Acta Univ. Sapientiae Informatica **15,** 1 (2023) 124–145

DOI: 10.2478/ausi-2023-0010

On agglomeration-based rupture degree in networks and a heuristic algorithm

Muammer AĞTAŞ
Pamukkale University
Faculty of Engineering
Department of Computer Engineering
20160 Denizli, Turkey
email: magtas21@posta.pau.edu.tr

Tufan TURACI
Pamukkale University
Faculty of Engineering
Department of Computer Engineering
20160 Denizli, Turkey
email: tturaci@pau.edu.tr

Abstract. The rupture degree is one the most important vulnerability parameter in networks which are modelled by graphs. Let $G(V(G),E(G))$ be a simple undirected graph. The rupture degree is defined by $r(G) = \max\{w(G–S)–|S|–m(G–S){:}S\subset V(G)$ and $w(G–S)>1\}$ where m$(G–S)$ is the order of a largest connected component in $G–S$ and $w(G–S)$ is the number of components of $G–S$, respectively. In this paper, we consider the vertex contraction method based on the network agglomeration operation for each vertex of G. Then, we have presented two graph vulnerability parameters called by *agglomeration rupture degree* and *average lower agglomeration rupture degree*. Furthermore, the exact values of them for some graph families are given. Finally, we proposed a polynomial time heuristic algorithm to obtain the values of *agglomeration rupture degree* and *average lower agglomeration rupture degree*.

1 Introduction

Networks can be modeled with graphs. The servers or hubs are illustrated by vertices and edges are connecting medium between them in any graph G. The

Key words and phrases: graphs, network design and communication, complex networks, connectivity, vulnerability, rupture degree, agglomeration.

vulnerability of a network is of main significance to network planners according to the nodes and links [7, 12]. Recently, networks vulnerability has been studied in widespread multidisciplinary area such as informatics, mathematics, computer science, chemistry and many other applied science and engineering science. The vulnerability value of networks is defined as the durability of the network after the breakdown of some vertices or edges until a communication disruption [12, 22].

In this paper, we consider only simple graphs. Now, some notations will be given. Let $G(V(G),E(G))$ be a simple connected graph whose vertex and edge sets are denoted by $V(G)$ and $E(G)$, where $V(G)=\{v_1,v_2,\ldots,v_n\}$, $E(G)=\{e_1,e_2,\ldots,e_m\}$, $|V(G)|=n$ and $|E(G)|=m$. Let $u\in V(G)$.The set $N(u)=\{v\in V(G)|(u,v)\in E(G)\}$ is called the *open neighborhood* of u. Furthermore, the number of $|N(u)|$ is called the degree of vertex u and is denoted by $d_G(u)$. The *maximum degree* of G is denoted by $\Delta(G)$ is defined by $\max\{d_G(v) \mid v\in V(G)\}$. Similarly, the *minimum degree* of G is denoted by $\delta(G)$ is defined by $\min\{d_G(v) \mid v\in V(G)\}$ [18, 33]. The set $N[u]=\{u\}\cup N(u)$ is called the *closed neighborhood* of the vertex u. The $d(u,v)$ represents the distance between two vertices as u and v. Furthermore, the distance is defined as the length of a shortest path between them [18, 33].

The connectivity value of any graph G is the best-known vulnerability measures in the literature. It is defined that to obtain disconnected graph after the minimum number of vertices are deleted from the given graph, also is denoted by $k(G)$ for any graph G [16]. The connectivity of any graph G is computed with polynomial time. There are many vulnerability measures for the networks. For example, integrity [9], toughness [12], tenacity [13], global distribution number [14] are considered and studied in many areas. Furthermore, there are many average vulnerability parameters are proposed to obtain the vulnerability values of the networks. For example, the average lower domination number [17], the average lower independence number [6], the average lower bondage number [32], the average lower reinforcement number [31], the average lower residual domination number [29], the average lower link residual domination numbers [30] etc. are considered and studied in many areas. The values of these parameters are not computed in polynomial time. Because they are classes of NP-Hard or NP-Complete.

The rupture degree is other best known vulnerability parameter. It is defined by Li et al. in [23] and the definition of it as the following:

$$r(G) = \max\{w(G-S)-|S|-m(G-S):S\subset V(G) \text{ and } w(G-S)>1\},$$

where $m(G\text{–}S)$ and $w(G\text{–}S)$ denote the is the largest connected component in $G\text{–}S$ and the number of components of the graph $G\text{–}S$, respectively.

Let C_6 be a cycle graph. It is showed by in Figure 1. The alternative rupture-sets of C_6 are showed with the set of darkened vertices. Clearly, $|S|$=3, $w(C_6 - S) = 3$ and $m(C_6 - S) = 1$. As a result,$r(C_6)$=1 is obtained.

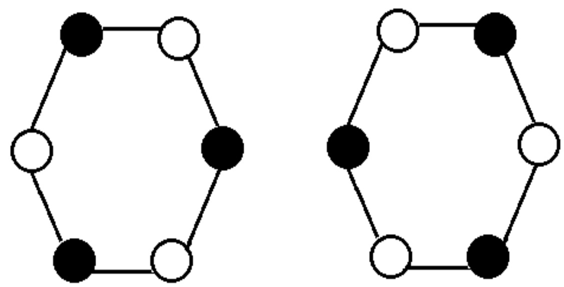

Figure 1: The rupture-set of the graph C_6

In [26], the authors showed that calculating the rupture degree problem is an NP-complete problem. However, it is possible to determine the rupture degree of large classes of graphs. For more results on rupture degree, we refer the readers to see [1, 2, 3, 4, 5, 8, 20, 21, 25, 27]. Furthermore, Li gave an algorithm whose complexity is $O(n^2)$ for isolating rupture degree in Trees of order n [24]. Another interesting study about the rupture degree is the references [15] by Durgut et al. In [15], a heuristic algorithm is given to find the rupture degree for any graph G. A similar study is in [11], where Berberler et al. gave a polynomial time heuristic algorithm for computing the integrity of any given graph G.

