linear time using the procedure described in [8]. Next, it traverses  $T$  in a bottom-up order. Each subtree in the parse tree corresponds to a  $p$ -graph for  $G$  and each vertex  $t$  of  $T$  is labeled by either  $(x, y)_i$ ,  $(x, y)_s$ ,  $(x, y)_p$  or  $(x, y)_g$  where  $x$  and  $y$  are terminals in the corresponding  $p$ -graph of  $\tau(t)$ .

For each visiting vertex  $t$ , one of the procedures  $\text{PROCESSLEAF}$ ,  $\text{PROCESSVERTEX}$ ,  $\text{PROCESSPVERTEX}$  or  $\text{PROCESSGVERTEX}$  is called based on the type of  $t$ . For each procedure, the input is consisted of vertices  $x$  and  $y$ . By traversing the parse tree  $T$  and calling proper procedures for each vertex, our algorithm finds a subset of  $\mathcal{MMD}_{i,j}(x, y) \subseteq V(\widehat{G})$  where for each  $i, j \in \{0, 1, 2, 3, 4, 5, 6\}$ , the set  $\mathcal{MMD}_{i,j}(x, y)$  stores a minimum mixed dominating set of  $\widehat{G}$  with the assumption that  $x$ ,  $y$  or some of their incident edges cannot be dominated.

After visiting the root node of  $T$  and computing  $\mathcal{MMD}_{i,j}(x, y)$  for it, a  $\gamma_m$ -set for  $G$  can be found. Finally, a  $\mathcal{MMD}_{i,j}(x, y)$  set with minimum cardinality is returned where  $i, j \in \{0, 1, 2, 3\}$ .

The input to the  $\text{PROCESSLEAF}$  procedure is a leaf vertex  $v \in V_T$  labeled by  $(x, y)_{leaf}$ , and the its output is a set  $\mathcal{MMD}_{i,j}(x, y)$  for  $i, j \in \mathcal{M}$ . Note that a leaf corresponds to an edge  $\{x, y\}$  in  $G$ . All valid cases for different  $i, j \in \mathcal{M}$  can be summarized as follows:

- (1) The vertices  $x$  and  $y$  and the edge  $\{x, y\}$  are dominated and at least one of them is a member of  $\mathcal{MMD}_{i,j}(x, y)$ . So  $i, j$  satisfies one of the following conditions:
  - $i = 0$  and  $j \in \{0, 2\}$ ,
  - $i = 1$  and  $j \in \{1, 3\}$ ,
  - $i = 2$  and  $j \in \{0, 2\}$ ,
  - $i = 3$  and  $j = 1$ .

- (2) The vertices  $x$  and  $y$  and the edge  $\{x, y\}$  are not dominated and are not members of  $\mathcal{MMD}_{i,j}(x, y)$ . So, we have  $i = j = 6$ .

Let  $v$  be a vertex of  $T$  labeled by  $(x, y)_s$ . In the PROCESSVERTEX procedure, we compute the set  $\mathcal{MMD}_{i,j}(x, y)$  for given terminal vertices  $x, y$  and a common vertex  $z$ . The sets  $\mathcal{MMD}_{i_\ell, j_\ell}^\ell(x, z)$  and  $\mathcal{MMD}_{i_r, j_r}^r(z, y)$  are corresponding to  $\tau_\ell(t)$  and  $\tau_r(t)$ , where the roots of  $\tau_\ell(t)$  and  $\tau_r(t)$  are labeled by  $(x, z)$  and  $(z, y)$ , respectively. The members of  $\mathcal{MMD}_{i_\ell, j_\ell}^\ell(x, z)$ ,  $\mathcal{MMD}_{i_r, j_r}^r(z, y)$ , and  $\mathcal{MMD}_{i,j}(x, y)$  are those vertices of  $T$  which are corresponding to  $p$ -graphs  $G_1 = (V_1, E_1, x, z)$ ,  $G_2 = (V_2, E_2, z, y)$  and  $\widehat{G} = G_1 o_s G_2 = (\widehat{V}, \widehat{E}, x, y)$ , respectively.

