IOSR Journal of Mathematics (IOSR-JM)
e-ISSN: 2278-5728, p-ISSN: 2319-765X. Volume 18, Issue 3 Ser. II (May. – June. 2022), PP 06-12
www.iosrjournals.org

Polynomial time algorithms of Critical Edge Detection Problem on a tree

Syed Md Omar Faruk[1,*], KamrunNahar Jabin[1]
[1]*Department of Mathematics, Shahjalal University of Science and Technology*

Abstract
In this paper we look at the critical edge detection problem (CEDP), which is the task of finding a set of K edges in a graph G whose removal reduces the number of connections between pairs of nodes in the residual graph. In particular, we mainly studied the case in which the given graph is a tree. We investigate the critical edge detection problem over trees in different situations of edge weights. We provide polynomial algorithms for the problems when all connections between pairs of nodes have unit cost.
Keywords*: Critical edge detection, Pairwise connectivity, Polynomial time algorithm*

Date of Submission: 03-06-2022 Date of Acceptance: 17-06-2022

---

1 Introduction

The critical edge detection problem (CEDP) is the optimization problem that consists in finding the set of edges from a graph in order to obtain a disconnected graph with some specific properties. In general, one of the oldest problems in graph theory involves removing edges from a graph. Myung and Kim [18] address the problem of removing a small number of edges from an undirected graph in order to reduce the weighted number of connections guaranteed in the residual graph. A variety of variants have been described and investigated in the literature for problems involving the deletion of edges, such as the graph partitioning problem [5, 12, 20], the minimum k-cut problem [14, 16], the multicut problem [8, 13, 15], the multiway cut problem [6, 7, 10], the multi-multiway cut problem [3], etc.

The choice of the connectivity measure is of course a central element in a critical edge detection problem. Indeed, various different connection measures have been presented in the literature, such as the total number of pairwise connections [2, 11], the weight of the connections between the node pairs [2, 11], the size of the largest connected components (to be minimized) [19, 21], the total number of connected components (which should be maximized in order to fragment the graph) [1, 21], average shortest path value [9], distance-based connectivity metrics [22], the diameter [1], and the graph information entropy [4, 23].

In this paper we consider the pairwise connectivity between nodes, formalized in [2]. The pairwise connectivity of a graph $G = (V, E)$ is defined as the number of pairs of nodes belonging to the same connected component. For each pair $(u, v) \in V \times V$, the pairwise connectivity c_{uv} is quantified as follows:

$$c_{uv} = \begin{cases} 1 & \text{if } u \text{ and } v \text{ are connected,} \\ 0 & \text{otherwise.} \end{cases}$$

According to the pairwise connectivity measure mentioned above, the Critical Edge Detection Problem (CEDP) is formally stated as follows:

Given an undirected graph $G(V, E)$ with $|V| = n$ nodes and an integer $K > 0$ which is the maximunm number of edges that can be removed. The Critical Edge Detection Problem calls for removing from G a subset of edges $S \subseteq E$, where $|S| \leq K$, in order to minimize pairwise connectivity among the nodes in the subgraph $G(E \backslash S)$. Mathematically, the objective of the CEDP is to determine

$$\text{(CEDP)} \qquad S = \underset{S \subseteq E}{\operatorname{argmin}} \sum_{u,v \in (E \setminus S)} c_{uv}(G(E \setminus S) : |S| \leq K$$

where

$$c_{uv} = \begin{cases} 1 & \text{if } u, v \text{ are in the same component of } G(E \setminus S) \\ 0 & \text{otherwise.} \end{cases}$$

Formally, CEDP's goal is to discover S whose objective function yields the smallest value.

We present a dynamic programming approach for the case with unit costs and unit edge weights in Section 2 whose complexity is polynomial. In Section 3 we present the case with unit costs and general edge weights. We note that structural parts of our algorithmic framework are similar to those provided in [11, 17, 21].

Throughout the paper in our all dynamic programs, we denote by T_a the subtree of the given tree $T(V, E)$ rooted at node $a \in V$. If a is not a leaf of T, we assume that an arbitrary order of its children is specified. If a has s children $a_1, \dots, a_s$, for every $i \in \{1, \dots, s\}$ we define $T_{a_{i,s}}$ as the subtree of T induced by $\{a\} \cup V(T_{a_i}) \cup \dots \cup V(T_{a_s})$, where we denote by $V(H)$ the set of vertices of H for any given subtree H of T. Figure 1 shows an example of a tree T rooted at node a where subtree T_{a_2} contains nodes of the set $\{a_2, 3,4,5,6,7\}$, while subtree $T_{a_{3,4}}$ contains nodes of the set $\{a, a_3, 8,9,10,11,12, a_4, 13,14,15\}$. In our dynamic programming approaches, all recursions are based on traversing the tree in postorder (that is, from the leaves to the root) and from the right part of each tree level to the left part.