When investigating the vulnerability of complex networks, the finding node importance is used for each node recently. There are some different methods for determining importance of each node. In this paper, we use node contraction method based on network agglomeration. Then, new two vulnerability parameter definitions have been made by combining node contraction method based on network agglomeration method and the rupture degree. By using methods based on agglomeration, more efficient results can be obtained in terms of vulnerability. Let $v_i \in V(G)$. The node contraction is defined as follows: the node v_i and other $d_G(v_i)$ nodes connected with v_i into a new node $v_i^{'}$, which takes place of the primary $d_G(v_i)$+1 nodes, and links connected with

$d_G(v_i)$–1 nodes originally turn to the new node $v_i^{'}$ now. For example, if the center node is contracted in a star-network, the network is agglomerated to one node. Another example can be seen in Figure 2.

The agglomeration operation has been used in different network vulnerability measures, some of these can be seen in [10, 19, 28]. In this paper, we incorporate the concept of the rupture degree and agglomeration operation, as well as the idea of the average vulnerability parameters, to introduce new graph parameters called the agglomeration rupture degree (ARD), denoted by $r^{agg}(G)$, and the average lower agglomeration rupture degree (ALARD), denoted by $r_{av}^{agg}(G)$, for any given graph G. Furthermore, we consider the ARD and ALARD to be two metrics for network vulnerability.

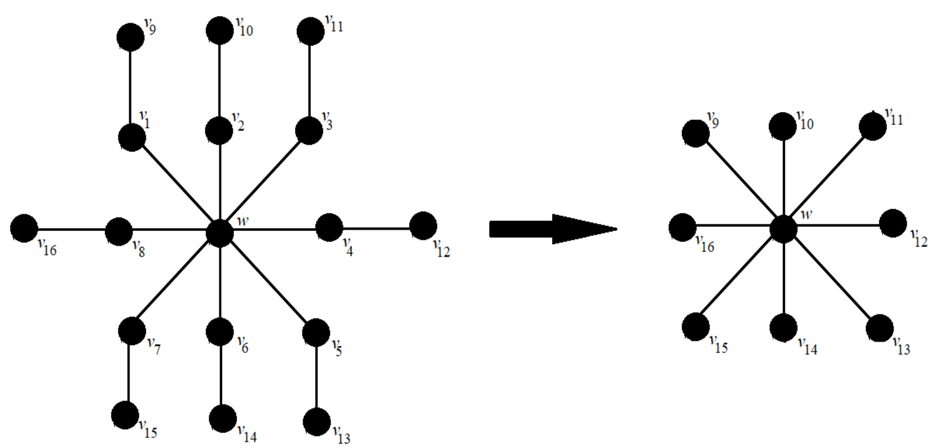

Figure 2: The agglomeration operation on the vertex w.

In this paper, there are 6-Sections. The ARD and ALARD are defined in Section 2. In Section 3, the difference of the ARD and ALARD is shown with different examples. The values of ARD and ALARD are obtained some well-known graph families in Section 4. In Section 5, we give a polynomial time heuristic algorithm to compute the values of ARD and ALARD, then the computational test results are presented. Finally, we give our conclusions in Section 6.

2 The definitions of the ARD and ALARD

The definitions of ARD and ALARD are given in this section. For a vertex v_k of a graph G, the *lower agglomeration rupture degree*, denoted by $r^{agg}_{v_k}(G)$, is the minimum cardinality of the *rupture set* in G derived from the graph G after the agglomeration operation for the vertex v_k. The *agglomeration rupture degree* of a graph G is defined as:

$$r^{agg}(G) = max_{v_k \in V(G)}\{r^{agg}_{v_k}(G)\}.$$

Furthermore, the *average lower rupture degree* of G is defined by

$$r^{agg}_{av}(G) = \frac{1}{|V(G)|}\sum_{v_k \in V(G)} r^{agg}_{v_k}(G).$$

Example 1.1. Let the graph G, which are showed in Figure3, be a graph with 6-vertices and 6-edges. Clearly, the connectivity number and the rupture degree of G is one. The rupture set of G is $\{v_1,v_4\}$ and $r(G)$=1.

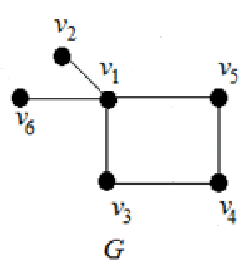

Figure 3: The graph G whose number of vertices and edges is 6.

Vertices	$r^{agg}_{v_k}(G)$
v_1	-1
v_2	0
v_3	1
v_4	1
v_5	1
v_6	0

Table 1: The lower agglomeration rupture degree of every vertex $v_k \in V(G)$

The lower agglomeration rupture degree of every vertex $v_k \in V(G)$ is presented in Table 1.

Clearly, we have $r^{agg}_{v_1}(G)$=-1, $r^{agg}_{v_2}(G)$=0, $r^{agg}_{v_3}(G)$=1, $r^{agg}_{v_4}(G)$=1, $r^{agg}_{v_5}(G)$=1, and $r^{agg}_{v_6}(G)$=0. Thus, $r^{agg}(G)$=1 and $r^{agg}_{av}(G)$=(-1+0+1+1+1+0)/6=2/6=0.33 are obtained.

3 Vulnerability examples of the ARD and ALARD

The ARD and ALARD can be more efficient than the connectivity and the rupture degree in measuring the vulnerability of some graphs. In this section, this situation is showed with different two examples.

In the first example, we consider the graphs G_1 and G_2 that are presented in Figure 4. Then, we want to show the values of ARD and ALARD can be used to distinguish between two given graphs. Clearly, the values of connectivity and domination number, and also the numbers of vertices and edges of the graphs G_1 and G_2 are equal. That is, $k(G_1)$=$k(G_2)$=1, $r(G_1)$=$r(G_2)$=1, $|V(G_1)|$=$|V(G_2)|$=8 and $|E(G_1)|$=$|E(G_2)|$=8.

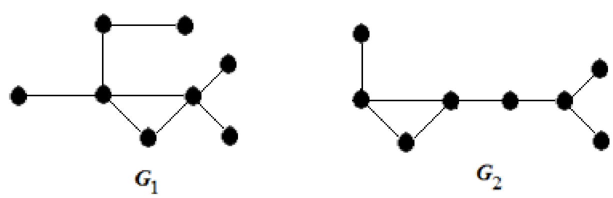

Figure 4: The graphs G_1 and G_2 with 8-vertices and 8-edges

The ARD of the graphs G_1 and G_2 are $r^{agg}(G_1)$=2 and $r^{agg}(G_2)$=1, while the ALARDs of these two graphs G_1 and G_2 are $r^{agg}_{av}(\mathrm{G}_1) = \frac{1}{2}$ and $r^{agg}_{av}(\mathrm{G}_2) = \frac{1}{4}$, respectively.