The possible cases based on belonging  $z$  to  $\mathcal{MMD}_{i,j}(x, y)$  and the vertex or edge dominating  $z$ , are summarized in Table 1. To be precise, consider the following cases:

- Case 0.:** Vertex  $z$  and at least one of its incident edges belong to  $\mathcal{MMD}_{i,j}(x, y)$ . So, we have  $(j_\ell, i_r) \in \{(0, 0), (0, 1), (1, 0)\}$ .
- Case 1.:** Vertex  $z \in \mathcal{MMD}_{i,j}(x, y)$  and none of its incident edges belong to  $\mathcal{MMD}_{i,j}(x, y)$  which implies  $j_\ell = i_r = 1$ .
- Case 2.:** Vertex  $z \notin \mathcal{MMD}_{i,j}(x, y)$  and an edge incident to  $z$  belong to  $\mathcal{MMD}_{i,j}(x, y)$ . So, we have either  $(j_\ell, i_r) \in \{2\} \times \{2, 3, 4, 5, 6\}$ , or  $(i_r, j_\ell) \in \{2\} \times \{2, 3, 4, 5, 6\}$ .
- Case 3.:** Vertex  $z$  and its incident edges does not belong to  $\mathcal{MMD}_{i,j}(x, y)$ . So, we have  $(j_\ell, i_r) \in \{(3, 3), (3, 5), (5, 3)\}$ .

Now, let  $v$  be a vertex of  $T$  labeled by  $(x, y)_p$ . The sets  $\mathcal{MMD}_{i_\ell, j_\ell}^\ell(x, y)$  and  $\mathcal{MMD}_{i_r, j_r}^r(x, y)$  correspond to  $\tau_\ell(t)$  and  $\tau_r(t)$ , respectively. For each  $i, j \in M$ , we describe a method for finding  $\mathcal{MMD}_{i,j}(x, y)$ . Note that it is enough to find a relation among the values  $(i, j)$ ,  $(i_\ell, j_\ell)$  and  $(i_r, j_r)$ . To do so, we use the procedure FINDLIST. Let the input to this procedure be a value like  $i \in M$ . Then, the procedure returns a set of pairs which are proper values for  $i_\ell$  and  $i_r$ . Note that for  $j \in M$ , the procedure returns proper  $j_\ell$  and  $j_r$ , similarly.

Note that  $\tau_\ell(t)$ ,  $\tau_r(t)$  and  $\tau(t)$  correspond to  $p$ -graphs  $G_1 = (V_1, E_1, x, y)$ ,  $G_2 = (V_2, E_2, x, y)$  and  $\widehat{G} = G_1 o_p G_2 = (\widehat{V}, \widehat{E}, x, y)$ , respectively. For  $i \in \mathcal{M}$  (resp.  $j \in \mathcal{M}$ ), the values of  $i_\ell$  and  $i_r$  (resp.  $j_\ell$  and  $j_r$ ) are determined as follows. The following cases are also illustrated in Table 2.

- Case 0.:**  $i = 0$  implies  $(i_\ell, i_r) \in \{(0, 0), (0, 1), (1, 0)\}$ ,
- Case 1.:**  $i = 1$  implies  $i_\ell = i_r = 1$ ,
- Case 2.:**  $i = 2$  implies either  $(i_\ell, i_r) \in \{2\} \times \{2, 3, 4, 5, 6\}$  or  $(i_\ell, i_r) \in \{2, 3, 4, 5, 6\} \times \{2\}$ ,
- Case 3.:**  $i = 3$  implies  $(i_\ell, i_r) \in \{(3, 3), (3, 5), (5, 3)\}$ ,
- Case 4.:**  $i = 4$  implies either  $(i_\ell, i_r) \in \{(3, 4), (3, 6), (4, 4), (4, 5), (4, 6)\}$  or  $(i_r, i_\ell) \in \{(3, 4), (3, 6), (4, 4), (4, 5), (4, 6)\}$ ,
- Case 5.:**  $i = 5$  implies  $i_\ell = i_r = 5$ ,
- Case 6.:**  $i = 6$  implies  $(i_\ell, i_r) \in \{(5, 6), (6, 5), (6, 6)\}$ .