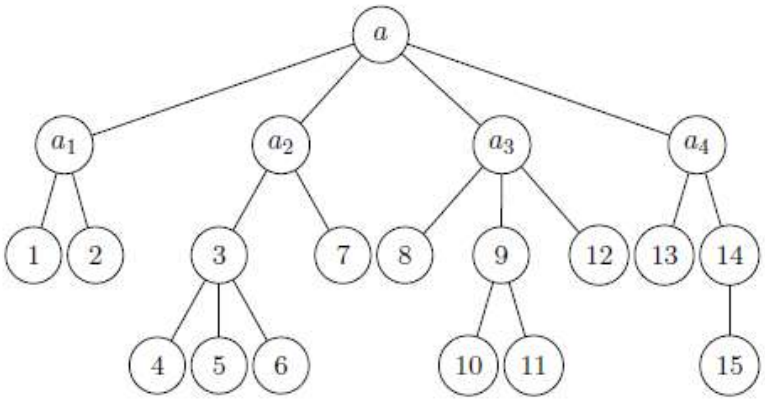

Figure 1: T_a is an example of a subtree in which node a has four children (i.e. $s = 4$).

In the next two sections, we will look at unit weights and general nonnegative weights for the edges separately, since even though the latter subsumes the former, unitary weights have a faster running time.

2 The unit cost, unit weight case on trees

In this part, we show how to solve CEDP on trees when $c_{uv} = 1$ for all u, v and $w_v = 1$ for all $v \in E$. In this scenario, the goal is to reduce the number of paths left in a tree $T(V, E)$ after removing at most K edges.

To derive a dynamic programming algorithm, we will calculate recursively the following values:

• $F_a(k, m)$ = minimum number of connections that still exists in the subtree T_a when k edges are removed from T_a and m nodes (including a itself) of T_a are still connected to the root a.

• $G_{a_i}(k,m)$ = minimum number of connections that still exists in the subtree $T_{a_i,s} = a + T_{a_i} + \cdots + T_{a_s}$ when k edges are removed from $T_{a_i,s}$ and m nodes of the subtree are still connected to a.

We remark that for both functions, the number of nodes connected to the root will never be 0 (i.e. $m > 0$) because we never remove the root as the root is a node and we can not remove a node in the edge deletion problem. Furthermore, whenever the conditions in one of the above definitions cannot be satisfied, we set the value of the function to infinity.

The values of F_a and G_{a_i} are calculated in this order:

• we determine F_a for every leaf a;

• for a non-leaf node a, assuming that the $F_{a'}$ and$G_{a'_i}$ have been already found for all $a' \in V(T_a)$, we calculate $G_{a_s}, G_{a_{s-1}}, \dots, G_{a_1}$, and then F_a.

At the end of the recursion, we can return the optimal value of the problem, assuming that the tree T is rooted at node 1, which is

$$OPT = \min\{F_1(K,m): m = 1, \dots, n\}.$$

As usual in dynamic programming, an optimal solution can be reconstructed by backtracking.

We now provide the explicit formulas and then a justification for each of them. For a non-leaf node $a \in V$, we have

$$F_a(k,m) = G_{a_1}(k,m), \quad (1)$$

while for every leaf a the formula is

$$F_a(k,m) = \begin{cases} 0 & \text{if } k = 0, m = 1, \\ \infty & \text{otherwise.} \end{cases} \quad (2)$$

For any non-leaf node $a \in V$ and $i < s$ (non-rightmost subtrees) we use the formula

$$G_{a_i}(k,m) =$$

$$\begin{cases} \min\{F_{a_i}(0,p) + G_{a_{i+1}}(0,m-p) + p(m-p): p = 0, \dots, m\} & \text{if } k = 0, \\ \min\{\min\{F_{a_i}(q,p) + G_{a_{i+1}}(k-1-q,m): q = 0, \dots, k-1, \\ \quad p = 0, \dots, V(T_{a_i})\}, \min\{F_{a_i}(q,p) + G_{a_{i+1}}(k-q,m-p) + p(m-p): \\ \quad q = 0, \dots, k, p = 0, \dots, m\}\} & \text{if } k > 0, \end{cases} \quad (3)$$

The initial conditions on each rightmost subtree T_{a_s} are calculated as follows:

$$G_{a_s}(k,m) = \begin{cases} \infty & \text{if } k = 0, m = 1, \\ \min\{F_{a_s}(k-1,p): p = 0, \dots, |V(T_{a_s})|\} & \text{if } k > 0, m = 1, \\ F_{a_s}(k,m-1) + (m-1) & \text{if } k \geq 0, m > 1. \end{cases} \quad (4)$$

We now give a justification for the above formulas. Equation (1) follows because $T_a = T_{a_{1,s}}$ for any non-leaf node $a \in V$.