In the second example, we consider the graphs G_3 and G_4 that are presented in Figure 5. Then, we want to show the value of ALARD can be used to distinguish between two given graphs. Clearly, the values of connectivity, the rupture degree and the agglomeration rupture degree of the graphs G_3 and G_4 are equal, with $k(G_3)$=$k(G_4)$=1, $r(G_3)$=$r(G_4)$=1 and $r^{agg}(G_3)$= $r^{agg}(G_4)$=1. Additionally, the numbers of vertices and edges of the graphs G_3 and G_4 are equal as like $|V(G_3)|$=$|V(G_4)|$=6 and $|E(G_3)|$=$|E(G_4)|$=6.

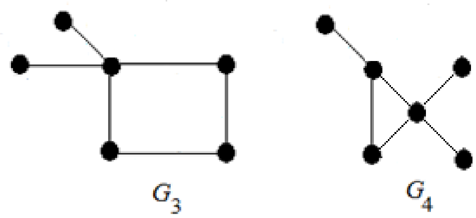

Figure 5: The graphs G_3 and G_4 with 6-vertices and 6-edges

The ALARD of the graphs G_3 and G_4 are $r_{av}^{agg}(\mathrm{G}_3) = \frac{1}{3}$ and $r_{av}^{agg}(\mathrm{G}_4) = 0$, respectively.

With these examples, we can say that these two new parameters ARD and ALARD may be more distinctive than other vulnerability parameters.

4 Computing the ARD and ALARD of well-known graphs

In this section, we compute the values of ARD and ALARD of well-known graphs such as the path graph P_n, the cycle graph C_n , the complete graph K_n, the star graph $K_{1,n-1}$, the wheel graph $W_{1,n}$ and complete bipartite graph $K_{n,m}$.

Theorem 1 *Let $G \cong P_n$ be a path graph of order n, where $n \geq 4$. Then,*

$$\textbf{(a)}\ r^{agg}(P_n) = 0 \quad \textbf{(b)}\ r_{av}^{agg}(P_n) = \begin{cases} -2/n, & \text{if } n \text{ is odd;} \\ (2-n)/n, & \text{if } n \text{ is even.} \end{cases}$$

Proof. We know that $r(P_n)$ = -1 if n is even; $r(P_n) = 0$ if n is odd (see [23]), and let $\{v_1, v_2, \ldots, v_{n-1}, v_n\}$ be vertices of P_n. In here, we say that the vertices v_1 and v_n are minor vertices, remaining vertices are called major vertices. Clearly, number of minor and major vertices are 2 and n-2, respectively. While we are calculating the ARD and ALARD of the path graph P_n, we have two cases depending on n.

Case 1. Let n be even. We distinguish two sub cases depending on the vertices of P_n.

Subcase 1.1. If a minor vertex is agglomerated, then a path P_{n-1} is obtained. Due to n is even, we have $r(P_{n-1}) = 0$. So, we obtain $r_{v_1}^{agg}(G)$=0 and $r_{v_n}^{agg}(G)$=0.

Subcase 1.2. If a major vertex is agglomerated, then a path $P_{n\text{-}2}$ is obtained. Due to n is even, we have $r(P_{n\text{-}2})$ = -1. So, we obtain $r^{agg}_{v_k}(G)$=-1, where $k \in \{2,3,\ldots,n\text{-}1\}$.

Finally, we get $r^{agg}(P_n)$=0 by the definition of ARD and the Subcases 1.1 and 1.2.

Furthermore, we get

$$\begin{aligned} r^{agg}_{av}(G) &= \frac{1}{|V(G)|}\left(\sum_{v_k \in V(G)} r^{agg}_{v_k}(G)\right) \\ &= \frac{1}{n}\left(r^{agg}_{v_1}(G) + r^{agg}_{v_n}(G) + \sum_{k=2}^{n-1} r^{agg}_{v_k}(G)\right) \\ &= \frac{1}{n}\Big(2(0) + (-1(n-2))\Big) \\ &= \frac{2-n}{n}. \end{aligned}$$

Case 2. Let n be odd. We distinguish two sub cases depending on the vertices of P_n.

Subcase 2.1. If a minor vertex is agglomerated, then a path $P_{n\text{-}1}$ is obtained. Due to n is odd, we have $r(P_{n\text{-}1}) = -1$. So, we obtain $r^{agg}_{v_1}(G)$=-1 and $r^{agg}_{v_n}(G)$=-1.

Subcase 2.2. If a major vertex is agglomerated, then a path $P_{n\text{-}2}$ is obtained. Due to n is odd, we have $r(P_{n\text{-}2}) = 0$. So, we obtain $r^{agg}_{v_k}(G)$=0, where $k \in \{2,3,\ldots,n\text{-}1\}$.

Finally, we get $r^{agg}(P_n)$=0 by the definition of ARD and the Subcases 2.1 and 2.2.

Furthermore, we get

$$\begin{aligned} r^{agg}_{av}(G) &= \frac{1}{|V(G)|}\left(\sum_{v_k \in V(G)} r^{agg}_{v_k}(G)\right) \\ &= \frac{1}{n}\left(r^{agg}_{v_1}(G) + r^{agg}_{v_n}(G) + \sum_{k=2}^{n-1} r^{agg}_{v_k}(G)\right) \\ &= \frac{1}{n}\Big(2(-1) + (0(n-2))\Big) \\ &= \frac{-2}{n}. \end{aligned}$$

By the Cases 1 and 2, the proof is completed. □

Theorem 2 *Let $G \cong C_n$ be a cycle graph of order n, where $n \geq 5$. Then,*

$$r^{agg}(C_n) = r^{agg}_{av}(C_n) = \begin{cases} -2, & \text{if } n \text{ is odd;} \\ -1, & \text{if } n \text{ is even.} \end{cases}$$

Proof. We know that $r(C_n)$ = -1 if n is even; $r(C_n)$ = -2 if n is odd (see [23]), and let $\{v_1, v_2, \ldots, v_{n-1}, v_n\}$ be vertices of C_n. If a vertex is agglomerated in the graph C_n, then a cycle C_{n-2} is obtained. We have two cases depending on n.