---

```

1: procedure PROCESSVERTEX( $x, z, y$ )
2:   for all  $i, j \in M$  do
3:      $setlist \leftarrow \emptyset$ ;
4:      $Min \leftarrow \mathcal{MMD}_{i,0}^\ell(x, z) \cup \mathcal{MMD}_{0,j}^r(z, y)$ 
5:      $\mathcal{N}_{i,j}(x, y) \leftarrow \mathcal{N}_{i,0}^\ell(x, z) \times \mathcal{N}_{0,j}^r(z, y)$ 
6:     for all  $(j_\ell, i_r) \in \{0, 1\}$  do
7:       Add  $\mathcal{MMD}_{i,j_\ell}^\ell(x, z) \cup \mathcal{MMD}_{i_r,j}^r(z, y)$  to  $setlist$ ;
8:       PROCESSCALNUM ( $Min, \mathcal{N}_{i,j}(x, y), \mathcal{MMD}_{i,j_\ell}^\ell(x, z), \mathcal{MMD}_{i_r,j}^r(z, y), \mathcal{N}_{i,j_\ell}^\ell(x, z),$ 
 $\mathcal{N}_{i_r,j}^r(z, y)$ )
9:     end for
10:    for all  $(i_r) \in \{2, 3, 4, 5, 6\}$  do
11:      Add  $\mathcal{MMD}_{i,2}^\ell(x, z) \cup \mathcal{MMD}_{i_r,j}^r(z, y)$  to  $setlist$ ;
12:      PROCESSCALNUM ( $Min, \mathcal{N}_{i,j}(x, y), \mathcal{MMD}_{i,j_\ell}^\ell(x, z), \mathcal{MMD}_{i_r,j}^r(z, y), \mathcal{N}_{i,j_\ell}^\ell(x, z),$ 
 $\mathcal{N}_{i_r,j}^r(z, y)$ )
13:    end for
14:    for all  $(j_\ell) \in \{2, 3, 4, 5, 6\}$  do
15:      Add  $\mathcal{MMD}_{i,j_\ell}^\ell(x, z) \cup \mathcal{MMD}_{2,j}^r(z, y)$  to  $setlist$ ;
16:      PROCESSCALNUM ( $Min, \mathcal{N}_{i,j}(x, y), \mathcal{MMD}_{i,j_\ell}^\ell(x, z), \mathcal{MMD}_{i_r,j}^r(z, y), \mathcal{N}_{i,j_\ell}^\ell(x, z),$ 
 $\mathcal{N}_{i_r,j}^r(z, y)$ )
17:    end for
18:    Add  $\mathcal{MMD}_{i,3}^\ell(x, z) \cup \mathcal{MMD}_{3,j}^r(z, y)$  to  $setlist$ ;
19:    PROCESSCALNUM ( $Min, \mathcal{N}_{i,j}(x, y), \mathcal{MMD}_{i,j_\ell}^\ell(x, z), \mathcal{MMD}_{i_r,y_j}^r(z, y), \mathcal{N}_{i,j_\ell}^\ell(x, z),$ 
 $\mathcal{N}_{i_r,j}^r(z, y)$ )
20:    Add  $\mathcal{MMD}_{i,3}^\ell(x, z) \cup \mathcal{MMD}_{5,j}^r(z, y)$  to  $setlist$ ;
21:    PROCESSCALNUM ( $Min, \mathcal{N}_{i,j}(x, y), \mathcal{MMD}_{i,j_\ell}^\ell(x, z), \mathcal{MMD}_{i_r,j}^r(z, y), \mathcal{N}_{i,j_\ell}^\ell(x, z),$ 
 $\mathcal{N}_{i_r,j}^r(z, y)$ )
22:    Add  $\mathcal{MMD}_{i,5}^\ell(x, z) \cup \mathcal{MMD}_{3,j}^r(z, y)$  to  $setlist$ ;
23:    PROCESSCALNUM ( $Min, \mathcal{N}_{i,j}(x, y), \mathcal{MMD}_{i,j_\ell}^\ell(x, z), \mathcal{MMD}_{i_r,j}^r(z, y), \mathcal{N}_{i,j_\ell}^\ell(x, z),$ 
 $\mathcal{N}_{i_r,j}^r(z, y)$ )
24:     $\mathcal{MMD}_{i,j}(x, y) \leftarrow \text{Minsize}(setlist)$ ;
25:  end for
26: end procedure

```

---



---

```

1: procedure FINDLIST( $k$ )
2:    $M \leftarrow \{0, 1, 2, 3, 4, 5, 6\}$ 
3:   for all  $k \in M$  do
4:      $list \leftarrow \emptyset$ 
5:   end for
6:   switch  $k$  do
7:     case 0
8:        $list \leftarrow \{(0, 0), (0, 1), (1, 0)\}$ 
9:     case 1
10:       $list \leftarrow \{(1, 1)\}$ 
11:    case 2
12:      for all  $k' \in M \setminus \{0, 1\}$  do
13:        Add  $(2, k')$  to  $list$ 
14:        Add  $(k', 2)$  to  $list$ 
15:      end for
16:    case 3
17:       $list \leftarrow \{(3, 3), (3, 5), (5, 3)\}$ 
18:    case 4
19:      for all  $k' \in M \setminus \{0, 1, 2\}$  do
20:        Add  $(4, k')$  to  $list$ 
21:        Add  $(k', 4)$  to  $list$ 
22:      end for
23:    case 5
24:      Add  $(5, 5)$  to  $list$ 
25:    case 6
26:      Add  $(5, 6), (6, 5), (6, 6)$  to  $list$ 
27: end procedure