Equation (2) handles the case of a one-node tree. Since $a \in V$ is a leaf, it is not possible to remove any edge ($k = 0$) and only a is connected to itself ($m = 1$) and the number of paths surviving in T_a is 0.

Recursion (3) can be interpreted as follows:

The case $k = 0$ means that we are not removing any edge from $T_{a_{i,s}} = T_{a_i} + T_{a_{i+1,s}}$. Since we have to keep everything, we are not allowed to remove anything from the subtrees T_{a_i} and $T_{a_{i+1,s}}$. If a_i is connected to p nodes of T_{a_i}, then a is connected to $m - p$ nodes in $T_{a_{i+1,s}}$ and the paths passing through a are exactly $p(m - p)$. Hence by definition of F and G the minimum number of paths that survive in $T_{a_{i,s}}$ when we are not removing anything will be $G_{a_i}(0, m) = \min_p\{F_{a_i}(0, p) + G_{a_{i+1}}(0, m - p) + p(m - p)\}$.

The case $k > 0$ means that we have to remove at least one edge. When the value of $G_{a_i}(k, m)$ is achieved from the expression $F_{a_i}(q, p) + G_{a_{i+1}}(k - 1 - q, m)$, we remove the edge e (which connects a to a_i). Expression $F_{a_i}(q, p)$ gives the minimum number of paths that survive in T_{a_i} when q edges are removed from T_{a_i} and p nodes of T_{a_i} are still connected to a_i. Since q edges have been removed from T_{a_i}, exactly $k - 1 - q$ edges must be removed from $T_{a_{i+1,s}}$. The minimum number of paths that survive in $T_{a_{i+1,s}}$ when $k - 1 - q$ edges are removed from $T_{a_{i+1,s}}$ is given by $G_{a_{i+1}}(k - 1 - q, m)$. Thus the expression $F_{a_i}(q, p) + G_{a_{i+1}}(k - 1 - q, m)$ gives the minimum number of paths that survive in $T_{a_{i,s}}$ when q edges are removed from T_{a_i} (and the other $k - 1 - q$ edges are removed from $T_{a_{i+1,s}}$) and p nodes of T_{a_i} are still connected to a_i. By taking the minimum over $q = 0, \dots, k - 1$ and $p = 0, \dots, |V(T_{a_i})|$, we find the value of $G_{a_i}(k, m)$.

When the value of $G_{a_i}(k, m)$ is achieved from the expression $F_{a_i}(q, p) + G_{a_{i+1}}(k - q, m - p) + p(m - p)$, we are not removing the edge e. As above, expression $F_{a_i}(q, p)$ gives the minimum number of paths that survive in T_{a_i} when q edges are removed from T_{a_i} and p nodes of T_{a_i} are still connected to a_i. Since q edges have been removed from T_{a_i}, exactly $k - q$ edges must be removed from $T_{a_{i+1,s}}$ and since p nodes of T_{a_i} are still connected to a_i and thus to a, exactly $m - p$ nodes of $T_{a_{i+1,s}}$ must remain connected to a. The minimum number of paths that survive in $T_{a_{i+1,s}}$ when $k - q$ edges are removed from $T_{a_{i+1,s}}$ and $m - p$ nodes of $T_{a_{i+1,s}}$ are still connected to a is given by $G_{a_{i+1}}(k - q, m - p)$. Thus the expression $F_{a_i}(q, p) + G_{a_{i+1}}(k - q, m - p)$ gives the minimum number of paths that survive in T_{a_i} or $T_{a_{i+1,s}}$ when q edges are removed from T_{a_i} (and the other $k - q$ edges are removed from $T_{a_{i+1,s}}$) and p nodes of T_{a_i} are still connected to a_i, while $m - p$ nodes of $T_{a_{i+1,s}}$ are still connected to a. Now we have to add the paths connecting nodes of T_{a_i} to nodes of $T_{a_{i+1,s}}$, i.e. $p(m - p)$ paths. This gives expression $F_{a_i}(q, p) + G_{a_{i+1}}(k - q, m - p) + p(m - p)$ of recursion (3). By taking the minimum over $q = 0, \dots, k$ and $p = 0, \dots, m$, we find the value of $G_{a_i}(k, m)$.