***Case* 1.** Let n be even. Due to n is even, we have $r(C_{n-2})$ = -1. So, we obtain $r^{agg}_{v_k}(G)$=-1, where $k \in \{1,2,\ldots,n\}$.

***Case* 2.** Let n be odd. Due to n is odd, we have $r(C_{n-2})$ = -2. So, we obtain $r^{agg}_{v_k}(G)$=-2, where $k \in \{1,2,\ldots,n\}$.

Finally, we get

$$r^{agg}(C_n) = r^{agg}_{av}(C_n) = \begin{cases} -2, & \text{if } n \text{ is odd;} \\ -1, & \text{if } n \text{ is even.} \end{cases}$$

By the Cases 1 and 2, the proof is completed. □

Theorem 3 *Let $G \cong K_n$ be a complete graph of order n, where $n \geq 3$. Then,*

$$r^{agg}(K_n) = r^{agg}_{av}(K_n) = 0.$$

Proof. The rupture degree of K_n is defined as $r(K_n)$=$1-n$ [23]. Let $\{v_1, v_2, \ldots, v_{n-1}, v_n\}$ be vertices of K_n. If a vertex is agglomerated in the graph K_n, then the graph K_1 is obtained. Clearly, $r(K_1)$=0. So, we get $r^{agg}_{v_k}(G)$=0, where $k \in \{1,2,\ldots,n\}$. Thus, $r^{agg}(K_n)$=$r^{agg}_{av}(K_n)$=0 is obtained. □

Theorem 4 *Let $G \cong K_{1,n-1}$ be a star graph of order n, where $n \geq 4$. Then,*

$$\textbf{\textit{(a)}}\ r^{agg}(K_{1,n-1}) = n-4 \quad \textbf{\textit{(b)}}\ r^{agg}_{av}(K_{1,n-1}) = \frac{n^2-5n+4}{n}.$$

Proof. The rupture degree of $K_{1,n\text{-}1}$ is defined as $r(K_{1,n\text{-}1})$=n-3 [23]. Let $\{v_c, v_1, v_2, \ldots, v_{n\text{-}2}, v_{n\text{-}1}\}$ be vertices of $K_{1,n\text{-}1}$, where the vertex v_c is the center vertex of $K_{1,n\text{-}1}$. We distinguish two cases depending on the vertices of $K_{1,n\text{-}1}$.

***Case* 1.** If the center vertex v_c is agglomerated, then the complete graph K_1 is obtained. We know $r(K_1)$=0 [23]. So, we obtain $r^{agg}_{v_c}(G) = 0$.

***Case* 2.** If a vertex v_k, where $k \in \{2,3,\ldots,n\text{-}1\}$, is agglomerated, then a star graph $K_{1,n\text{-}1}$ is obtained. Thus, we get $r^{agg}_{v_k}(G) = n - 4$ for $k \in \{2,3,\ldots,n\text{-}1\}$.

Finally, we have $r^{agg}(K_{1,n\text{-}1})$=0 by the definition of ARD and the Cases 1 and 2.

Furthermore, we get

$$\begin{aligned}
r^{agg}_{av}(G) &= \frac{1}{|V(G)|}\Bigg(\sum_{v_k \in V(G)} r^{agg}_{v_k}(G)\Bigg) \\
&= \frac{1}{n}\Bigg(r^{agg}_{v_c}(G) + \sum_{k=1}^{n-1} r^{agg}_{v_k}(G)\Bigg) \\
&= \frac{1}{n}\Bigg((n-1)(n-4)\Bigg) \\
&= \frac{n^2 - 5n + 4}{n}.
\end{aligned}$$

By the Cases 1 and 2, the proof is completed. □

Theorem 5 *Let $G \cong W_{1,n}$ be a wheel graph of order $n+1$, where $n \geq 5$. Then,*

$$\textbf{\textit{(a)}}\ r^{agg}(W_{1,n}) = 0 \quad \textbf{\textit{(b)}}\ r^{agg}_{av}(W_{1,n}) = \begin{cases} -2n/(n+1), & \textit{if } n \textit{ is odd;} \\ -n/(n+1), & \textit{if } n \textit{ is even.} \end{cases}$$

Proof. The rupture degree of $W_{1,n}$ is defined as $r(W_{1,n})$ = -2 if n is even, and $r(W_{1,n})$ = -3 if n is odd (see [23]). Let $\{v_c, v_1, v_2, \ldots, v_{n\text{-}1}, v_n\}$ be vertices of $W_{1,n}$, where the vertex v_c is the center vertex of $W_{1,n}$. We distinguish two cases depending on the vertices of $W_{1,n}$.

Case 1. If the center vertex v_c is agglomerated, then the complete graph K_1 is obtained. We know $r(K_1)$=0 [23]. So, we obtain $r^{agg}_{v_c}(G)$=0.

Case 2. If a vertex v_k, where $k \in \{1,2,\ldots,n\}$, is agglomerated, then a join graph K_1+P_{n-3} is obtained. We distinguish two sub cases depending on the number of n.

Subcase 2.1. If n is even, then n-3 will be odd. Due to is $n-3$ odd, then we get r(K_1+P_{n-3})=-1 [23]. That is $r^{agg}_{v_k}(G)=-1$, where $k \in \{1,2,\ldots,n\}$.

Subcase 2.2. If n is odd, then $n-3$ will be even. Due to is $n-3$ even, then we get r(K_1+P_{n-3})=-2 [23]. That is $r^{agg}_{v_k}(G)=-2$, where $k \in \{1,2,\ldots,n\}$.