```

---

Let  $v$  be a vertex of  $T$  labeled  $(x, y)_g$ . In the procedure PROCESSGVERTEX, the set  $\mathcal{MMD}_{i,j}(x, y)$  is computed for the given vertices  $x$  and  $y$ . The sets  $\mathcal{MMD}_{i_\ell, j_\ell}^\ell(x, y)$  and  $\mathcal{MMD}_{i_r, j_r}^r(x, y)$  correspond to  $\tau_\ell(t)$  and  $\tau_r(t)$ , respectively.

Let the roots of  $\tau_\ell(t)$  and  $\tau_r(t)$  be labeled by  $(x, y)$  and  $(y, z)$ , respectively, for some  $z \in V$ . Also, assume that  $\mathcal{MMD}_{i_\ell, j_\ell}^\ell(x, y)$  and  $\mathcal{MMD}_{i_r, j_r}^r(x, y)$  are the associated sets with the vertices  $(x, y)$  and

---

```

1: procedure PROCESSPVERTEX( $x, y$ )
2:   for all  $i, j \in M$  do
3:      $list1 \leftarrow ProcessFindlist(i)$ 
4:      $list2 \leftarrow ProcessFindlist(j)$ 
5:      $(i_{\ell_1}, j_{\ell_1}), (i_{r_1}, j_{r_1}) \leftarrow$  an arbitrary element of  $list1 \times list2$ 
6:      $Min \leftarrow \mathcal{MMD}_{i_{\ell_1}, j_{\ell_1}}^{\ell}(x, y) \cup \mathcal{MMD}_{i_{r_1}, j_{r_1}}^r(x, y)$ 
7:      $\mathcal{N}_{i,j}(x, y) \leftarrow \mathcal{N}_{i_{\ell_1}, j_{\ell_1}}^{\ell}(x, y) \times \mathcal{N}_{i_{r_1}, j_{r_1}}^r(x, y)$ 
8:     for all  $(i_{\ell}, j_{\ell}), (i_r, j_r) \in list1 \times list2$  do
9:       Add  $\mathcal{MMD}_{i_{\ell}, j_{\ell}}^{\ell}(x, y) \cup \mathcal{MMD}_{i_r, j_r}^r(x, y)$  to  $setlist$ ;
10:      PROCESSCALNUM ( $Min, \mathcal{N}_{i,j}(x, y), \mathcal{MMD}_{i_{\ell}, j_{\ell}}^{\ell}(x, y), \mathcal{MMD}_{i_r, j_r}^r(x, y),$ 
 $\mathcal{N}_{i_{\ell}, j_{\ell}}^{\ell}(x, y), \mathcal{N}_{i_r, j_r}^r(x, y)$ )
11:    end for
12:     $\mathcal{MMD}_{i,j}(x, y) \leftarrow Minsize(setlist)$ ;
13:  end for
14: end procedure

```

---

```

1: procedure PROCESSGVERTEX( $x, y$ )
2:   for all  $i, j \in M$  do
3:      $list1 \leftarrow ProcessFindlist(j)$ ;
4:      $j_{\ell_1}, i_{r_1} \leftarrow$  an arbitrary element of  $list1$ 
5:      $Min \leftarrow \mathcal{MMD}_{i, j_{\ell_1}}^{\ell}(x, z) \cup \mathcal{MMD}_{i_{r_1}, j_r}^r(z, y)$ 
6:      $\mathcal{N}_{i,j}(x, y) \leftarrow \mathcal{N}_{i, j_{\ell_1}}^{\ell}(x, z) \times \mathcal{N}_{i_{r_1}, j_r}^r(z, y)$ 
7:     for all  $((j_{\ell}, i_r), j_r) \in list1 \times \{0, 1, 2, 3\}$  do
8:       Add  $\mathcal{MMD}_{i, j_{\ell}}^{\ell}(x, y) \cup \mathcal{MMD}_{i_r, j_r}^r(x, y)$  to  $setlist$ ;
9:       PROCESSCALNUM ( $Min, \mathcal{N}_{i,j}(x, y), \mathcal{MMD}_{i, j_{\ell}}^{\ell}(x, z), \mathcal{MMD}_{i_r, j_r}^r(x, y),$ 
 $\mathcal{N}_{i, j_{\ell}}^{\ell}(x, z), \mathcal{N}_{i_r, j_r}^r(x, y)$ )
10:    end for
11:     $\mathcal{MMD}_{i,j}(x, y) \leftarrow Minsize(setlist)$ ;
12:  end for
13: end procedure