More specifically, if $k = 0$, we have only one choice so that we have to keep the edge e (which connects a to a_i) because we are not removing any edge. While in the case $k > 0$, we have two possibilities that both are possible i.e., we can choose if we want to remove the edge e or we want to keep it.

For a justification of (4), recall that $T_{a_{s,s}} = a + T_{a_s}$. If $m = 1$ and $k > 0$, then we have to remove the edge between a and a_s and the other $k - 1$ edges we have to be removed from the subtree T_{a_s} and the number of connections that survive are those in the subtree T_{a_s}. On the other hand if $m > 1$, then we can not remove the edge a to a_s and in this time we have to remove all the k edges inside the subtree T_{a_s}. Since m nodes are connected to a including a itself, in the subtree we will find the other $m - 1$ nodes connected to a_s. Then we have to add all the connections of a to the nodes that are connected to a_s in the subtree.

We obtain the following result.

Proposition 1 *CEDP on a tree with unit connection costs and unit edge weights can be solved by recursion (1)–(4) in $\mathcal{O}(K^2n^3)$ time.*

Proof. For each node $a \in V$ there are at most $K + 1 = \mathcal{O}(K)$ values for k and $n + 1 = \mathcal{O}(n)$ values for m; this gives $\mathcal{O}(Kn^2)$ values of F and G to compute. The heaviest computation is that of equation (3) that

requires at most $O(Kn)$ steps. Hence in the worst case a number of operations bounded by $O(K^2n^3)$ are required.

3 The case with unit costs and arbitrary edge weights

Let $w_e \geq 0$ be arbitrary weights assigned to the edges $e \in E$. The CEDP problem in this case amounts to finding a subset S of edges with total weight $\sum_{e\in S} w_e$ not exceeding a given budget W such that the number of surviving paths after having removed the edge set S is minimized.

A dynamic programming algorithm, constructed in the same spirit as the one described in the preceding section, can be used to solve this case. The recursion uses two parameters, m and k, which represent the number of nodes connected to the root of a subtree and the number of paths that survive within that subtree, respectively.

The following functions are defined using the subtree notation described in the preceding section.

• $F_a(k,m)$ is the minimum total weight of the edges to be removed from the subtree T_a in order to have node a connected to exactly m nodes (including a itself) and k paths surviving in T_a.

• $G_{a_i}(k,m)$ is the minimum total weight of the edges to be removed from the subtree $T_{a_{i,s}} = a + T_{a_i} + T_{a_{i+1}} + \cdots + T_{a_s}$ in order to have a connected to m nodes of $T_{a_{i,s}}$ and k paths surviving in $T_{a_{i,s}}$.

We compute the values for F and G recursively for all $a \in V$, $k = 0, \dots, n(n-1)/2$, $m = 1, \dots, n$, as follows. Assume $F_a(k,m) = \infty$, $G_a(k,m) = \infty$ if $k < 0$ or $m < 0$.

$$F_a(k,m) = G_{a_1}(k,m) \text{for all non} - \text{leaf nodes a} \in \text{V,} \quad (5)$$

$$G_{a_i}(k,m) = \min\{w_e + \min\{F_{a_i}(q,p) + G_{a_{i+1}}(k-q,m): q = 0,\dots,k, p = 0,\dots,V(T_{a_i})\},$$

$$\min\{F_{a_i}(q,p) + G_{a_{i+1}}[k-q-p(m-p), m-p]: q = 0,\dots,k, p = 0,\dots,m\}\}, \quad (6)$$

where e is the edge connecting a to a_i. Equation (6) is written for all non-leaf nodes $a_i \in V$ with $i < s$ (non-rightmost subtrees). For each rightmost subtree T_{a_s} we specify the initial condition

$$G_{a_s}(k,m) = \begin{cases} w_e + \min\{F_{a_s}(k,p): p = 0,\dots,V(T_{a_s})\} & \text{if } m = 1, \\ F_{a_s}(k-m+1, m-1) & \text{if } m > 1, \end{cases} \quad (7)$$

where e is the edge connecting a to a_s, and for every leaf a:

$$F_a(k,m) = \begin{cases} 0 & \text{if } k = 0, m = 1, \\ \infty & \text{in all other cases.} \end{cases} \quad (8)$$

To explain equations (5)–(8), we apply the same reasoning as in the preceding section for equations (1)–(4).