Finally, we get $r^{agg}(W_{1,n})=0$ by the definition of ARD and the Cases 1 and 2. Thus, we get

$$r^{agg}_{av}(G) = \frac{1}{|V(G)|}\left(\sum_{v_k\in V(G)} r^{agg}_{v_k}(G)\right) = \frac{1}{n}\left(r^{agg}_{v_c}(G) + \sum_{k=1}^{n-1} r^{agg}_{v_k}(G)\right)$$
$$= \frac{1}{n+1}(n)(-1) = \frac{-n}{n+1}, \quad \text{if } n \text{ is even.}$$

If n is odd, then we have

$$r^{agg}_{av}(G) = \frac{1}{|V(G)|}\left(\sum_{v_k\in V(G)} r^{agg}_{v_k}(G)\right) = \frac{1}{n}\left(r^{agg}_{v_c}(G) + \sum_{k=1}^{n-1} r^{agg}_{v_k}(G)\right)$$
$$= \frac{1}{n+1}(n)(-2) = \frac{-2n}{n+1}.$$

By the Cases 1 and 2, the proof is completed. □

Theorem 6 *Let $G \cong K_{n,m}$ be a complete bipartite graph of order $n+m$, where $1 < n \le m$. Then,*

$$\textbf{\textit{(a)}}\ r^{agg}(K_{n,m}) = m-3 \quad \textbf{\textit{(b)}}\ r^{agg}_{av}(K_{n,m}) = \frac{m^2+n^2-3m-3n}{n+m}.$$

Proof. The rupture degree of $K_{n,m}$ is defined as $r(K_{n,m})=1-m-n$ [23]. Let $\{v_1,v_2,\ldots,v_n,v'_1, v'_2,\ldots,v'_n\}$ be vertices of $K_{n,m}$. We distinguish two cases depending on the vertices of $K_{n,m}$.

Case 1. If a vertex v_k, where $k \in \{1,2,\ldots,n\}$, is agglomerated, then a star graph $K_{1,n-1}$ is obtained. We have $r(K_{1,n-1})=n-3$. Thus, we get $r^{agg}_{v_k}(G)=n-3$ for $k \in \{1,2,\ldots,n\}$.

Case 2. If a vertex v'_k, where $k \in \{1,2,\ldots,m\}$, is agglomerated, then a star graph $K_{1,m-1}$ is obtained. We have $r(K_{1,m-1})=m-3$. Thus, we get $r^{agg}_{v_k}(G)=$ $m-3$ for $k \in \{1,2,\ldots,m\}$.

We have $r(K_{n,m}){=}max\{n{-}3{,}m{-}3\}$. Due to $n{\leq}m$, we obtain $r(K_{n,m}){=}m{-}3$ by Cases 1 and 2.

Furthermore, we get

$$
\begin{aligned}
r^{agg}_{av}(G) &= \frac{1}{|V(G)|}\Bigg(\sum_{v_k\in V(G)} r^{agg}_{v_k}(G)\Bigg)\\
&= \frac{1}{n+m}\Bigg(\sum_{k=1}^{n} r^{agg}_{v_k}(G) + \sum_{k=1}^{m} r^{agg}_{v'_k}(G)\Bigg)\\
&= \frac{1}{n+m}\Bigg((n)(n-3)+(m)(m-3)\Bigg)\\
&= \frac{m^2+n^2-3m-3n}{n+m}.
\end{aligned}
$$

By the Cases 1 and 2, the proof is completed. □

5 A heuristic algorithm for computing the ARD and ALARD

In this section, firstly we give the pseudocode of heuristic algorithm for ARD and ALARD in Appendix A. This algorithm runs polynomial time to find the ARD and ALARD of an arbitrary graph G. We give an example how the proposed algorithm works on the following graph P_4.

Let P_4 be a path graph and the node array(labelled of nodes) is [0, 1, 2, 3] into the operation function. This graph showed in the Figure 6.

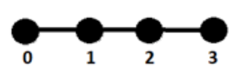

Figure 6: The graph P_4 whose vertices labelled by [0,1,2,3]

Let the vertex 1 and graph go to Agglomeration function in our Algorithm. The content of our neighbors array will be [[1], [0,2], [1,3], [2]]. For example, the content of the zeroth index is 1. So, the neighbor of node 0 will be 1. The content of aggCluster array is also [0,2]. Then we add the corresponding node and sort it from the largest number to the smallest number. The content would be [2, 1, 0]. Now we need to delete the row and column from the two-dimensional graph array. This process is also based on aggCluster. After deletion, our components array is created, then we have components = [[0], [1], [2,3]] as like the following Figure 7.

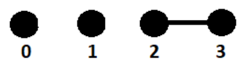

Figure 7: The graph P_4 after the first deletion.

Now the neighbors are deleted from the array of components. In summary, this is the function of keeping vertices that are not adjacent to the *agglomeration vertex* in the array of components. Now, we have components = [[3]].

Then, labeledComponents = components has been made. Furthermore, tag_number is 1 in the loop. Since labeledComponents is single content, the loop returns one and labeledComponents [0][0] = label_number. In other words, label 1 is given to the neighbor of the merged nodes(0) and the newGraf becomes P_2 as like the following Figure 8.

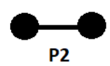

Figure 8: The graph P_2

The created newGraph is sent to the Rupture function which is proposed in [15] and then returns -1. The Agglomeration function also returns this Integer value. The Integer value from the operation function is added to the ruptures sequence. This event is made for all nodes and ruptures = [0, -1, -1, 0] is obtained. As a result, the maximum value will be ARD and its arithmetic average will be ALARD, that is $r^{agg}(P_4)$=0 and $r^{agg}_{av}(P_4) = \frac{-1}{2}$ are obtained.

5.1 Computational tests

In this section, the datasets of the references [11] and [15] have been used to perform our proposed algorithm. In the following tables, |V| is the number of vertices; ARD is the Heuristic result of Agglomeration Rupture Degree; ARDopt displays the Brute Force result of Agglomeration Rupture Degree; ALARD is the Heuristic result of the Average Lower Agglomeration Rupture Degree; ALARDopt, Brute Force result of Average Lower Agglomeration Rupture Degree; t(s) represents the running time in seconds. *Error* is the absolute gap, which is the magnitude of the difference between the values of ARD, ALARD and the results of ARD, ALARD obtained by the proposed algorithm. Furthermore, 25%, 50%, 75% and 100% indicate the edge density of the graph G.

The proposed algorithm is implemented in JAVA and tested on i5-7600U machine with 2.9 GHz processor and 8 GB RAM. Clearly, we can see that the results of the actual ARD and ALARD is almost similar the result of ARD and ALARD obtained by proposed algorithm in Tables 2 and 3. We also tested our algorithm for the medium size graphs whose numbers of vertices more than *100*. Since we don't know actual values of ARD and ALARD, we give only heuristic result of ARD and ALARD with CPU time in the Tables 4 and 5. As a result, we have tested the algorithm on some graph families which are used in Theorems 1–6. Then, same values of ARD and ALARD are obtained as given Theorems 1–6.