```

---

$(y, z)$  of  $T$ . It is obvious that  $z$  does not appear in ancestors of  $t$  in the parse tree. So,  $z$  and all of its incident edges must be closely dominated which implies  $j \in \{0, 1, 2, 3\}$ . Since  $y$  is the common vertex between  $G_1$  and  $G_2$ , based on  $j$ , the set denoted by  $list1$ , which equals the set of possible pairs, can be

computed for  $j_\ell$  and  $i_r$  by procedure FINDLIST. Several cases are possible for  $y$  which are shown in Table 3 which are discussed below:

**Case 0.:**  $j = 0$  implies  $(j_\ell, i_r) \in \{(0, 0), (1, 0), (0, 1)\}$ ,

**Case 1.:**  $j = 1$  implies  $j_\ell = i_r = 1$ ,

**Case 2.:**  $j = 2$  implies either  $(j_\ell, i_r) \in \{2\} \times \{2, 3, 4, 5, 6\}$  or  $(j_\ell, i_r) \in \{2, 3, 4, 5, 6\} \times \{2\}$ ,

**Case 3.:**  $j = 3$  implies  $(j_\ell, i_r) \in \{(3, 3), (3, 5), (5, 3)\}$ ,

**Case 4.:**  $j = 4$  implies either  $(j_\ell, i_r) \in \{3\} \times \{4, 6\}$ ,  $(j_\ell, i_r) \in \{4\} \times \{4, 5, 6\}$ ,  $(i_r, j_\ell) \in \{3\} \times \{4, 6\}$  or  $(i_r, j_\ell) \in \{4\} \times \{4, 5, 6\}$ ,

**Case 5.:**  $j = 5$  implies  $j_\ell = i_r = 5$ ,

**Case 6.:**  $j = 6$  implies  $(j_\ell, i_r) \in \{(5, 6), (6, 5), (6, 6)\}$ .

By  $MMD_{i,j}(x, y) \leftarrow \text{MINSIZE}(\text{Setlist})$ , we remove all undefinable sets from *Setlist*. If *Setlist* is empty, then  $MMD_{i,j}(x, y)$  becomes undefinable.

---

```

1: procedure PROCESSCALNUM(Min, N, S1, S2, s'1, s'2)
2:   if  $|S_1 \cup S_2| \leq \textit{Min}$  then
3:      $N \leftarrow s'_1 \times s'_2$ 
4:   else if  $|S_1 \cup S_2| = \textit{Min}$  then
5:      $N \leftarrow N + s'_1 \times s'_2$ 
6:   end if
7: end procedure

```

---

We study the correctness and complexity of our proposed algorithm below.

**Theorem 3.1.** *For a given generalized series-parallel graph  $G = (V, E)$ , the Algorithm 1 finds a  $\gamma_m$ -set for  $G$  in time  $O(|V|)$ .*

*Proof.* In Algorithm1, we traverse the parse tree  $T$  in a bottom-up fashion and compute at most 49 sets for each of its internal vertices. Each initial set for the leaves of the tree represents all possible mixed dominating sets in a graph consisting of only one edge. Let  $G_1 = (V_1, E_1)$  and  $G_2 = (V_2, E_2)$  be the graphs represented by the subtrees  $\tau_\ell(t)$  and  $\tau_r(t)$ , respectively. Assume that they are given to the procedures PROCESSLEAF, PROCESSVERTEX, PROCESSPVERTEX and PROCESSGVERTEX. It is easy to see that these procedures find all possible  $\gamma_m$ -sets in each corresponding graph. Finally, our algorithm extracts only a valid minimum mixed dominating set. The steps of the algorithm require at most  $|O(V_T)|$  operations. Since each binary tree with  $n$  leaves has  $O(n)$  vertices and the binary parse tree of every  $GSP$  graph  $G$  has  $|E(G)|$  leaves. So, we have  $|V_T| \in O(|E(G)|)$ . Every  $GSP$  graph  $G$  is planar. In a planar graph, we have  $|E| \leq 3|V| - 6$ . Also, we know that a parse tree  $T$  can be constructed in  $O(|V|)$ [9]. So, the algorithm computes a  $\gamma_m$ -set for a given  $GSP$  graph  $G$  in time  $O(|V|)$ .  $\square$