For equation (5), note that $T_a = T_{a_{1,s}}$ for any non-leaf node $a \in V$.

For equation (6) note that the edge e (which connects a to a_i) is the connecting edge between the subtrees T_{a_i} and $T_{a_{i+1,s}}$. There are two cases to compute the value of $G_{a_i}(k,m)$ based on the edge e, either we remove e or we have to keep it. If the value of $G_{a_i}(k,m)$ is achieved from the expression $w_e + \min\{F_{a_i}(q,p) + G_{a_{i+1}}(k-q,m)\}$, then besides removing the optimal edges from T_{a_i} and $T_{a_{i+1,s}}$ we should also remove the edge e and in this case a path surviving in $T_{a_{i,s}} = T_{a_i} + T_{a_{i+1,s}}$ either completely belongs to T_{a_i} or to $T_{a_{i+1},s}$. According to the definition of $G_{a_i}(k,m)$, m nodes are still connected to a and all nodes are inside the subtree $T_{a_{i+1},s}$, whereas at most $V(T_{a_i})$ nodes can be connected to a_i. If q paths belong to T_{a_i} exactly $k-q$ paths belong to $T_{a_{i+1},s}$. Hence by definition of F and G the minimum total weight of the edges removed from $T_{a_{i,s}}$ will be $G_{a_i}(k,m) = w_e + \min_{q,p}\{F_{a_i}(q,p) + G_{a_{i+1}}(k-q,m)\}$ where w_e corresponds to the weight of the edge e.

On the other hand, if the edge e is not removed, then a is connected to m nodes of T_a and a path in $T_{a_{i,s}}$ can be either completely contained in one of T_{a_i}, $T_{a_{i+1,s}}$, or partially contained in both subtrees because it passes through the edge e. If a_i is connected to p nodes of T_{a_i} and a is connected to $m-p$ nodes in $T_{a_{i+1,s}}$, the paths passing through a are exactly $p(m-p)$. If q paths survive in T_{a_i}, $k-q-p(m-p)$ paths survive in $T_{a_{i+1,s}}$ and, by definition of F and G, $G_{a_i}(k,m) = \min_{q,p}\{F_{a_i}(q,p) + G_{a_{i+1}}[k-q-p(m-p), m-p]\}$.

The initial condition (7) takes into account that, if the edge e (which connects a to a_s) is removed ($m=1$), the k surviving paths of $T_{a_{s,s}} = a + T_{a_s}$ must belong entirely to T_{a_s}, hence the minimum possible weight for the edges removed from $T_{a_{s,s}}$ will be $G_{a_s}(k,m) = w_e + \min_p\{F_{a_s}(k,p)\}$. On the other hand if the edge e is not removed ($m>1$), we must have $m-1$ nodes connected to a_s in T_{a_s} and the number of surviving paths is $k-(m-1)$. Thus $G_{a_s}(k,m) = F_{a_s}(k-m+1, m-1)$ follows.

The equation (8) says that since $a \in V$ is a leaf, the only possible condition is when $m=1, k=0$ and all other combinations of m and k are infeasible and are considered to have an infinite weight.

The optimal value, assuming the tree is rooted at node 1, is given by

$$OPT = \min\{k: F_1(k,m) \le W, k = 0, \dots, n(n-1)/2, m = 1, \dots, n\}. \quad (9)$$

The optimal solution is recovered by backtracking.

Proposition 2 *CEDP on a tree with unit connection costs and arbitrary edge weights can be solved by recursion (5)–(8) in $\mathcal{O}(n^7)$ time.*

Proof. For each node $a \in V$ there are at most $n(n-1)/2 + 1 = \mathcal{O}(n^2)$ values for k and $n+1 = \mathcal{O}(n)$ values for m; this gives $\mathcal{O}(n^4)$ values $F_a(\cdot,\cdot)$ and $G_{a_i}(\cdot,\cdot)$ to compute. The heaviest computation lies in equation (6), where $\mathcal{O}(n^2)$ values are possible for q and $\mathcal{O}(n)$ for p. Hence in the worst case a number of operations bounded by $\mathcal{O}(n) \cdot \mathcal{O}(n^2) \cdot \mathcal{O}(n^4) = \mathcal{O}(n^7)$ are required.