	25%			50%			75%			95%		
|V|	ARD	ARDopt	Error	ARD	ARDopt	Error	ARD	ARDopt	Error	ARD	ARDopt	Error
10	0	0	0	0	0	0	1	1	0	0	0	0
11	1	1	0	1	1	0	0	0	0	0	0	0
12	0	0	0	0	0	0	-1	-1	0	0	0	0
13	0	0	0	1	1	0	1	1	0	0	0	0
14	-1	-1	0	-1	-1	0	0	0	0	0	0	0
15	2	2	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	-1	-1	0	0	0	0
17	2	2	0	0	0	0	0	0	0	0	0	0
18	-1	-1	0	-1	-1	0	0	0	0	0	0	0
19	-1	-1	0	-1	-1	0	-1	-1	0	0	0	0
20	-1	-1	0	0	0	0	-1	-1	0	0	0	0
21	-1	-1	0	-2	-2	0	0	0	0	0	0	0
22	-1	-1	0	-1	-1	0	-1	-1	0	0	0	0
23	1	1	0	-2	-2	0	-1	-1	0	0	0	0
24	-3	-3	0	-2	-2	0	0	0	0	0	0	0
25	-3	-3	0	-3	-3	0	0	0	0	0	0	0
26	-3	-3	0	-3	-3	0	1	1	0	0	0	0
27	-4	-4	0	-2	-2	0	-1	-1	0	0	0	0

Table 2: Computational experiments on small-sized graphs for ARD.

	25%			50%			75%			95%		
|V|	ALARD	ALARDopt	Error	ALARD	ALARDopt	Error	ALARD	ALARDopt	Error	ALARD	ALARDopt	Error
10	-11/10	-11/10	0	-14/10	14/10	0	-6/10	-6/10	0	-4/10	-4/10	0
11	-7/11	-6/11	-1/11	-14/11	-14/11	0	-12/11	-12/11	0	-6/11	-6/11	0
12	-9/12	-9/12	0	-28/12	-28/12	0	-32/12	-32/12	0	-6/12	-6/12	0
13	-19/13	-19/13	0	-19/13	-19/13	0	-18/13	-18/13	0	-8/13	-8/13	0
14	-32/14	-32/14	0	-34/14	-34/14	0	-26/14	-26/14	0	-10/14	-10/14	0
15	-15/15	-14/15	-1/15	-37/15	-37/15	0	-30/15	-30/15	0	-10/15	-10/15	0
16	-32/16	-32/16	0	-48/16	-48/16	0	-42/16	-42/16	0	-12/16	-12/16	0
17	-23/17	-23/17	0	-43/17	-43/17	0	-40/17	-40/17	0	-14/17	-14/17	0
18	-43/18	-43/18	0	-58/18	-58/18	0	-40/18	-40/18	0	-16/18	-16/18	0
19	-53/19	-53/19	0	-56/19	-56/19	0	-50/19	-50/19	0	-12/19	-12/18	0
20	-63/20	-63/20	0	-69/20	-69/20	0	-53/20	-53/20	0	-20/20	-20/20	0
21	-90/21	-87/21	-3/21	-99/21	-99/21	0	-70/21	-70/21	0	-14/21	-14/21	0
22	-86/22	-86/22	0	-105/22	-104/22	-1/22	-59/22	-59/22	0	-24/22	-24/22	0
23	-95/23	-91/23	-4/23	-109/23	-109/23	0	-62/23	-62/23	0	-26/23	-26/23	0
24	-133/24	-133/24	0	-132/24	-132/24	0	-82/24	-82/24	0	-28/24	-28/24	0
25	-143/25	-142/25	-1/25	-144/25	-144/25	0	-95/25	-95/25	0	-30/25	-30/25	0
26	-148/26	-148/26	0	-168/26	-166/26	-2/26	-96/26	-96/26	0	-32/26	-32/26	0
27	-157/27	-157/27	0	-167/27	-167/27	0	-105/27	-105/27	0	-36/27	-36/27	0

Table 3: Computational experiments on small-sized graphs for ALARD

	20%			30%			40%			50%		
|V|	ARD	ALARD	t(s)	ARD	ALARD	t(s)	ARD	ALARD	t(s)	ARD	ALARD	t(s)
100	-33	-4400/100	3.8	-36	-4630/100	3.6	-31	-4203/100	2.9	-26	-3712/100	3.2
110	-42	-5809/110	4.7	-42	-5788/110	4.3	-37	-5293/110	3.1	-33	-4580/110	3.0
120	-48	-7075/120	5.5	-48	-7022/120	5.2	-42	-6346/120	4.0	-35	-5565/120	3.4
130	-51	-8399/130	7.3	-50	-8379/130	6.6	-43	-7652/130	5.0	-33	-6610/130	7.7
140	-61	-9972/140	8.8	-59	-9966/140	8.4	-48	-8990/140	8.8	-39	-7793/140	5.9
150	-72	-12293/150	13.0	-65	-11686/150	10.1	-60	-10548/150	9.7	-46	-9003/150	6.6
160	-73	-13783/160	15.4	-69	-13504/160	12.5	-60	-12125/160	10.9	-47	-10418/160	6.7
170	-82	-16071/170	19.7	-71	-15485/170	18.3	-66	-13863/170	11.2	-53	-11775/170	7.8
180	-79	-18034/180	22.6	-72	-17357/180	18.0	-68	-15719/180	12.9	-57	-13406/180	8.9
190	-95	-20661/190	28.8	-88	-19728/190	23.8	-76	-17706/190	17.6	-57	-15129/190	11.8
200	-101	-23233/200	35.3	-96	-22132/200	31.2	-80	-19638/200	20.2	-66	-16858/200	13.0

Table 4: Computational experiments on medium-sized graphs.