---

```

1: Find a parse tree of  $G$  like  $T$ 
2: for each  $v$  in the post order traversal of the parse tree do
3:   switch type of  $v$  do
4:     case Leaf
5:       PROCESSLEAF( $x, y$ )  $\triangleright (x, y)_i$  is the label of  $v$ 
6:     case  $s$  – vertex
7:       PROCESSVERTEX( $x, z, y$ )  $\triangleright (x, y)_s, (x, z)$  and  $(z, y)$  are labels of  $v$ , the left and the
       right children of  $v$ , respectively.
8:     case  $p$  – vertex
9:       PROCESSPVERTEX( $x, y$ )  $\triangleright (x, y)_p$  is the label of  $v$  and the labels of left and right
       children of  $v$  are  $(x, y)$ .
10:    case  $g$  – vertex
11:      PROCESSGVERTEX( $x, y, z$ )  $\triangleright (x, y)_s, (x, y)$  and  $(y, z)$  are labels of  $v$ , the left and the
       right children of  $v$ , respectively.
12:   end for
13:  $D \leftarrow \emptyset$ 
14:  $Min \leftarrow \mathcal{MMD}_{0,0}(x, y)$ 
15: for all  $i, j \in \{0, 1, 2, 3\}$  do
16:   Add  $\mathcal{MMD}_{i,j}(x, y)$  to  $D$ 
17:   if  $|\mathcal{MMD}_{i,j}(x, y)| \leq Min$  then
18:      $N_{\gamma_m} \leftarrow N(x_i, y_j)$ 
19:   else if  $|\mathcal{MMD}_{i,j}(x, y)| = Min$  then
20:      $N_{\gamma_m} \leftarrow N_{\gamma_m} + N(x_i, y_j)$ 
21:   end if
22: end for
23:  $\gamma_m\text{-set} \leftarrow \text{Minsize}(D)$ 
24:  $\gamma_m(G) \leftarrow |\gamma_m\text{-set}|$ 

```

---

#### 4. ACKNOWLEDGMENTS

This article has been written while the fourth author was in a sabbatical visit to University of Auckland. He would like to express his gratitude to Prof. Cristian S. Calude and his research group for the nice and friendly hospitality.

**Algorithm 1** Finding a  $\gamma_m$ -sets of a *GSP* graph  $G$ 


---

```

1: procedure PROCESSLEAF( $x, y$ )
2:   for all  $i, j \in \{0, 1, 2, 3, 4, 5, 6\}$  do
3:      $\mathcal{MMD}_{i,j}(x, y) \leftarrow NaN;$ 
4:      $\mathcal{N}_{i,j}(x, y) = 0;$ 
5:   end for
6:    $\mathcal{MMD}_{0,0}(x, y) \leftarrow \{x, y, xy\}$  and  $\mathcal{N}_{0,0}(x, y) = 1;$ 
7:    $\mathcal{MMD}_{0,2}(x, y) \leftarrow \{x, xy\}$  and  $\mathcal{N}_{0,2}(x, y) = 1;$ 
8:    $\mathcal{MMD}_{1,1}(x, y) \leftarrow \{x, y\}$  and  $\mathcal{N}_{1,1}(x, y) = 1;$ 
9:    $\mathcal{MMD}_{1,3}(x, y) \leftarrow \{x\}$  and  $\mathcal{N}_{1,3}(x, y) = 1;$ 
10:   $\mathcal{MMD}_{2,0}(x, y) \leftarrow \{y, xy\}$  and  $\mathcal{N}_{2,0}(x, y) = 1;$ 
11:   $\mathcal{MMD}_{2,2}(x, y) \leftarrow \{xy\}$  and  $\mathcal{N}_{2,2}(x, y) = 1;$ 
12:   $\mathcal{MMD}_{3,1}(x, y) \leftarrow \{y\}$  and  $\mathcal{N}_{3,1}(x, y) = 1;$ 
13:   $\mathcal{MMD}_{6,6}(x, y) \leftarrow \emptyset$  and  $\mathcal{N}_{6,6}(x, y) = 1.$ 
14: end procedure