4 Conclusion

In this paper we investigated the critical edge detection problem over trees. We showed that the cases with unit connection costs and unit or arbitrary edge weights are solvable in polynomial time through dynamic programming approaches.

References

[1] Albert, R., Jeong, H., A. L. Barabasi, A. L.: *Error and attack tolerance of complex networks*, Nature, 406, 378–382 (2000).

[2] Arulselvan A., Commander C.W., Elefteriadou L., Pardalos P.M.: *Detecting critical nodes in sparse graphs*, Computers & Operations Research, 36, 2193–2200 (2009).

[3] Avidor, A., Langberg, M.: *The multi-multiway cut problem*, Theoret. Comput. Sci., 377 (1-3), 35-42 (2007).

[4] Borgatti, S.P.: *Identifying sets of key players in a network*, Computational and Mathematical Organization Theory, 12, 21–34 (2006).

[5] Buluc, A., Meyerhenke, H., Safro, I., Sanders, P., Schulz, C.: *Recent advances in graph partitioning*, in: Algorithm Engineering, Springer, 117-158 (2016).

[6] Chopra, S., Rao, M.R.: *On the multiway cut polyhedron*, Networks, 21 (1), 51-89 (1991).

[7] Chopra, S., Owen, J.H.: *Extended formulations for the A-cut problem*, Math. Program., 73 (1), 7-30 (1996).

[8] Costa, M.-C., Letocart, L., Roupin, F.: *Minimal multicut and maximal integer multiflow: A survey*, European J. Oper. Res., 162 (1), 55-69 (2005).

[9] Crucitti, P., Latora, V., Marchiori, M., Rapisarda, A.: *Efficiency of scale-free networks: Error and attack tolerance*, PHYSICA A, 320, 622–642 (2003).

[10] Dahlhaus, E., Johnson, D.S., Papadimitriou, C.H., Seymour, P.D., Yannakakis, M.: *The complexity of multiterminal cuts*, SIAM J. Comput., 23 (4), 864-894 (1994).

[11] Di Summa, M., Grosso, A., Locatelli, M.: *Complexity of the critical node problem over trees*, Computers & Operations Research, 38(12), 1766-1774 (2011).

[12] Fjallstrom, P.-O.: *Algorithms for Graph Partitioning: A Survey*, Linkoping University Electronic Press Linkoping, Vol. 3, (1998).

[13] Garg N., Vazirani V.V., Yannakakis M.: *Primal-dual approximation algorithms for integral flow and multicut in trees,* Algorithmica, 18, 3–30 (1997).

[14] Goldschmidt, O., Hochbaum, D.S.: *A polynomial algorithm for the k-cut problem for fixed k*, Math. Oper. Res. 19 (1), 24-37 (1994).

[15] Guo, J., Niedermeier, R.: *Fixed-parameter tractability and data reduction for multicut in trees*, Networks, 46 (3), 124-135 (2005).

[16] He, X.: *An improved algorithm for the planar 3-cut problem*, J. Algorithms, 12 (1), 23-37 (1991).

[17] Lalou, M., Tahraoui, M., Kheddouci, H.: *Component-cardinality-constrained critical node problem in graphs*, Discrete Applied Mathematics, 210, 150-163 (2016).

[18] Myung, Y.-S. and Kim, H.-J.: *A cutting plane algorithm for computing k-edge survivability of a network*, European Journal of Operational Research, 156(3), 579-589 (2004).

[19] Oosten, M., Rutten, J.H.G.C., Spieksma, F.C.R.: *Disconnecting graphs by removing vertices: a polyhedral approach*, Statistica Neerlandica 61(1), 35-60 (2007).

[20] Pothen, A.: *Graph partitioning algorithms with applications to scientific computing*, in: Parallel Numerical Algorithms, Springer, 323-368 (1997).

[21] Shen, S., Smith, J.: *Polynomial-time algorithms for solving a class of critical node problems on trees and series-parallel graphs*, Networks, 60(2), 103-119 (2012).

[22] Veremyev, A., Prokopyev, O., Pasiliao, E.: *Critical nodes for distance-based connectivity and related problems in graphs*, Networks, 66(3), 170-195 (2015).

[23] Veremyev, A., Prokopyev, O.A., Pasiliao, E.: *An integer programming framework for critical elements detection in graphs*, Journal of Combinatorial Optimization, 28(1), 233-273 (2014).

Syed Md Omar Faruk, et. al. "Polynomial time algorithms of Critical Edge Detection Problem on a tree."*IOSR Journal of Mathematics (IOSR-JM)*, 18(3), (2022): pp. 06-12.