	60%			70%			80%			90%		
|V|	ARD	ALARD	t(s)	ARD	ALARD	t(s)	ARD	ALARD	t(s)	ARD	ALARD	t(s)
100	-19	-3033/100	2.1	-11	-4630/100	1.8	-8	-1536/100	2.7	-2	-786/100	1.7
110	-22	-3733/110	3.0	-16	-2812/110	2.2	-10	-1890/110	1.6	-4	-936/110	1.7
120	-26	-4553/120	2.6	-18	-3425/120	2.5	-9	-2279/120	1.9	-3	-1133/120	1.3
130	-27	-5380/130	3.4	-21	-4042/130	2.9	-11	-2692/130	1.9	-3	-1319/130	1.9
140	-29	-6356/140	3.8	-20	-4764/140	2.8	-10	-3148/140	2.6	-3	-1577/140	1.6
150	-36	-7397/150	4.8	-23	-5547/150	3.1	-14	-3657/150	2.3	-4	-1808/150	1.6
160	-36	-8423/160	5.7	-26	-6390/160	3.8	-16	-4207/160	1.7	-5	-2104/160	2.7
170	-40	-9559/170	4.9	-26	-7228/170	3.9	-15	-4789/170	2.2	-7	-2354/170	2.4
180	-42	-10818/180	6.6	-30	-8204/180	4.2	-16	-5407/180	2.7	-4	-2677/180	2.3
190	-44	-12186/190	7.0	-34	-9202/190	4.7	-18	-6083/190	3.7	-7	-2965/190	2.7
200	-54	-13659/200	8.9	-36	-10267/200	5.9	-20	-6819/200	4.5	-7	-3318/200	4.0

Table 5: Computational experiments on medium-sized graphs.

6 Conclusion

In this paper, we considered agglomeration-based rupture degree in graphs. We define and investigate the agglomeration rupture degree $r^{agg}(G)$ and the average lower agglomeration rupture degree $r_{av}^{agg}(G)$, then these values have been computed for well-known families of graphs. Finally, we proposed a polynomial time heuristic algorithm to find the set of the lower agglomeration rupture degree $r_{v_k}^{agg}(G)$ for every vertex and also the values of $r^{agg}(G)$ and $r_{av}^{agg}(G)$ for any graph G. Then, we present the results of computational experiments on graphs with up to *200* vertices. The results show that the proposed heuristic algorithm efficiently computes the values of $r^{agg}(G)$ and $r_{av}^{agg}(G)$ of a given graph G. Developing of several heuristics for computing the other agglomeration-based graph parameters of graphs are the subjects of future work.

References

[1] E. Aslan, Weak-rupture degree of graphs, *Int. Journal of Foundations of Computer Science*. **27**, 6 (2016) 725–738. doi:10.1142/S0129054116500258 ⇒126

[2] E. Aslan, Edge-rupture degree of a graph, *Dynamics of Continuous, Discrete and Impulsive Systems Series B: Applications and Algorithms*. **22**, 2 (2015) 155–161. ⇒126

[3] A. Aytac, H. Aksu, Some results for the rupture degree, *International Journal of Foundations of Computer Science*. **24**, 8 (2013) 1329–1338. doi:10.1142/S0129054113500366 ⇒126

[4] A. Aytac, H. Aksu, The rupture degree of some graphs, *Mathematica Balkanica*. **24**, 1-2(2010) 85–101. ⇒126

[5] A. Aytac, Z.N. Odabas, Computing the rupture degree in composite graphs, *International Journal of Foundations of Computer Science*. **21**, 3 (2010) 311–319. doi:10.1142/S012905411000726X ⇒126

[6] A. Aytac, T. Turaci, Vertex vulnerability parameter of gear graphs, *International Journal of Foundations of Computer Science*. **22**, 5 (2011) 1187–1195. ⇒125

[7] V. Aytac, T. Turaci, Relationships between vertex attack tolerance and other vulnerability parameters, *RAIRO - Theoretical Informatics and Applications*. **51**, 1 (2017) 17–27. doi:10.1051/ita/2017005 ⇒125

[8] G. Bacak-Turan, E. Oz, Neighbor rupture degree of transformation graphs G^{xy-}, *International Journal of Foundations of Computer Science*. **28**, 4 (2017) 335–355. doi:10.1142/S0129054117500216 ⇒126

[9] K.S. Bagga, L.W. Beineke, W.D. Goddard, M.J. Lipman, R.E. Pippert, A survey of integrity, *Discrete Applied Mathematics*. **37-38** (1992) 13–28. doi:10.1016/0166-218X(92)90122-Q ⇒125

[10] Z.N. Berberler, H.İ. Yildirim, T. İltüzer, İ Tunç, Agglomeration-Based Node Importance Analysis in Wheel-Type Networks, *International Journal of Foundations of Computer Science*. **32**, 3 (2021) 26–288. doi:10.1142/S0129054121500210 ⇒127

[11] M. E. Berberler, Z. N. Berberler, Measuring the vulnerability in networks: a heuristic approach, *Ars Combinatoria*. **135** (2017) 3–15. ⇒126, 136

[12] V. Chvatal, Tough graphs and Hamiltonian circuits, *Discrete Mathematics*. **5**, 3 (1973) 215–228. doi:10.1016/0012-365X(73)90138-6 ⇒125

[13] M. Cozzens, D. Moazzami, S. Stueckle, The tenacity of a graph, *17th Int. Conf. Theory and Applications of Graphs*, Wiley, New York, 1995, pp. 1111–1122. ⇒125

[14] R. Durgut, H. Kutucu and T. Turacı, Global distribution center number of some graphs and an algorithm, *RAIRO-Operations Research*. **53**, 4 (2019) 1217–1227. doi:10.1051/ro/2018119 ⇒125

[15] R. Durgut, T. Turacı, H. Kutucu, A heuristic algorithm to find rupture degree in graphs, *Turk. J. Elec. Eng. & Comp Sci*. **27** (2019) 3433 – 3441. doi:10.3906/elk-1903-29 ⇒126, 136

[16] H. Frank, I.T. Frisch, Analysis and design of survivable networks, *IEEE Transactions on Communications Technology*, **18**, 5 (1970) 501–519. doi:10.1109/TCOM.1970.1090419 ⇒125

[17] M.A. Henning, Trees with Equal Average Domination and Independent Domination Numbers, *Ars Combinatoria*. **71** (2004) 305–318. ⇒125

[18] H. A. Jung, On maximal circuits infinite graphs, *Annals of Discrete Mathematics*. **3** (1978) 129–144. doi:10.1016/S0167-5060(08)70503-X ⇒125