```

---

## REFERENCES

- [1] G. S. Adhar, S. Peng, Mixed domination in trees: a parallel algorithm, *Congr. Numer.* **100** (1994) 73-80.
- [2] Y. Alavi, M. Behzad, L. M. Lesniak-Foster, E. Nordhaus, Total matchings and total coverings of graphs, *J. Graph Theory* **1(2)** (1977) 135-140.
- [3] Y. Alavi, J. Liu, J. Wang, Z. Zhang, On total covers of graphs, *Discrete Math.* **100** (1992) 229-233.
- [4] P. Chebolu, M. Cryan, R. Martin, Exact counting of Euler tours for generalized series-parallel graphs, *J. Discrete Algorithms* **10** (2012) 110-122.
- [5] T. W. Haynes, S. Hedetniemi, P. Slater, *Fundamentals of domination in graphs*, CRC Press, 1998.
- [6] T. W. Haynes, S. Hedetniemi, P. Slater, *Domination in graphs: advanced topics*, Taylor & Francis, 1998.
- [7] S. M. Hedetniemi, S. T. Hedetniemi, R. Laskar, A. McRae, A. Majumdar, Domination, independence and irredundance in total graphs: a brief survey, *Graph Theory, Combinatorics and Applications: Proceedings of the 7th Quadrennial International Conference on the Theory and Applications of Graphs* **2** (1995) 671-683.
- [8] J. E. Hopcroft, R. E. Tarjan, Dividing a graph into triconnected components, *SIAM J. Comput.* **2(3)** (1973) 135-158.
- [9] T. Kikuno, N. Yoshida, Y. Kakuda, A linear algorithm for the domination number of a series-parallel graph, *Discrete Appl. Math.* **5(3)** (1983) 299-311.
- [10] J. K. Lan, G. J. Chang, On the mixed domination problem in graphs, *Theoret. Comput. Sci.* **476(84)** (2013) 84-93.
- [11] A. Majumdar, *Neighborhood Hypergraphs: A Framework for Covering and Packing Parameters in Graphs*, PhD thesis, Clemson University, Department of Mathematical Sciences, South Carolina, 1992.
- [12] D. F. Manlove, On the algorithmic complexity of twelve covering and independence parameters of graphs, *Discrete Appl. Math.* **91(1)** (1999) 155-175.

- [13] M. Rajaati, M. R. Hooshmandasl, M. J. Dinneen, A. Shakiba, On fixed-parameter tractability of the mixed domination problem for graphs with bounded tree-width, *Discrete Math. Theor. Comput. Sci.* **20(2)** (2018) 1-25.
- [14] D. B. West, Introduction to graph theory, Prentice Hall Upper Saddle River, 2001.
- [15] Y. Zhao, L. Kang, M. Y. Sohn, The algorithmic complexity of mixed domination in graphs, *Theoret. Comput. Sci.* **412(22)** (2011) 2387-2392.

TABLE 1. Different situations for  $s$ -vertices

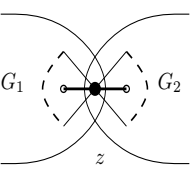
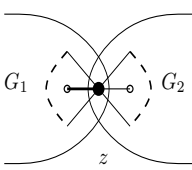
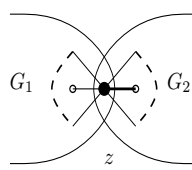
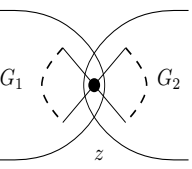
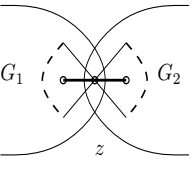
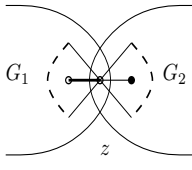
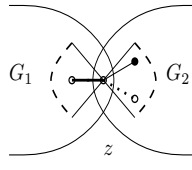
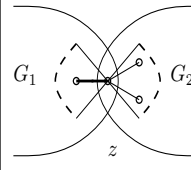
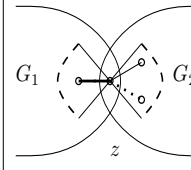
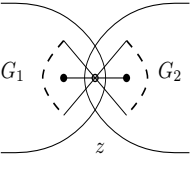
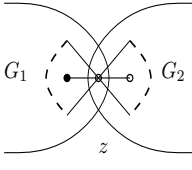
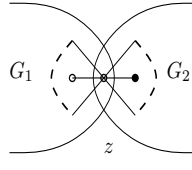
Case	(a)	(b)	(c)	(d)	(e)
0					
	$j_\ell = 0, i_r = 0$	$j_\ell = 0, i_r = 1$	$j_\ell = 1, i_r = 0$		
1					
	$j_\ell = 1, i_r = 1$				
2					
	$j_\ell = 2, i_r = 2$	$j_\ell = 2, i_r = 3$	$j_\ell = 2, i_r = 4$	$j_\ell = 2, i_r = 5$	$j_\ell = 2, i_r = 6$
		$j_\ell = 3, i_r = 2$	$j_\ell = 4, i_r = 2$	$j_\ell = 5, i_r = 2$	$j_\ell = 6, i_r = 2$
3					
	$j_\ell = 3, i_r = 3$	$j_\ell = 3, i_r = 5$	$j_\ell = 5, i_r = 3$		

TABLE 2. Different situations for  $p$ -vertices

Case	(a)	(b)	(c)	(d)	(e)
0					
	$j_\ell = 0, i_r = 0$	$j_\ell = 0, i_r = 1$	$j_\ell = 1, i_r = 0$		
1					
	$j_\ell = 1, i_r = 1$				
2					
	$j_\ell = 2, i_r = 2$	$j_\ell = 2, i_r = 3$	$j_\ell = 2, i_r = 4$	$j_\ell = 2, i_r = 5$	$j_\ell = 2, i_r = 6$
		$j_\ell = 3, i_r = 2$	$j_\ell = 4, i_r = 2$	$j_\ell = 5, i_r = 2$	$j_\ell = 6, i_r = 2$
3					
	$j_\ell = 3, i_r = 3$	$j_\ell = 3, i_r = 5$	$j_\ell = 5, i_r = 3$		
4					
	$j_\ell = 3, i_r = 4$	$j_\ell = 3, i_r = 6$	$j_\ell = 4, i_r = 4$	$j_\ell = 4, i_r = 5$	$j_\ell = 4, i_r = 6$
	$j_\ell = 4, i_r = 3$	$j_\ell = 6, i_r = 3$		$j_\ell = 5, i_r = 4$	$j_\ell = 6, i_r = 4$
5					
	$j_\ell = 5, i_r = 5$				
6					
	$j_\ell = 6, i_r = 5$	$j_\ell = 5, i_r = 6$	$j_\ell = 6, i_r = 6$		

TABLE 3. Different situations for  $g$ -vertices

Case	(a)	(b)	(c)	(d)	(e)
0					
	$j_\ell = 0, i_r = 0$	$j_\ell = 0, i_r = 1$	$j_\ell = 1, i_r = 0$		
1					
	$j_\ell = 1, i_r = 1$				
2					
	$j_\ell = 2, i_r = 2$	$j_\ell = 2, i_r = 3$	$j_\ell = 2, i_r = 4$	$j_\ell = 2, i_r = 5$	$j_\ell = 2, i_r = 6$
		$j_\ell = 3, i_r = 2$	$j_\ell = 4, i_r = 2$	$j_\ell = 5, i_r = 2$	$j_\ell = 6, i_r = 2$
3					
	$j_\ell = 3, i_r = 3$	$j_\ell = 3, i_r = 5$	$j_\ell = 5, i_r = 3$		
4					
	$j_\ell = 3, i_r = 4$	$j_\ell = 3, i_r = 6$	$j_\ell = 4, i_r = 4$	$j_\ell = 4, i_r = 5$	$j_\ell = 4, i_r = 6$
	$j_\ell = 4, i_r = 3$	$j_\ell = 6, i_r = 3$		$j_\ell = 5, i_r = 4$	$j_\ell = 6, i_r = 4$
5					
	$j_\ell = 5, i_r = 5$				
6					
	$j_\ell = 6, i_r = 5$	$j_\ell = 5, i_r = 6$	$j_\ell = 6, i_r = 6$		

**M.Rajaati**

Department of Computer Science

Yazd University



Yazd, Iran.

m.rajaati@stu.yazd.ac.ir

**M. R. Hooshmandasl**

Department of Computer Science

Yazd University

Yazd, Iran.

hooshmandasl@yazd.ac.ir

**A. Shakiba**

Department of Computer Science

Vali-e-Asr University of Rafsanjan

Rafsanjan, Iran.

ali.shakiba@vru.ac.ir

**P. Sharifani**

Department of Computer Science

Yazd University

Yazd, Iran.

pouyeh.sharifani@gmail.com

**M. J. Dinneen**

Department of Computer Science

The University of Auckland

Auckland, New Zealand.

m.dinneen@auckland.ac.nz