[19] A. A. Kunt, Z. N. Berberler, Efficient identification of node importance based on agglomeration in cycle-related networks, *International Journal of Foundations of Computer Science*. **31**, 7 (2020) 969–978. doi:doi:10.1142/S0129054120500379 ⇒127

[20] A. Kirlangic, G. Bacak-Turan, On the rupture degree of a graph, *Neural Network World*. **22**, 1 (2012) 39–51. doi:10.14311/NNW.2012.22.003 ⇒126

[21] O.K. Kurkçu, E. Aslan, A comparison between edge neighbor rupture degree and edge scattering number in graphs, *International Journal of Foundations of Computer Science* **29**, 7 (2018) 1119–1142. doi:10.1142/S0129054118500247 ⇒126

[22] I. Mishkovski, M. Biey, L. Kocarev, Vulnerability of complex networks, *Communications in Nonlinear Science and Numerical Simulation*. **16**, 1 (2011) 341–349. doi:10.1016/j.cnsns.2010.03.018 ⇒125

[23] Y. Li, S. Zhang, X. Li, Rupture degree of graphs, *International Journal of Computer Mathematics*. **82**, 7 (2005) 793–803. doi:10.1080/00207160412331336062 ⇒125, 130, 132, 133, 134

[24] Y. Li, The rupture degree of trees. *International Journal of Computer Mathematics*. **85**, 11 (2008) 1629–1635. doi:10.1080/00207160701553367 ⇒126

[25] F. Li, Isolated rupture degree of trees and gear graphs, *Neural Network World*. **25**, 3 (2015) 287–300. doi:10.14311/NNW.2015.25.015 ⇒126

[26] F. Li, X. Li, Computing the rupture degrees of graphs. *7th International Symposium on Parallel Architectures, Algorithms and Networks*. 2004, pp. 368–373, Hong Kong, China. ⇒126

[27] Z.N. Odabas, A. Aytac, Rupture degree and middle graphs, *Comptes Rendus de Lacademie Bulgare des Sciences*. **65**, 3 (2010) 315–322. ⇒126

[28] Y. J. Tan, W. Jun, H. Z. Deng, Evaluation method for node importance based on node contraction in complex networks, Systems *Engineering-Theory & Practice*. **11.11** (2006) 79–83. ⇒127

[29] T. Turacı, A. Aytac, Combining the Concepts of Residual and Domination in Graphs, *Fundamenta Informaticae*. **166**, 4 (2019) 379–392. doi:10.3233/FI-2019-1806 ⇒125

[30] T. Turacı, On combining the methods of link residual and domination in networks, *Fundamenta Informaticae*. **174**, 1 (2020) 43–59. doi:10.3233/FI-2020-1930 ⇒125

[31] T. Turaci, E. Aslan, The average lower reinforcement number a graph, *RAIRO-Theor. Inf. Appl*. **50**, 2 (2016) 135–144. doi:doi:10.1051/ita/2016015 ⇒125

[32] T. Turaci, On the average lower bondage number a graph, *RAIRO-Operations Research*. **50**, 4-5 (2016) 1003–1012. doi:10.1051/ro/2015062 ⇒125

[33] B.D. West, *Introduction to Graph Theory*. 2nd ed. Urbana, NJ, USA: Prentice Hall, 2001. ⇒125

APPENDIX

Void_function **Process**(graph_parameter mainGraph, node_array_parameter arrayNode) {
ruptures[] ← Ø
for i ← 0 to arrayNode's length {
    ruptures[i] ← Agglomeration(mainGraph, arrayNode[i]) }
Integer ARD ← largest value of array ruptures
Double ALARD ← sum of all ruptures array elements / arrayNode's length
}. # end function

Integer_function **Agglomeration**(graph_parameter mainGraph, node_parameter Node) {
Graph ← mainGraph # Cloning the master graph to avoid corruptions.
neighbors ← neighboring nodes corresponding to each index.
aggCluster ← neighbors[Node] # Finding the neighbors of the node.
for i ← 0 to aggCluster's length { # aggCluster nodes find their neighbors.
    temp[i] ← neighbors[aggCluster[i]]
    for j ← 0 to temp's length {
        if temp[i] isn't equal to Node {

```
        add temp[j] to neighbors } } }

add Node to aggCluster
sort aggCluster by contents from largest to smallest
for i ← 0 to aggCluster's length { # Reset row and column.
    for j ← 0 to Graph's length {
        Graph[j][aggCluster[i]] ← 0
        Graph[aggCluster[i]][j] ← 0 } }
components[][] ← newly formed graph sets # add new graphs.
for i ← 0 to length of components { # deleting the neighborhood from the
components array.
    for j ← 0 to length of components[i] {
    if aggCluster contains components[i][j] {
        components[i][0] ← Ø } } }
for i ← 0 to length of components {
    if the length of components[i] is 0 {
        remove the i. variable from components } }

# Creating tagged component ↓
labeledComponents[][] ← Ø
labeledComponents ← components

# new tags ↓
Integer tag_number ← 0
for i ← 0 to length of labeledComponents {
    for j ← 0 to length of labeledComponents[i] {
        tag_number ← tag_number + 1
        labeledComponents[i][j] ← tag_number } }

# remove if empty ↓
for i ← 0 to length of labeledComponents {
    if the length of labeledComponents[i] is 0 {
        remove the i. variable from labeledComponents
        remove the i. variable from components } }

# create new graph ↓
Integer value ← Graph's length − aggCluster's length + 1
newGraph[value][value] ← Ø # newGraph is the matrix with value*value
length.
```

```
for i ← 0 to length of labeledComponents {
    for j ← 0 to length of labeledComponents[i] {
        if neighbors contain components[i][j] {
            newGraph[0][labeledComponents[I][j]] ← 1
            newGraph[labeledComponents[I][j]][0] ← 1 } } }
for i ← 0 to length of components {
    for j ← 0 to length of components[i] {
        for k ← j to length of components[i] {
            Integer a ← Graph[components[i][j]][components[i][k]]
            newGraph[labeledComponents[i][j]][labeledComponents[i][k]]← a
            Integer b ← Graph[components[i][k]][components[i][j]]
            newGraph[labeledComponents[i][k]][labeledComponents[i][j]] ← b
} } }

return function Rupture(newGraph) # Branched into the heuristic rupture algorithm.
}. # end function
```

Received: April 27, 2023 • Revised: July 16, 2023